

Open Research Online

The Open University's repository of research publications and other research outputs

Vector Signal Processors in Data Compression and Image Processing

Thesis

How to cite:

King, G (1990). Vector Signal Processors in Data Compression and Image Processing. MPhil thesis. The Open University.

For guidance on citations see [FAQs](#).

© 1990 The Author

Version: Version of Record

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

oro.open.ac.uk

Vector Signal Processors in
Data Compression and Image Processing

G.A.King

MPhil 1990

ProQuest Number: 27758418

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent on the quality of the copy submitted.

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



ProQuest 27758418

Published by ProQuest LLC (2019). Copyright of the Dissertation is held by the Author.

All Rights Reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

UNRESTRICTED

Thesis for the degree of
MPhil

by G.A.King BA (OU)

Vector Signal Processors
in
Data Compression
and
Image Processing

Related disciplines:

Electronics, Computer Engineering, Data Compression, Image Processing

Submitted August 1990

Date of submission : 30th August 1990

Date of award : 31st October 1990

Abstract

The objective is to evaluate the applicability of the Vector Signal Processor to real time signal processing for data compression or manipulation. Particular emphasis has been placed on its role as a co-processor and the contribution that it might be expected to make during joint activities with the host.

These activities would have the combination used as the embedded computing subsystem of a FAX machine or as an image processing unit in desk top publishing. In these cases the hypothesis is that the Vector Signal Processor would act as an accelerator for many computationally intensive applicable processes.

After a review of current data compression techniques and of specialised architectures which may also be appropriate it is concluded that the Vector Signal Processor is the best option available. The operational details are then discussed. In order to be able to approximately compare experimental results with other workers a benchmarking exercise is undertaken.

Following this is the core of the study which details schemes for data compression of data sources involving character symbols, line drawings, and grey scale pictures. This involves pattern matching and substitution, Transform coding and quadrees.

New encoding procedures are suggested based on Morse code for the secondary encoding of symbols and on Delta modulation for quadrees. Image entity manipulation is discussed followed by some speculative work on neural networks and error control coding.

It is concluded that some processes are well served by the Vector Signal Processor but that the lack of conditional decision making and the difficulty of performing certain arithmetic functions make the processor unwieldy in its necessary host interactions.

Vector Signal Processors in Data Compression
and Image Processing

Contents

Abstract	
1.0	Introduction 1
1.1	Context 1
1.2	Study areas and rationale 4
1.3	Summary of work 4
1.4	Innovation and original ideas 7
2.0	Standard techniques and innovation in time and binary source coding. 10
2.1	Introduction 10
2.2	Shannon Fano code 10
2.3	Huffman Code 14
2.4	Run length coding 15
2.5	CCITT group 3 codes 15
2.6	Binary non-consecutive one code 19
2.7	Lynch Davisson code 20
2.8	Morse code 22
3.0	Computer Architectural Considerations 27
3.1	Introduction 27
3.2	Survey 27
3.3	Alternative architectures 29
3.3.1	Peripheral vector processors 29
3.3.2	Digital Filters 30

3.4	Vector Signal Processor organisation	30
3.4.1	VSP program style	33
3.4.2	Flow control	34
3.4.3	The scaling system	35
4.0	Text compression benchmarking rationale	38
4.1	Philosophy	38
4.2	Experimentation	39
4.2.1	The benchmark encoding scheme	40
5.0	Pattern Matching and Substitution	46
5.1	The Method	46
5.1.1	Matched filters	49
5.1.2	Implementing the matched filter	50
5.1.3	n-valued model	53
5.1.4	Using the Vector Signal Processor	53
5.2	Generating autocorrelation functions	53
5.3	The screening stage	56
5.4	The main statistical technique	60
5.5	Experiments and results	64
5.6	Secondary encoding	72
5.7	Implementing fast correlation	79
5.7.1	Overlap and discard	81
5.7.2	Convolution or correlation?	81
5.7.3	The VSP program	83
6.0	Transform Coding	86
6.1	Introduction	86
6.2	Principles and rationale	86

6.3	Transform coding and the vector signal processor	88
6.3.1	The Hotelling transform	88
6.3.2	The Fast Fourier Transform	92
6.3.3	Architectural factors	98
6.3.4	2-D FFT	99
6.3.5	2-D FFT implementation	101
6.3.6	FFT instruction parameters	103
6.3.7	The Fast Cosine Transform	108
6.3.8	VSP implementation of the FCT	110
6.4	Transfer Coding	113
7.0	Quadtree encoding + Delta modulation	118
7.1	The application	118
7.2	Quadtree representations	118
7.2.1	Quadtree addressing scheme	121
7.3	Delta modulation	122
7.4	VSP implementation	126
8.0	Image manipulation using the Vector Signal Processor	128
8.1	Introduction	128
8.2	Object representation	128
8.2.1	Edge detection	128
8.2.2	Filtering	130
8.3	Chain codes	133
8.3.1	VSP chain code implementation	137
8.4	Fourier Descriptors	137
8.4.1	Fourier Descriptor manipulation with the VSP	139

9.0	An alternative classifier - the Neural Network	144
9.1	Introduction	144
9.2	Basic concepts	144
9.2.1	Structure and components	144
9.2.2	Operation	146
9.2.3	The Hopfield net	148
9.3	Net simulation with the VSP	148
9.3.1	Matched filter models	151
9.3.2	Matched filter equivalence	151
9.4	Implementation with the VSP	154
10.0	Error Control Coding	157
10.1	Underlying principles and issues	157
10.2	Appropriate error control methods	160
10.3	Essential background	161
10.3.1	Galois fields	161
10.3.2	Cyclic codes	162
10.3.3	Polynomial representations	162
10.3.4	Generator matrix	163
10.3.5	Parity check matrix	165
10.3.6	Syndrome	165
10.4	Bose-Chaudery-Hocquenghem codes	167
10.5	BCH implementation	168
10.6	Error detection - CRCs	169
11.0	Conclusions	172
11.1	Scope	172
11.2	Relevance, currency and validity	172

11.3 Signal processing devices and data

compression	175
Appendix A Published Papers	182
Appendix B VSP programs	235
Appendix C VSP instruction set	247
Appendix D Morse compression program design documentation	253
Appendix E EPLD matched filter design data	268

gak 10/90

1. Introduction

1.1 Context

The general objectives are to quantify the appropriateness and performance advantages of using a Vector Signal Processor in real time embedded data compression and image processing functions. The Vector Signal Processor is a microprocessor class device designed to act as a co-processor to a standard CPU.

This study seeks to review techniques used to implement data compression and manipulation in a context such as a publishing database or FAX.

The contention is that the use of devices optimised for vector processing would bring benefits especially in real time applications. Vector orientated processors are found in large supercomputer environments (e.g. Amdahl-Cray complexes) but the Vector Signal Processor is a rare example of a device specifically devised to operate at the dedicated computer level.

Such devices as Vector Signal Processors, and Digital Signal Processors form the core of the work. Consequently an overlap between signal processing and specialised computer architectures will feature strongly.

Many entities that are the subject of data compression or manipulation may need to be "recognised" or "processed". There are links between data compression, image processing, and pattern recognition.

Dougherty and Guardina^[1] make further links by exemplifying the purpose of image processing.

"Image processing is related to computer vision, robotics, and artificial intelligence - with a mathematical

representation of the image in hand we seek to extract information which can be operated upon by a decision algorithm."

These thoughts are echoed by Gevarter^[2] in his classification of application areas in Artificial Intelligence (AI).

Dougherty and Guardina go on to analyse image processing as having four levels of transformation :-

- Level 0 Image representation
- Level 1 Image to Image transformations
- Level 2 Image to Parameter transformations
- Level 3 Parameter to Decision transformations

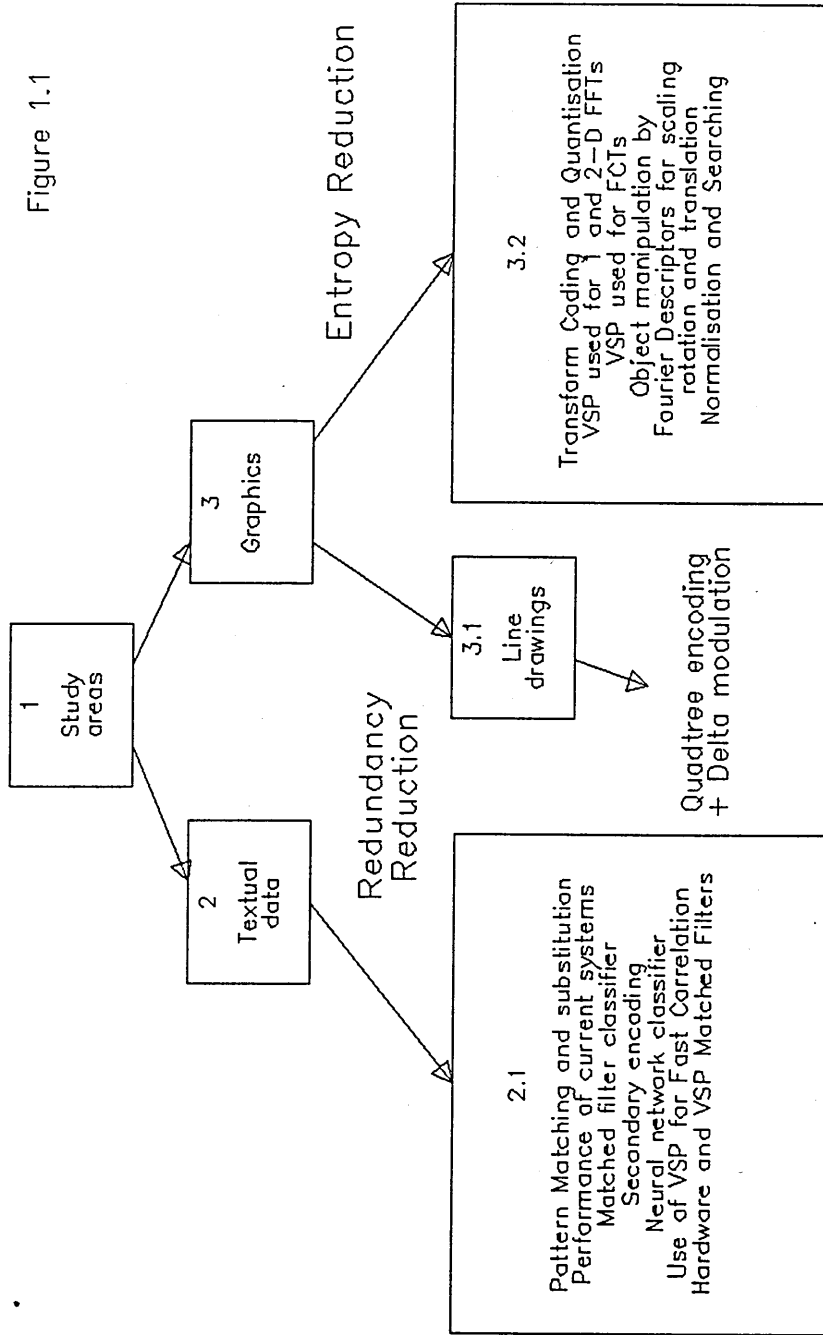
If an image is encoded with a given symbol set it is possible to transform the symbol set to one of lesser cardinality without losing any information, or at least with insubstantial loss.

Compression can be achieved at levels 0,1,2. Since it might be that we reduce the number of feature parameters in a feature vector, compression often involves vector to vector transformations.

Wechsler^[3] states that data compression is concerned with reducing load on resources and that pattern recognition techniques can be used to "facilitate the task". Wechsler offers a model which generalises the processes involved to a prescriptive sequence.

- * Pre-processing
- * Feature extraction

Figure 1.1



* Classification (Pattern recognition)

* Post-processing

A synthesis of these thoughts may be regarded as the watershed from which this study flows.

1.2 Study Areas and Rationale

Figure 1.1 is a structure chart showing the main study areas and the techniques that are postulated to be most appropriate to the application of Vector Signal and Digital Signal Processors.

A study of data compression requires an understanding of the background issues and terminology. It also requires that any computing engines used are optimised to the task.

Apart from this a question of comparing like with like arises. If a particular compression method is to be used in a given environment it is necessary to be able to determine the performance of the method relative to the state of the art. In the case of this work the standard methods, computer architectural considerations, and the quantifying of performance are covered before experimental ideas are developed.

The aims are to investigate the applicability of digital signal processing techniques to those data compression ideas that are amenable.

1.3 Summary of work

Figure 1.1 shows that data sources were expected to generate three data types:- textual, line drawings, and grey scale photographic images.

Techniques applicable to each data type were evaluated for implementation with specialised digital signal processing

architectures. A number of architectures exist and a study of the options is carried out in Section 3.0 which concludes that the most practical designs are peripheral vector signal processors and digital signal processors.

If a technique appeared theoretically appropriate to these architectures then implementation was attempted. Results were checked by simulation with high level language programs or a maths package.

This approach was taken with the topics covered by Sections 5.0, 6.0, 7.0 and 8.0.

Of considerable interest in the area of pattern recognition is the neural network. Section 9.0 speculates how the vector signal processor could be used to produce a classifier of this type. Practical achievement was precluded because the best route to realisation requires arrays of matched filters. The numbers needed were outside resource limits.

Error control coding, the subject of Section 10.0, is similar in its speculative nature and although individual ideas were tested by simulation a comprehensive implementation was not attempted.

It was established at an early stage that where symbols or other entities are represented in bit mapped form the resolution in dots per inch can affect the achievable compression. As an example, consider an alphabetical symbol such as "a". It may be represented within an 8X16 character cell and be perfectly well understood. It may also be represented at a resolution of 300 dots per inch, as is used in typical image scanners. The latter

has greater compression potential and many other workers have produced results based on this resolution.

Owing to resource limitations this study is based on the IBM PC textual and CGA standards. The effectiveness of any new ideas can be measured by comparison with previous results, and a benchmarking phase documented in Section 4.0 enabled valid comparisons to be made. Section 4.0 involved data from 240 screens captured with a variety of content from dense textual to sparse line drawings.

Capture utilities were created to allow this data acquisition and details are given within the section.

Three sets of experiments were carried out using variations on standard encoding techniques to obtain the benchmark results.

The main technique chosen for textual compression implementation with the VSP was symbol matching and substitution, details of which are given in Section 5.0. Experimental work was necessary to evaluate the performance of the screening algorithm and to test the main statistical matching technique. In the latter case 26 autocorrelation functions and 676 cross correlation functions were generated from an alphabetic character set. All functionality was modelled in software.

After typical compression ratios were obtained, fast convolution/correlation in the frequency domain was implemented with the VSP. Results were checked against the original high level language software.

The programmable matched filter demanded by the method was first created in software and then implemented in a dedicated

electrically programmable logic device. Details of the semi-custom design are contained in Appendix E.

Only the Hotelling transform was not amenable to VSP implementation in Section 6.0. Example VSP programs produced during study of transform coding of screen images is provided in Appendix B.

Quadtree encoding and delta modulation discussed in Section 7.0 were developed as an alternative to run length or Huffman coding for line drawing compression. The VSP could not cope with the processing required but nevertheless encouraging results were achieved. The coding scheme was devised from scratch and tests were made on specimen sub-pictures to quantify potential compression performance.

The role of the VSP in manipulation of graphical data objects was investigated because of the likelihood of a system needing to compress images for storage purposes also being involved with editing those images. As examples consider a graphics workstation for CAD or a desk top publisher system.

1.4 Innovation and original ideas

i) The main statistical matching technique used in the pattern matching and substitution section is a well known technique for analysing gas chromatograph results, and is also used in production engineering to identify systematic faults in the operation of production machines. No evidence was found for its use in the context described here. Other, less sophisticated options have been used and one of these is compared and discussed.

ii) A typical quadtree encoding scheme does not optimise in terms of data compression defining, as it does, the addresses of significant pixels. A delta modulation scheme was devised because it was recognised that a "locality of reference" principle applied in the address lists. It was further recognised that line drawings with their large proportion of vertical and horizontal lines very often produced adjacent quadrants which were identical. The delta modulation scheme was extended to include special repeat codes. Arbitrary unused and unambiguous codes were chosen for this purpose.

The overall combination of ideas is thought to be original. Details of the advantages of the scheme are given in the section.

iii) To achieve a required compression ratio it is not unusual to cascade different techniques. A secondary encoding scheme was applied in the case of the pattern matching and substitution scheme. Instead of signalling ASCII codes for the character symbols in the text, it was decided to try to take advantage of the differing occurrence frequencies of the symbols.

Ideally this involves the use of a variable length code such as Huffman or Shannon-Fano. There is a need for an analytical phase to determine the allocation of the codes in this scheme. To avoid this overhead it was remembered that Morse coding uses statistical data derived from occurrence frequencies in the english language. The codes allocated are fixed. The code is variable length. An adaption would be necessary to give a computer implementation. A scheme was devised and its performance compared to existing standard methods. The scheme is thought to be innovative.

A number of standard encoding schemes are explained in Section 2.0.

References

- [1] Dougherty, E.R. and Guardina, C.R., "Image Processing, Vol.1, Geometric, Transform, and Statistical Methods", Prentice Hall 1987.
- [2] Gevarter, W.B., "Intelligent Machines", Prentice Hall 1985.
- [3] Wechsler, H., "Invariance in Pattern Recognition" in "Advances in Electronics and Electron Physics", Vol.69, pp261-320, 1987.

2.0 Standard techniques and innovation in time and binary source coding

2.1 Introduction

In subsequent sections involving techniques that are redundancy reducing, references are made to methods that have become standards. In some cases these standards have been used as a benchmark against which performances can be measured.

Since this work is concerned with sources that are mainly binary, only those techniques useful for direct compression of that data type are considered.

The term "time" in the heading refers to the fact that additional information may need to be sent or stored along with the compressed data to aid in the reconstruction process. This additional information is time or position information which needs to be coded to minimise the overhead that it represents.

The section concludes with a newly devised innovative encoding scheme using Morse code as a basis. The scheme is included here to provide a reference source backing up the experimental data in Section 5.0.

2.2 Shannon-Fano Code

Fano^[1] describes a procedure for the generation of a variable length code with codes allocated according to the frequency with which the symbols to be encoded appear.

A feature of the scheme is that the codes are instantaneously decodable because none are prefixes of any other.

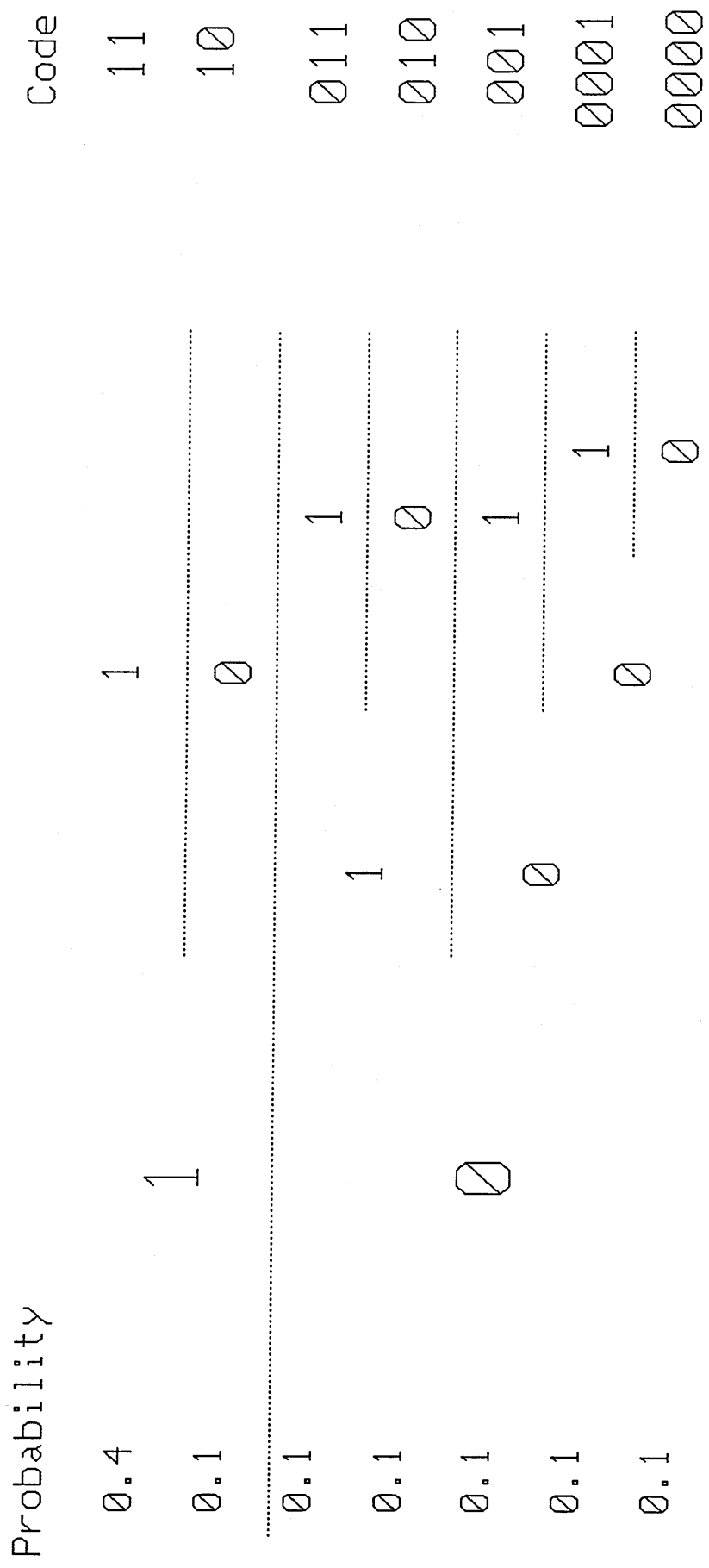


Figure 2.1

Assuming a set of symbols to be encoded, the procedure is as follows :-

Arrange the elements of the set in descending order of probability.

Split the set into two groups having approximately equal total probability, and allocate a "0" to the first group and a "1" to the second.

Iterate the process until the groups are reduced to single elements.

As an example consider the following message set and associated probabilities.

message	probability
1	0.4
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1

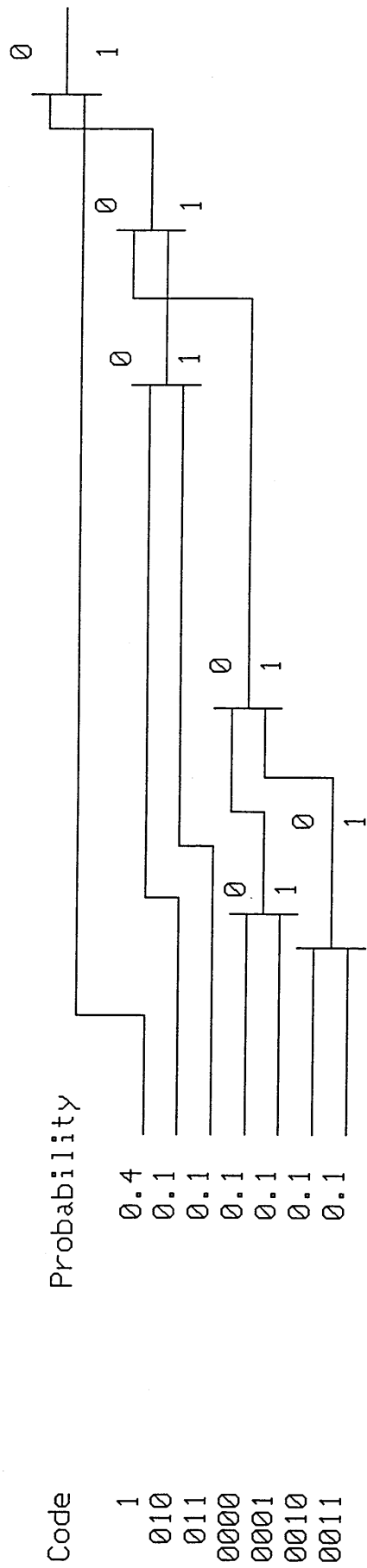
Figure 2.1 shows the derivation of the code words. The full communication encodes as :-

1	3	1	2	5	6	4	1	7	1
11	011	11	10	001	0001	010	11	0000	11

If each message were an eight bit ASCII character the use of Shannon Fano code results in a compression because the average code length per message is 2.7 bits.

The code is easily decoded according to the following rules.

Figure 2.2



1st digit "1", code is 2 bits. First three digits "0", code is 4 bits. Other cases define a 3 bit code. A codebook can be used to identify individual symbols, but clearly a codebook must be generated during encoding and sent or stored for the future use of the decoding process. Long code words are generated if a large number of low probability symbols appear in the message set.

2.3 Huffman Code

This coding scheme is a development of the Shannon-Fano code which has the benefit of always being more efficient. Efficiency is defined in this context in terms of the ratio of the message entropy to the average code word length. The code is developed and applied in Huffman [2].

The coding procedure again requires analysis in order to arrange the source probabilities in descending order, but the process continues :-

Combine the two lowest probabilities putting the highest on top in a code tree arrangement.

Continue until unity is reached and then allocate "0"s and "1"s, "0"s to the upper branch members and "1" to the lower.

Read off the code by tracing the path right to left and noting the sequence. Consider the same messages and probabilities that were used for the Shannon Fano example. The code tree resulting is shown in figure 2.2.

The fully encoded communication is :-

1	3	1	2	5	6	4	1	7	1
1	011	1	010	0001	0010	0000	1	0011	1

This gives a slightly improved performance when compared with Shannon Fano since the average code length is 2.6 bits.

2.4 Run Length Coding

Consider bit mapped displays that are scanned. Whatever information the display carries, when a line of pixels is examined the following is typical :-

```
0000000011111111100000001111111100000011000000011100000000000000
```

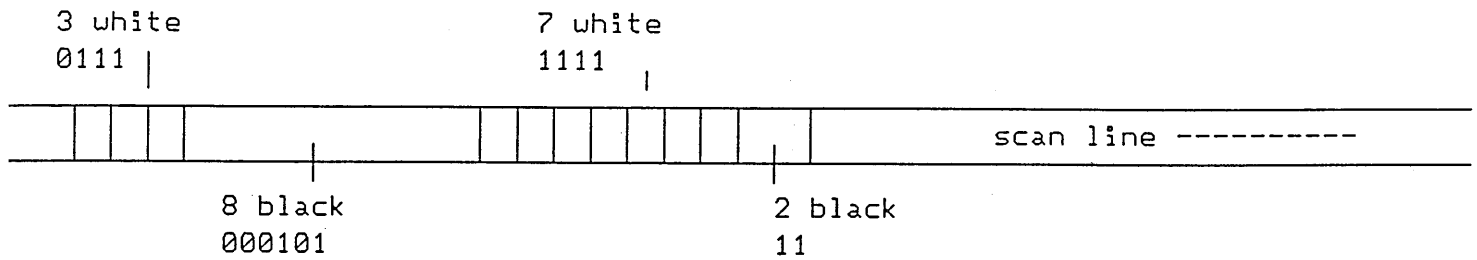
Some sources have greater sparsity than others and the codes that have been devised may encode the run lengths of both "0"s and "1"s or alternatively just that which has the greatest frequency. It is often the case that "1"s are considered non-redundant and "0"s redundant (or vice-versa). This gives rise to the notion that a code can be classified as encoding n-r,r,or n (non redundant, redundant, redundant, or non redundant) run lengths. In the literature an alternative classification may be found as in Lynch^[3]. This alternative calls n-r "total information time codes" because the complete n-r sequence is involved and the term "partial information time codes" applies to the other two possibilities.

In the example pixel line above the n-r run lengths are :- 8,9,7,8,6,2,7,3,14. Clearly an analysis would find runs of 7 and 8 more probable than the others. This leads to the idea of applying say, Huffman code to the run lengths.

2.5 CCITT Group 3 codes

This digital facsimile set of compression recommendations are described in CCITT^[4]. Two algorithms are possible, the 1-D

Modified Huffman 1-D



Coded data 0111000101111111

- * Each scan line is analysed
- * Short codes are allocated on the basis of run length occurrence
- * Codes are unique variable length or fixed length (beware of negative compression)
- * Coded/uncoded flag
- * Compression 5 to 15:1 (CCITT tests)

Figure 2.3

or the 2-D.

The 1-D system involves counting the run length of consecutive same colour pixels and then counts the next pixel group of the other colour (black or white) and so on. The run lengths are encoded using a modified Huffman code.

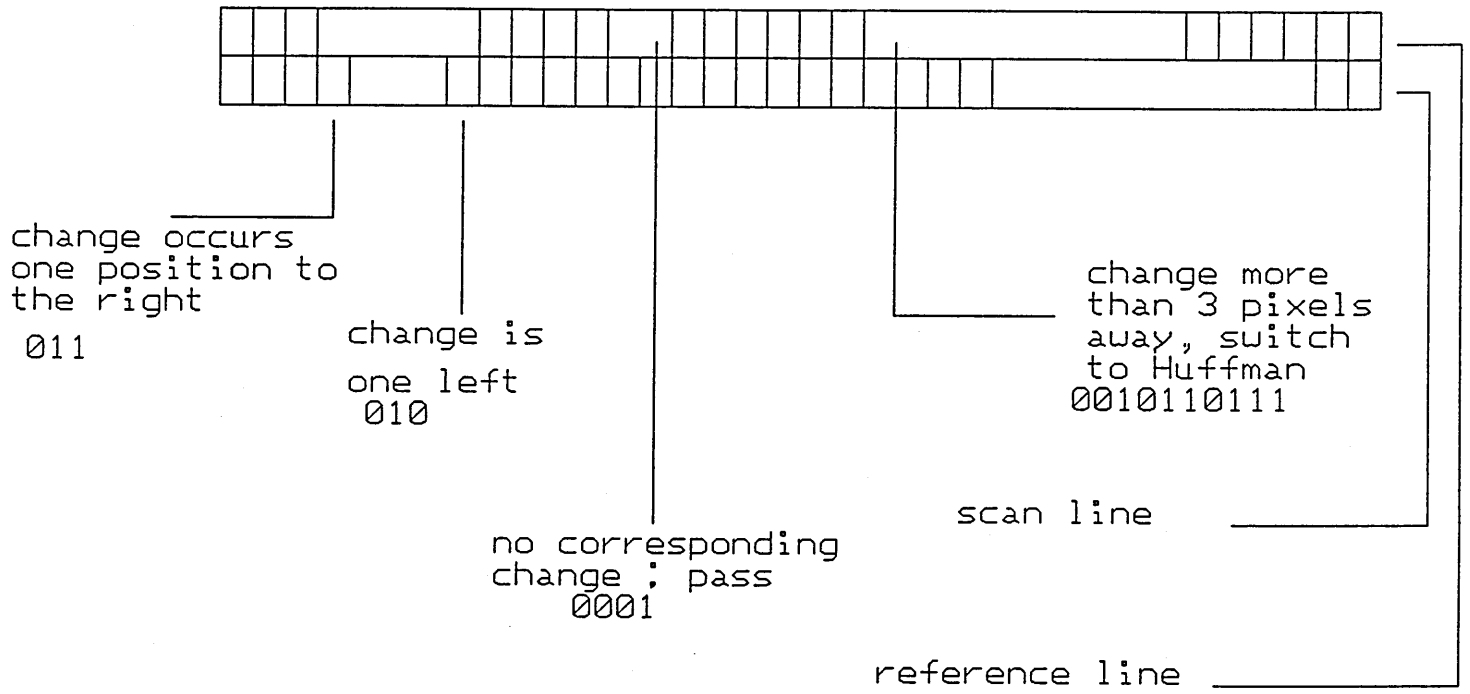
An assumption is made that each scanned line begins with at least one white pixel. If this is not so, that is, the line begins with a black pixel then the compressor actually sends the code for a zero length white run first. The assumption is not unreasonable when the typical document that might be faxed is considered. Figure 2.3 illustrates the method which is described by Fuchs^[5]. The results obtained with varying documents or diagrams is given by Bodson et al^[6] as between 5:1 and 15:1.

The 2-D algorithm is called "Relative Address Designate" or READ coding. This operates by encoding scan line pixels based on pixel positions and colour changes in the previous line. When a change occurs in a newly scanned line directly below a change in the previous (reference) one, then a single code bit represents the change. Other codes cope with changes that are no further away than 3 pixels from a change in the reference line. For differences further apart than this a special code is generated and a switch is made to the 1-D system. The method is illustrated by figure 2.4. Bodson et al quantify compression for a range of sources as between 7:1 and 50:1.

In either algorithm, the compression achieved is a function of the type of document (text, drawing, schematic), and the resolution with which it is scanned.

An interesting difference between the two options is in the

Relative Address Designate Code



Coded data 01101000010010110111

Figure 2.4

effect that data corruption has. An error in the 1-D code only affects that scan line. In the 2-D case errors can propagate through the entire document.

2.6 Binary Non-consecutive One Code

Wai-Hung Ng^[7] described a scheme for coding redundant sample run lengths using variable length codewords which may have consecutive "0"s, but not consecutive "1"s. The code is shown below :-

Run Length	Code (BNO)
1	0
2	1
3	00
4	01
5	10
6	000
7	001
8	010
9	100

and so on.

The data is coded in two parts the non-redundant samples and the BNO code for the redundant runs. The two are separated by the use of a special code word containing consecutive "1"s. The format of the special word is :-

11111111....0 There are $2n$ "1"s where n is the number of non-redundant samples that follow the run encoded by the previous BNO code.

This yields a structure as follows :-

(BNO)1111110(A₁, A₂, A_n) (BNO)111111..0 etc

Because the structure encodes both redundant and non-redundant run lengths it is a total time information code. Best performance is achieved when non-redundancy is relatively sparse.

2.7 Lynch Davisson Code

Suggested by Lynch and Davisson [8,9] this scheme encodes the pattern of the sequence to be coded. The pattern is represented

by a number given by :-
$$T = \sum_{j=1}^q \binom{n_j - 1}{j}$$

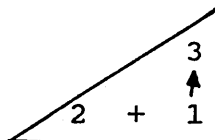
Where q is the number of non-redundant samples, $1 \leq q \leq N-1$
 N is the number of bits.

n_j is the bit number of the non-redundant bit $1 \leq n_j \leq N-1$
 j is the index of the non-redundant sample $1 \leq j \leq q$

The notation identifies code number positions in a matrix. The matrix is generated using a Pascals triangle rule and is reproduced below.

10	45	120	210	252	210	120	45	10	1	0	0
9	36	84	126	126	84	36	9	1	0	0	0
8	28	56	70	56	28	8	1	0	0	0	0
7	21	35	35	21	7	1	0	0	0	0	0
6	15	20	15	6	1	0	0	0	0	0	0
5	10	10	5	1	0	0	0	0	0	0	0
4	6	4	1	0	0	0	0	0	0	0	0
3	3	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Each element is calculated....



Consider now a 16 bit pattern 0010100000000000

$$T = \frac{3-1}{1} + \frac{5-1}{2} = 2+6=8$$

since element (2,1)=2 and element (4,2)=6

The number of non-redundant bits must be used in conjunction with the codeword in the decoding process, and so storage or transmission would for this example be :- 2,6

Element (n,r) may also be generated by the equation :-

$$n,r = \frac{n!}{r! \cdot (n-r)!}$$

The value of T is unique for each sequence of q bits distributed over N-1 positions.

Decoding also involves use of the matrix. Starting with the qth column the T value is offered up to the matrix until it is greater than or equal to an element, but less than the next higher element in the column. Add 1 to the row. Subtract the identified value and continue until the first column is reached.

According to Lynch the number of bits required for q and T is:-

$$k + k \quad \text{where } k = \log_2 N \quad \text{and } K_T = \log_2 \left[\binom{N-1}{q} \right]$$

2.8 Morse Code

The Morse code was originally devised in a telecommunications context and was defined after extensive research into character frequency in the English language. The code employs two code elements, the dot and the dash which are differentiated in duration. The dash duration being three times longer than the dot.

The code is a variable length code representing characters by various combinations of the two code elements. The two most common letters in English are "e" and "t" which are coded as a single dot and dash respectively.

In the case of Huffman code the variable length is inversely proportional to the probability of the message. Underlying the Morse code is the same general relationship but it is applicable to English text specifically.

Whilst researching the secondary encoding possibilities for the pattern matching and substitution a scheme utilising Morse principles was devised. In the scheme a dash is substituted for by a "1" and a dot by a "0". Morse was intended as a real time system and in the context of data compression some time or positional information had to be introduced. The results are given in Section 5.0 and the Morse code is listed in table 2.1.

The need for positional information is clear when it is recognised that the code is not a pre-fix code, that is some codes are pre-fixes of others so that ambiguity is possible i.e. 100000 could be "nh" or "ds" or "t5" or "bi" when "the" is intended.

Table 2.1

Character	Morse	Binary	Character	Morse	Binary
a	.-	01	8	---	11100
b	.-...	1000	9	---.	11110
c	-.-.	1010	0	---..	11111
d	-. .	100	/	---..	10010
e	. -	0	.	---..	010101
f	..-. .	0010	,	---..	110011
g	---.	110	(---.	10110
h	0000)	---.	101101
i	..	00	-	---..	100001
j	.-.-	0111	=	---..	10001
k	-. -	101			
l	.-..	0100			
m	--	11			
n	-. -	10			
o	---	111			
p	.-.-.	0110			
q	-. -.	1101			
r	.. -	010			
s	... -	000			
t	- .	1			
u	.. -	001			
v	... -	0001			
w	-. -.	011			
x	-.-.	1001			
y	-. -.	1011			
z	--.	1100			
1	01111			
2	00111			
3	00011			
4	00001			
5	00000			
6	10000			
7	11000			

The scheme is a two part code comprising the morse section and a section involving morse element counts in a fixed field of 3 bits. The example above is encoded unambiguously as :-

100000 - Morse section

001 100 001 - element counts (normally run together - 001100001)

The total code size is 15 bits compared with 24 bits for the ASCII representation. The compression ratio is slightly reduced by the need to explicitly indicate an inter-word space. Since no Morse entity exceeds 6 code elements, the code 111 (7) in the element count is used for the purpose.

One further overhead is generated when marking the end of the Morse section and the beginning of the element count section. A code previously unallocated such as 11011 (normally a French accented character) has been used. Normally, coding maximises the amount of text within the Morse section in order to reduce the effect on the compression ratio.

The format of the coding is as follows. Because the example is relatively short the result is well below what can be achieved with significant quantities of text and simply serves to illustrate the scheme.

Text :-

"the optimum noiseless codes"

Morse section :-

10000011101101001100111101110000000100000000

element count :-

001100001111011100001010010011010111010011010011001100001011011

111100011011001011

separated by :- 11011

Total bits :- 133

Total bits as ASCII code :- 216

Compression :- 1.63:1

Results obtained during benchmarking in Section 4.0 using Huffman techniques combined with run length coding were comparable for congested text. The analytical pass necessary to establish the probabilities for the Huffman scheme is not necessary in the Morse case. The Morse system should be substantially quicker in operation.

Using Morse applied in this way, together with the adaptations necessary for binary working are thought to be original. Since the data encoding and the time/positional information can be signalled or stored separately the new scheme is a total time information code.

References

- [1] Fano, R.M., "The transmission of information", Technical report no. 65, M.I.T Research Lab of Electronics, 1949.
- [2] Huffman, D.A., "A method for the construction of minimum redundancy codes", Proc IRE, vol 40, pp 1098-1101, September 1952.
- [3] Lynch, T.J., "Data Compression techniques and applications", Van Nostrand Rheinhold, 1985.
- [4] CCITT Recommendation T-6, ITU, Geneva.
- [5] Fuchs, P.M., "Compressing data conserves memory in bit mapped displays", EDN, pp 173-188, October 1986.

[6] Bodson, D; Urban, S.J.; Deutermann, A.R; Clarke, C.E., "Measurement of data compression in advanced group 4 facsimile systems", Proc IEEE, vol 73, No.4, pp731-739, April 1985.

[7] Ng, Wai-Hung, "Binary non consecutive one code for time tag data compression", Proc IEE, vol 118, No.10, pp1358-1360, October 1971.

[8] Lynch, T.J. "Sequence time coding for data compression" Proc IEEE, vol 54, No.10, pp1490-1491, 1966.

[9] Davisson, L.D., "Comments on 'Sequence time coding for data compression'", Proc IEEE, vol 54, No.12, p2010, 1966.

3.0 Computer System Architectural Considerations

3.1 Introduction

This section reviews some of the possible computer system architectures considered for use in image data compression.

The most suited architecture is identified as being the Vector Signal Processor (VSP), and the operational details of the VSP are discussed.

3.2 Survey

Many image processing functions are commonly carried out in the spatial domain. Consider the example of pattern matching used in the context of the compression strategy for textual characters described by Holt^[1]. The scheme uses template methods, producing an error pel map by primitive XOR operations at bit level in the image.

Such methods always generate the need for considerable computing power. It is a natural consequence of the 2-D spatial distribution of the image to map the pels into an array of processors, one processor per pel. This kind of thinking only requires simple processors of the bit serial type, and the array is classified according to Flynn^[2] as SIMD (single instruction multiple data stream).

The idea that an array of simple processors execute simultaneously a broadcast sequence of instructions allows segments of the image to be processed in parallel as described by Fountain^[3], and Pass^[4].

Whilst SIMD is an appropriate technology for simple processes performed by pel mapped bit processors, there are forms of processing, mainly concerned with higher level functions that are abstract in terms of the image data. Examples of the former simple processes are image filtering and feature extraction. An important type of higher level task is pattern recognition. The SIMD structure is not well suited to this and the requirements of higher level image tasks are MIMD as described by Flynn^[2].

Multiple instruction multiple data stream systems can be organised in a number of different ways. A common memory can be shared by a number of sophisticated processors, alternatively each processor can have local memory and the image can be distributed among them. In the latter case it becomes most important that high speed communications and a connectivity scheme be implemented. Where memory is common high speed communication is likely to be achieved by means of a parallel bus, and the system is classified as "tightly coupled". The distributed option is known as a "loosely coupled" system. The advantages and problems associated with these choices are discussed by Edwards^[5].

The conclusions reached by Edwards^[5] indicate that the communications bottlenecks in MIMD systems have not produced solutions where throughput is extremely high. Another problem concerns the sharing of tasks by a master processor and the need for a satisfactory synchronisation of task completion. Again this provides a throughput constraint.

Conversely, SIMD architectures have become widespread and have good performance at the level of working discussed above.

Use in data compression for databases and for transmission schemes does infer the higher level of operation.

SIMD is not particularly suited to the application and MIMD may not reach the performance criterion even though costs would be high. This applies particularly when dedicated systems are considered.

SIMD arrays are largely designed to act as co-processors for mainframes and minicomputers. This is evidenced by the survey carried out by Kittler and Duff^[6]. This survey also identifies MIMD difficulties having been solved by machines able to switch classification. The complexity inherent in both these options does not assist in developing systems for microprocessor hosts working magnetic or optical disc technology at the PC workstation level.

3.3 Alternative architectures

3.3.1 Peripheral Vector Processors

Rather than arrays of processors, with all their attendant costs, the most promising alternative is a processor that is optimised to operate on arrays. This could provide a cost effective co-processor within a PC type host. The question posed is can significant and satisfactory speeds be obtained?

This option, a Peripheral Vector Processor, is a notion that existed for some time and a number have been designed. A survey by Karplus^[8] lists several examples. In general, these processors are classified as Horizontal architectures since they use uncommonly long microcode words with many fields. Each field controls adders and multipliers in a way that makes resources

available in parallel. Another unusual feature of this type is the fact that the Assembler instructions specify the microcode level directly. Pipelining is used extensively during vector operations and layouts specialised to certain operations are common.

The Vector Signal Processor is a typical peripheral vector processor and has been used extensively in this research.

3.3.2 Digital Filters

Convolution and Correlation are never far from mind in image processing environments. The interesting thing is that an FIR digital filter may easily perform spatial domain convolution by virtue of its configuration. Digital Filter systems are optimised to perform sum of products operations and some designs are reconfigurable and flexible. These Digital Filter Processors (DFPs) are based on the parallel operation of a number of cells. Extra parallelism in terms of recursive operation is described by Brumfitt [7].

3.4 Vector Signal Processor Organisation

The Vector Signal Processor (VSP) is designed for use as a co-processor and has a bus structure and timing regime suitable for operation in a manner similar to microprocessor peripherals with a DMA interface.

The architectural philosophy for the microword organisation has been approximately categorised as "horizontal" by Flynn [2] and is comprehensively described by Dasgupta [9].

The main features of this structure are :-

1. It enables different resources (e.g. functional units, data paths) in the micromachine to be controlled independently. i.e. A single microinstruction may specify concurrent operations.
2. It leads to relatively large microword lengths.
3. Programmers can exercise control of parallelism at machine operation level.

The characteristics of the VSP instruction set are :-

1. It operates at algorithm level - it is programmed at functional level.
2. Instructions are Vector orientated.
3. Instructions are slanted towards DSP problems (e.g. FFT butterflies, Magnitude Squares)

The VSP is capable of tightly coupled co-processor action, which implies shared memory and a DMA bus structure. Two running modes are possible, Master or Slave. The latter demands that the host is responsible for delivering instructions to the VSP's FIFO instruction buffer. Up to 12 instructions can be queued. The VSP FIFO and its other internal registers are memory mapped. As a Master device the VSP fetches and executes its own instructions once it has been "started" by the host. Action is invoked by the host writing a start address into the VSP's instruction base/start register.

Alternatively, a start can be initiated by a JUMP indirect instruction being written to the instruction FIFO. The format of the JUMP instruction may be :-

```
JMPI RS:0,EI:0,MBA:START_ADDRESS;
```

This presumes the host has pre-loaded START_ADDRESS with

Table 3.1 VSP instructions and fields

Full details in Appendix C.

Instruction	Meaning
JMPI	Jump indirect via address given
NOP	No operation
STI n	Store internal register n
LD	Load array into internal ram from base address specified
ST	Store array in internal ram to base address specified
FFT	Perform a fast fourier transform using parameters supplied
MGSQ	Calculates the square of the magnitude of an internal vector

Field	Meaning
MBA	Memory base address
NMPT	Number of points in array
NMBT	Number of butterflies (FFT)
LPS	Last pass separation (FFT)
FPS	First pass separation (FFT)
MDF	Memory data format (real/imaginary)
ADF	Arithmetic data format (FP)
RS	Internal ram section no.
RV	Reverse data ordering (if 1)
EI	Interrupt host when done (if 1)
STR	Start register (used with STI, with NMPTs to dump n registers)

the starting address of the VSP program. The processor performs an indirect jump (JMPI) via this address. MBA is a term used to refer to the address of the beginning of an array, or as in this case of a single entity. MBA stands for "Memory Base Address". The operand RS defines the requirement for internal concurrent action. Here, RS:0 precludes this. If RS:1 were used with a memory instruction then the VSP would progress any available arithmetic instruction using a separate section of internal RAM for scratchpad while concurrently executing the memory instruction. The EI field allows the programmer to generate an interrupt to the host processor after any VSP instruction. This mechanism makes it possible for the host to interact by providing conditional testing outside the ability and specialisms of the VSP. Details of the VSP instruction fields are given in Appendix C and a table of those instructions and fields used in this section are summarized in table 3.1.

In implementations the slave mode was rejected as too slow since it relies on the host speed. In dedicated systems the host may be a simple microcontroller. The master choice allows VSP programs to run to completion, or until an interrupt is generated by an instruction containing the field (EI:1).

The nature of VSP programs is best illustrated by example.

3.4.1 VSP program style

The programming effort for this processor may be compared by considering the calculation of a Magnitude Squared Spectrum of 128 real valued time samples.

```
LD NMPT:128,RS:0,MDF:2,ZR:1,MBA:0;
```

FFT NMBT:128,RS:0,FPS:64,LPS:1;
MGSQ NMPT:128,RS:0,ADF:2;
ST NMPT:128,RS:0,RV:1,MDF:2,MBA:256;

The characteristics of the horizontal architecture are evident. NMPT defines the number of points in the vector, MBA is the memory base address of the array, MDF defines the data format-2 is REAL data. For the FFT instructions NMBT is the number of butterflies to be performed whilst FPS=first pass separation and LPS=last pass separation of points for the butterflies which specifies the Cooley Tukey algorithm in this case.

The AS: field selects the arithmetic selected. Only two options are possible, fixed point arithmetic with automatic scaling by 0.5 at each stage or block floating with shift on overflow. AS:0 selects the former, AS:1 the latter.

A useful feature is that the Assembler allows global or default values to be allocated to the fields.

3.4.2 Flow Control

There is a paucity of flow control instructions, JMPI,HLT and NOP being all that is available. Since all flow control must be achieved with these instructions, there is a need for interaction with the host. Assume a memory location SWITCH_ADD. Depending on certain tests, this address is loaded with the address of a previous VSP instruction in order to loop. Alternatively it could be loaded with the address of another routine, such as a scaling module.

The host is invoked by means of an interrupt, which, via the control bus halts further instruction fetches by the VSP. The interrupt is caused by a VSP instruction having its EI field set.

The instruction FIFO pipelining principle means that if the instruction following that which caused the interrupt is a JMPI then, because the MBA field is a variable, the loading must be delayed. Assume that this is the case with SWITCH_ADD. After the interrupt, the host, as part of the interrupt service routine, makes a decision and loads SWITCH_ADD appropriately. So as to allow this, bearing in mind the FIFO pipelining, a number (4) of NOP instructions must be interposed. Consider the program fragment :-

```
STI NMPT:1,STR:5,EI:1,MBA:SCALE;
```

This stores a VSP internal register (the scale register) in a location denoted SCALE, whilst interrupting the host EI:1. The host loads SWITCH_ADD according to circumstances and subsequent lines are :-

```
NOP NMPT:1;  
NOP NMPT:1;  
NOP NMPT:1;  
NOP NMPT:1;  
JMPI MBA:SWITCH_ADD;
```

3.4.3 The Scaling System

A 16 bit register is divided into 4 nibbles and each nibble carries a 4 bit number that is the scaling factor for an FFT or IFFT operation. A Scale Register pointer fills the register from the LS end. The pointer is reset after the MS nybble is used, or after an LDSM instruction with MD:0,UP:1.

The Scale Register can be read by the STI STR:5, or by reading its memory mapped address. The largest scale factor generated in a series of FFTs (since the last reset) is stored in the Maximum Scale Register (MSR). The MSR is reset by LDSM

MD:0,UP:1. The replaced value of the MSR is retained in the Old MSR register by LDSM MD:0,UP:1. Details are given by Zoran [10]

References

- [1] Holt,M.J.J., "A fast binary template matching algorithm for document image data compression", Pattern recognition, 4th International Conference Proceedings, Cambridge, pp230-9, 28th March 1988.
- [2] Flynn,M.J., "Very high speed computer systems", Proc IEEE, vol 54, pp 1901-1909, 1966.
In "Image processing system architecture", Ed. Kittler, J. Research Studies Press, 1986.
- [3] Fountain,T.J., "A review of SIMD architectures", in "Image processing system architecture", Ed. Kittler, J. ,Research Studies Press 1986.
- [4] Pass,S., "The GRID parallel computing system", ibid.
- [5] Edwards,M.D., "A review of MIMD architecture for image processing", ibid.
- [6] Kittler,J., Duff,M.J.B., "Image processing system architectures", ibid.
- [7] Brumfitt, P.J., "A review of other architectural concepts of image processing", ibid.
- [8] Karplus,W.J., "Architectural and software issues in the design and application of peripheral array processors", Computer, Vol 14, No.9, pp11-17 1985.
- [9] Dasgupta,S., "Computer architectures-a modern synthesis", Vol 1, Wiley, 1989.
- [10] Zoran Technical Notes TN92040-0187

TN92045-0187
TN92043-1087
TN94028-0187.
VSP-161 Assembler reference manual
1987.

4.0 Text Compression Benchmarking Rationale

4.1 Philosophy

One aspect of developing a data compression system based on feature extraction and pattern recognition is the need to quantify performance and compare with currently popular technology. The aim is to determine whether the greater complexity yields a sufficient advantage.

Many current methods were examined but experimental results quoted in papers refer to systems having such different operational parameters as to make comparison difficult. One example concerns FAX having a scanner resolution of 200 or 300 dots inch⁻¹. This resolution gives greater potential for run length coding than a source complying with CGA screen parameters.

The group 3 CCITT compression algorithm uses a modified Huffman encoding scheme which includes a run length feature. See CCITT^[6].

Fuchs^[1] indicates a compression of 5:1 to 15:1 can be achieved. He further states that a resolution of 100 pels inch⁻¹ and a minimum of 1728 horizontal pels resulting in an A4 sheet requiring 1.9Mbytes of storage. Assuming 10:1 compression this is reduced to 190kbytes.

The same page can be represented by just over two CGA screens with a total uncompressed size of 245 kbits as pixels, taking 36.6kbytes of storage. The disparity between the information stored in 1.9M bytes and the same data stored in 36.6k bytes is accounted for in that the former case has greater redundancy. No more information is inherent in the former than

the latter, that is, the entropy remains the same.

The FAX resolution, by virtue of greater redundancy, gives the greater compression potential as noted previously. This applies only to the textual case and it must be remembered that FAX is intended to communicate the entropy inherent to line drawings and hand writing as well.

It was calculated that when working with CGA like resolutions the compression ratio achieved would be scaled down by one order of magnitude. To confirm this hypothesis experiments were set up.

4.2 Experimentation

To carry out the tests it was first necessary to capture 640X200 screens and arrange to store the data for use as test data subsequently.

The requirements of screen capture clearly involve direct hardware control of the host computer keyboard, interrupt controller, and video RAM of the host IBM PC. Source programs in 8086 Assembly Language were written using the Wordstar/MASM/LINK/DEBUG environment.

A bottom up approach was used starting with the interrupt handling routine and progressing via DOS BIOS key handler and finally File Control Block operations. This method is appropriate when feasibility is unknown. Several difficulties were resolved by reference to Duncan^[2]. An example of idiosyncratic behaviour occurs if a screen is saved to a filename already used. The result was a corrupt file of double size. The

problem is associated with the detail of File Control Block handling and was resolved by flushing the block by zero filling below the file name section before a new capture.

4.2.1 The Benchmark Encoding Scheme

The basic principles of Huffman encoding are explained in the original paper^[3] and in Section 2.0. The procedure is appropriate to statistically independent sources and yields the minimum average word length.

In the Section 2.0 discussion, the problem of very long code words being generated by a large number of low probability entities was raised.

An approach designed to avoid these long code words is suggested by Hankamer^[4]. All low probability words are lumped together and signalled/stored literally, preceded by a special word indicating "uncoded data follows".

Implicit in this thinking is a decision about how many patterns should be considered "common" enough to be Huffman encoded. Obviously, such a decision would depend on the material and the logical approach is proposed in a slightly different context by Langdon and Rissanen^[5]. The key here is "adaption", which requires statistical analysis of the source.

If analysis is implemented the question arises as to whether the analysis should be carried out line by line or just once per frame. Clearly an extra overhead is incurred by the line by line method since data recovery will require that the new assignments for the codewords are recorded at each statistical event.

Experiments were necessary to quantify the answers to the

questions posed. Software written in Pascal and running on the host CPU utilised screens captured with the Assembler routine.

The experimental system was byte orientated which led to constraints in terms of accommodating the codewords if a pure Huffman code was to be used. It was realised that file sizes would not reflect the compression obtainable and that a separate "codeword size counting" exercise would be necessary.

An alternative approach proposed that if Hankamers principles were employed for all but the most probable 4 symbols, then a fixed codeword size of 2 or 3 bits could be used with little departure from the theoretical performance of Huffman. The reason for carrying out these experiments was to establish a rough benchmark and manual test calculations comparing variable length coding to the fixed scheme were made.

The style of the tests was to define strings of 4 possible entities or symbols. The appropriate Huffman codes were assigned by probability, and the 4 possible two bit fixed codes were also assigned to the entities. The strings were encoded both ways and a simple bit count taken. An example is shown below :-

String abdaaacbdc

Probabilities a=0.4, b,c,d=0.2

Codes allocated

Entity	Huffman	2-bit
a	1	00
b	00	01
c	010	10
d	011	11

Strings - Huffman 1 00 011 1 1 1 010 00 011 010

2-bit 00 01 11 00 00 00 10 01 11 10

In this case, for these probabilities the fixed 2-bit code is the same as the Huffman variable length equivalent.

The reason for this is that the Huffman coding procedure is capable of producing more than one coding option and the two bit code is one of the possibilities.

The fixed code length results in a simple relationship between file size in bytes and the total data stored within it.

To allow a Hankamer "uncoded data in this byte" signal one bit (b7) was reserved. Three 2-bit fields could carry data in a coded byte, whereas an uncoded byte could carry 1 of 128 possible entities. Software analysis of the data before encoding precluded any problems when less than three consecutive codeable entities existed.

As has been stated analysis to find the most probable entities could be based on a line by line scheme or on an analysis of a complete text screen. Performance differences between the two analysis options were unknown and it was resolved to quantify those differences by experiment.

At this stage it was suggested that two 4-bit fields might be used to carry the most probable 16 entities. Intuitively, it was felt that this would deviate substantially from Huffman equivalence but nevertheless it was thought that it would be useful to quantify results. Control functions necessary because no "Hankamer" bit was available in this option reduced the number of coded entities to the most probable 14.

Two software versions allowed the fourteen most probable words to be coded, or four. Frame analysis and line analysis were available by choice and finally a run length feature was added so that approximation to CCITT group 3 could be achieved.

Results

The maximum compression achieved with the available data occurred with four code words, line derived statistics, plus run length coding. Results suggested a compression ratio of 3:1 might be a typical average, but when congested text is the source it was noted that all the options gave similar results with ratios of 1.45 to 1.7 being achieved. This was predicted since congested text character cells were expected not to have dominant probabilities.

Figure 4.1 shows the results achieved using 10 different screens with 6 variations of encoding scheme. The encoding options are denoted A to F and the array elements 0 to 9 represent the different screens. Elements 0 and 1 are textual sources whilst 2 to 9 are line drawings captured from an electronic CAD package (ALTERA LOGICAPS). Increasing element number involving increasing sparseness in the drawing. Where screens contained entities of high probabilities such as line runs in drawings or blank space in sparse drawings the 14 code system without run length enhancements confirmed the intuitive predictions. This can be readily seen in comparing C(8)/C(9) and F(8)/F(9) in figure 4.1.

This work established benchmarks against which the pattern matching and substitution method which is the subject

Results - Huffman encoding + runlength

A: 14 code words. frame derived probability
 B: 4 code words. frame derived probability
 C: 14 code words. line derived probability
 D: 4 code words. line derived probability
 E: 14 code words. line derived + runlength
 F: 4 code words. line derived + runlength

x := 0 . . 9 0 to 9 are screens of increasing redundancy and sparseness
 0.1 are textual, remainder are line drawings.

A := $\begin{bmatrix} 1.0 \\ 0.799 \\ 0.655 \\ 0.744 \\ 0.693 \\ 0.622 \\ 0.622 \\ 0.655 \\ 0.662 \\ 0.6 \end{bmatrix}$

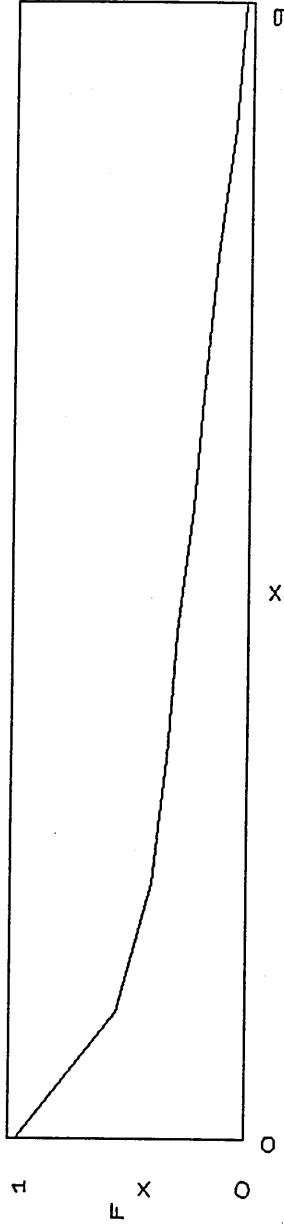
B := $\begin{bmatrix} 1.0 \\ 0.79 \\ 0.62 \\ 0.71 \\ 0.62 \\ 0.62 \\ 0.6 \\ 0.62 \\ 0.62 \\ 0.6 \end{bmatrix}$

C := $\begin{bmatrix} 1.0 \\ 0.76 \\ 0.56 \\ 0.6 \\ 0.53 \\ 0.47 \\ 0.45 \\ 0.48 \\ 0.41 \end{bmatrix}$

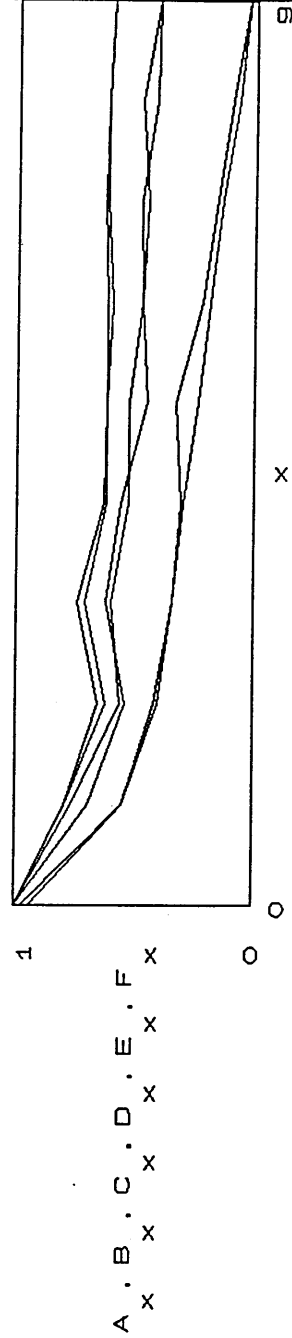
D := $\begin{bmatrix} 1.0 \\ 0.69 \\ 0.54 \\ 0.62 \\ 0.56 \\ 0.45 \\ 0.47 \\ 0.48 \\ 0.41 \end{bmatrix}$

E := $\begin{bmatrix} 0.94 \\ 0.55 \\ 0.42 \\ 0.34 \\ 0.31 \\ 0.33 \\ 0.22 \\ 0.16 \\ 0.10 \\ 0.03 \end{bmatrix}$

F := $\begin{bmatrix} 0.97 \\ 0.55 \\ 0.40 \\ 0.34 \\ 0.30 \\ 0.24 \\ 0.19 \\ 0.14 \\ 0.07 \\ 0.03 \end{bmatrix}$



best overall performance obtained



of Section 5.0 is compared. The results confirmed that the CGA resolution would cause compression algorithms to perform with a compression ratio approximately one order of magnitude less than for FAX resolutions.

References

- [1] Fuchs,P.M., "Compressing data in bit mapped displays",EDN,pp 173-183,October 1986.
- [2] Duncan,R., "Advanced MS-DOS",Microsoft Press, 1987.
- [3] Huffman,D.A., "A method for the construction of minimum redundancy codes",Proc I.R.E.,vol 40,pp 1098-1101,September 1952.
- [4] Hankamer,M., "A modified Huffman procedure with reduced memory requirements",IEE Trans on Comm, vol COM-27, no.6, pp930-932, June 1979.
- [5] Langdon,G.G.,Rissanen,J., "Compression of black and white images with arithmetic coding",IEEE Trans on Comms, vol COM-29, no. 6,pp858-867,June 1981.
- [6] CCITT recommendation T-6, ITU, Geneva.

5.0 Pattern Matching and Substitution

5.1 The Method

Documents comprise of textual characters, line drawings, and possibly photographs. This section deals with textual characters only. The intention is to use pattern matching and substitution after the fashion of Johnson, Segen and Cash^[1], and separately in a different context by Wilcox and Spitz^[2].

The method assumes document scanning, typewritten symbols and the ability to isolate individual entities during the segmentation process.

Several segmentation methods were considered. The classic means of isolating an entity on a uniform background is segmentation by histogram but this is only effective when there is a gradual change between the background and the object. This is because detection of the object is based on a trough in a bimodal histogram plotting numbers of pixels against grey scale. The background produces one peak, the object the other and intermediate values are expected to identify the boundary.

Textual characters from a scanner or on a computer screen have a sharp transition since only binary and not n-ary grey scale values exist. A good candidate for isolating text characters is thought to be chain coding as described in Section 8.2 and illustrated by figures 8.7 and 8.8. No implementation was attempted.

Once characters are isolated, their form is matched against those already recognised. If a match occurs the 2-D pixel image is not transmitted or stored. Instead a suitable short code is

substituted, typically a variable length type in order to approach the source entropy and data compression results.

If a "no match" condition occurs the pixel image and the code to be associated with it in future is transmitted or stored for use in the data recovery phase. A small amount of negative compression is incurred but in practice this is more than compensated for by savings when matches do occur.

The key process for successful application of these ideas is the pattern matching method.

Most workers in this field have used template matching, constructing an error pel map using weighted XOR to create a match figure of merit as did Holt and Xydeas^[3]. Many of the difficulties with template matching arise from registration of segmented characters or anomalies due to very thin entities.

The idea is illustrated by Figure 5.1. The bit maps for the two characters are combined by means of exclusively ORing them together. This creates an error pel map in which the error pels are weighted according to the number of adjacent error pels that exist.

The weighted error pel map is then totalled in order to generate a match figure of merit. A threshold for match is defined and decisions may then be made.

The difficulties mentioned above, concerning thin entities and registration can easily be identified as being exacerbated by the weighting procedure. This may be explained by considering a 1 pixel wide vertical element such as a thin "i" or "l". Slight mis-registration in an overlaid template could mean that no

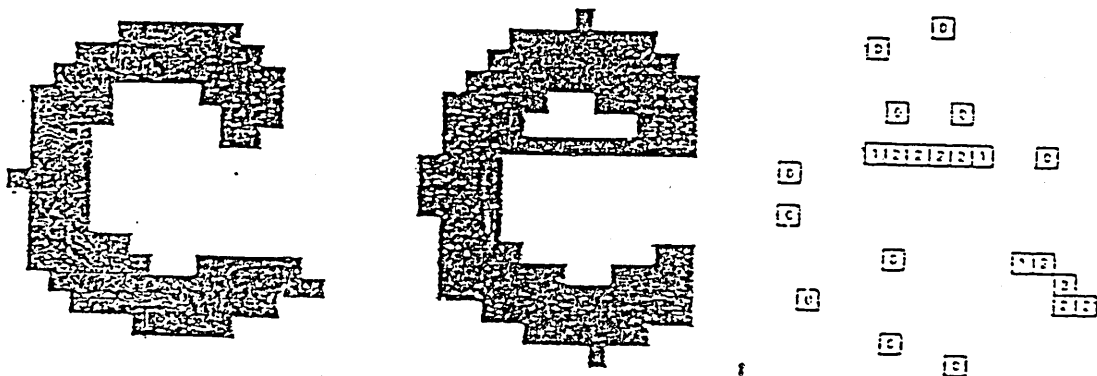


Figure 5.1

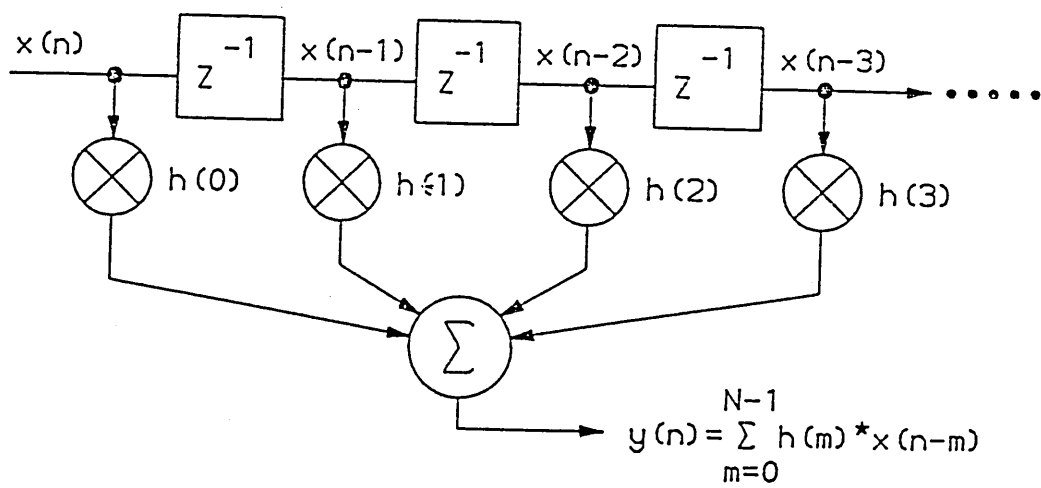


Figure 5.2

pixels line up. Thus every error pel would be weighted 2, as defined in figure 5.1, excepting the two end pixels which would be valued 1.

Holt and Xydeas have tried to compensate by further customising, or increasing the conditionality of their algorithms.

Other problems such as those reported by Boyle and Thomas^[4] concern the orientation of symbols.

Moving a template across a symbol and checking for best fit amounts to spatial domain convolution. The computational overhead has led to convolution being performed with very small kernels. Specialised microprocessor architectures such as the Vector Signal Processor give the potential of larger and much faster convolution.

The discrepancies of the weighted XOR map prompted an investigation of an alternative scheme. Research into early optical work revealed the notion of using matched filters for pattern recognition as suggested by Horowitz and Shelton^[5], and correspondence by Kain^[6] supports the idea by proposing statistical measures of matches.

5.1.1 Matched Filters

A matched filter is characterised by having an impulse response equal to the time reversed version of the signal waveform to which it is matched. In the context of this application, the filter is configured to an array generated from the segmented symbol.

When the symbol for which the filter is configured is applied, the output of the matched filter is the autocorrelation

function (ACF) for that symbol. Incorrect symbols will produce a cross correlation function (CCF). The main matching algorithm recognises which by statistical methods.

This application calls for the need to match large numbers of symbols and consequently the matched filter required must be programmable. The impulse response must be capable of being varied by writing control words to it.

5.1.2 Implementing the Matched Filter

[7]
Lynn discusses matched filters and demonstrates a particularly appropriate technique which applies to the sampled data system case. The assumption is that the signal is modelled as two-variable with binary states. This will generate an expression for the impulse response where the coefficients of the terms will either be 0 or 1. Clearly, where the coefficient is 0 then that term may be neglected.


In order to arrive at a configuration for the filter the following procedure was derived from Lynn [8], and Blandford [9]. Although for real segmented characters a much longer string is involved, consider a fragment of the single dimensional string representing a "slice" of the two dimensional character array.

101110000101.....

The time reversed version is :-

.....101000011101

Assuming a digital filter, the z plane transfer function will be :-

	SILVAF-11500
Matched Filter	
G.A. King -4/15/89	
sheet	2.00

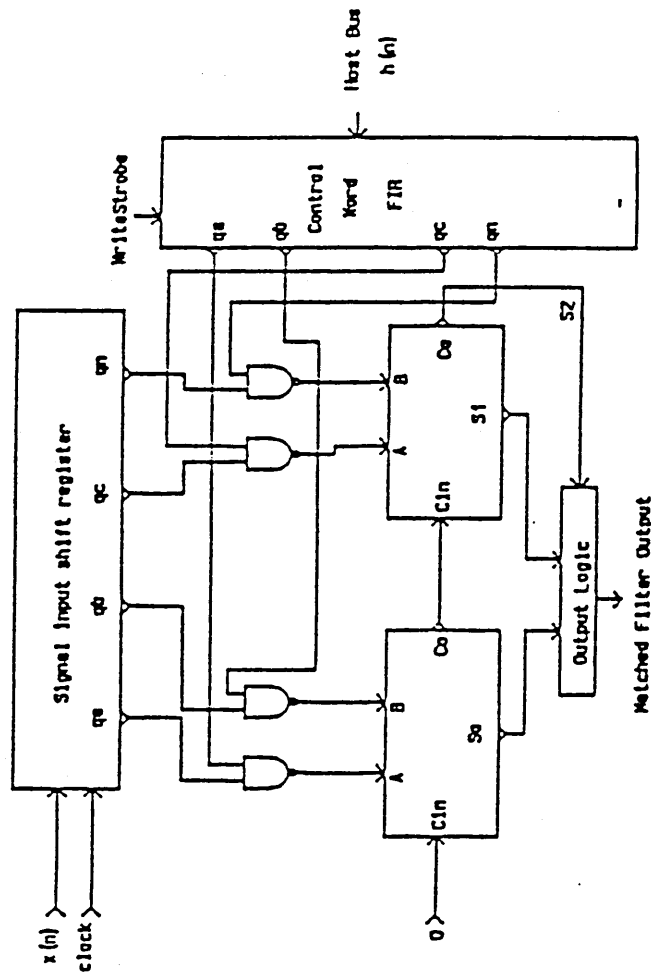


Figure 5.3

$$H(z)=1+z^{-2}+z^{-7}+z^{-8}+z^{-9}+z^{-11}$$

since $H(z)=Y(z)/X(z)$ then :-

$$Y(z)=X(z)+X(z).z^{-2}+X(z).z^{-7}+X(z).z^{-8}+X(z).z^{-9}..$$

This allows the recurrence formula to be written as :-

$$y(n)=x(n)+x(n-2)+x(n-7)+x(n-8)+x(n-9)+x(n-11)$$

which may be implemented as a Finite Impulse Response (FIR) filter of the non-recursive type. Figure 5.2 shows the system required. For the beginning of the example sequence fragment :-

$$h(0)=1, h(1)=0, h(2)=1, \dots, h(11)=1$$

This system was committed to an electrically programmable logic device (EPLD) of the Altera EP1200 type. The delay elements are implemented as a shift register of D type flip flops and the multipliers could simply take the form of two input AND functions because of the basic two valued approach. The coefficient input to the multipliers comes from a d-type PIPO (parallel in-parallel out) register. This register connects via address decoding and glue logic to the host processor synchronous bus. In this way the host can store a "configuration" word in order to configure the matched filter.

The summation requirement is satisfied by cascaded full adders and output logic. The implementation is shown in Figure 5.3. The problem with the two valued approach is that each character cell of, say, 16X8 pels results in long strings e.g. 128 pels. Each pel needs a signal shift register bit and consequently the basic implementation of Figure 5.3 would need to

be cascaded in large numbers.

5.1.3 n-valued Model

An alternative approach involves treating each horizontal slice of the character cell as a single sample value, and thus each character would comprise of 16 samples. This will reduce the string length but there will be a compensating increase in the complexity of the multiplier circuits and the summation unit.

Using arbitrary coefficient values to illustrate, the recurrence formula might appear :-

$$y(n)=252+79.x(n)+22.x(n-1)+121.x(n-2).....$$

The requirement is for at least a double precision (16 bit) result, together with array multiply hardware similar to that described by Hayes ^[10]. The complexity is significant.

5.1.4 Using the Vector Signal Processor

Taylor ^[11] describes a variant VSP designated ZR34325 which features an FIR instruction. This instruction has a field specifying the number of taps in the desired filter and expects an array in external memory to define the coefficients necessary. This solution will not be as fast as a pure hardware implementation but Taylor reveals results showing the ZR34325 capable of 32 tap filter action with an array of 128 real samples in 382 microseconds. If this is a viable solution both the Matched filter requirements and the Correlation needs can be provided by the VSP.

5.2 Generating Autocorrelation Functions

Flowchart 5.2 shows the complete process that is proposed.

Matching (1)

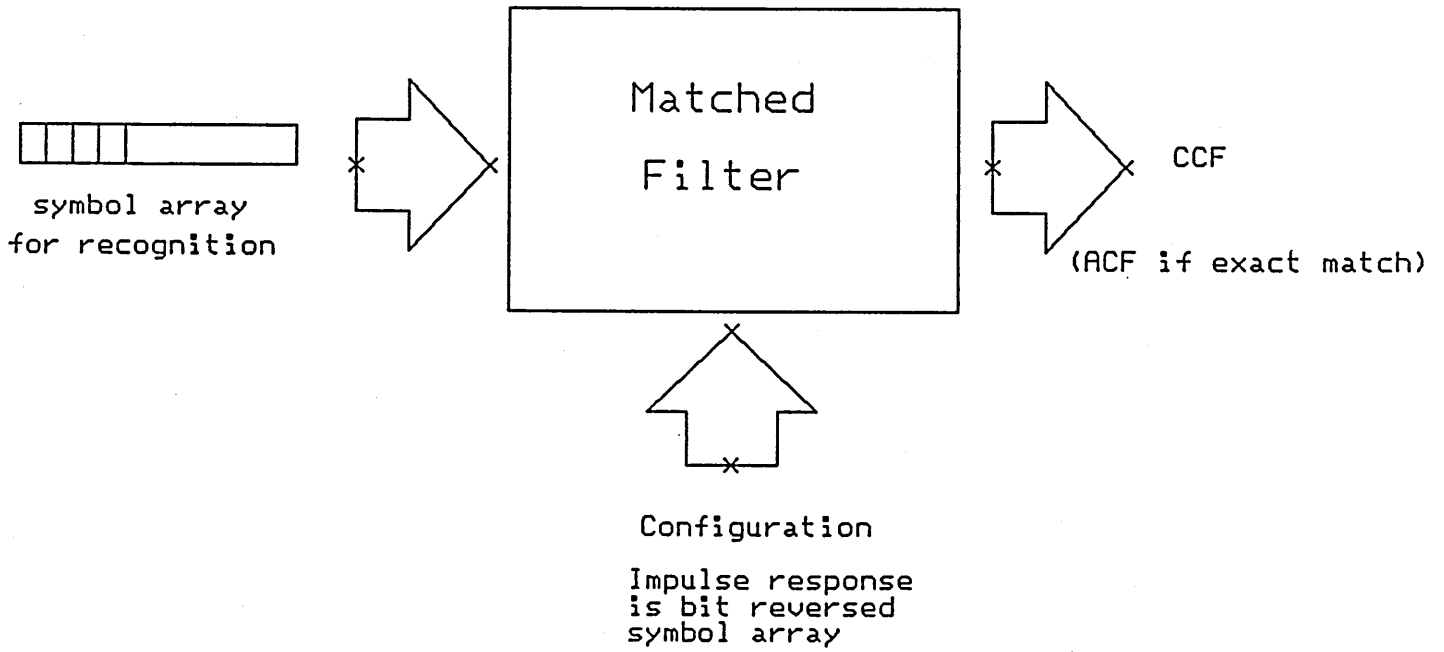


Figure 5.4

It is implied that for each character a set of full details will be created when it is encountered for the first time. This set has the following constituents :-

1. A bit reversed version of the character cell to be used as configuration input to the matched filter. see Figure 5.4
2. An Autocorrelation function of the character cell sample values.
3. A short code associated with this symbol.

Since a large number of new symbols will have to be dealt with in the early stages of textual processing it is advantageous if the Autocorrelation functions can be generated quickly. The need is for fast correlation. Fast correlation involves translations into the frequency domain.

Time domain

convolution is :-
$$y(n) = \sum_m x(m) \cdot h(n - m)$$

and correlation is :-
$$R(m) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_n x(n) \cdot y(n+m)$$

The applicability to the task in hand is evident if $h(n)$ is the impulse response of the matched filter, $m=0$ to n , $x(n)$ is the input sample sequence and $y(n)$ is the output. N is a power of 2 greater than the length of the shorter sequence. If DFTs (discrete Fourier Transforms) are used then the expressions become :-

for convolution,

$$Y(k) = H(k) \cdot X(k)$$

and for correlation,

$$S(k) = H(n-k) \cdot X(k)$$

Where :-

$$\text{DFT}[x(n)] \rightarrow [X(k)]$$

$$\text{DFT}[h(n)] \rightarrow [H(k)]$$

$$\text{DFT}[y(n)] \rightarrow [Y(k)]$$

$$\text{DFT}[R(m)] \rightarrow [S(k)]$$

The key point is that fast convolution can be used for correlation if $H(k)$ is stored in bit reversed order. The array store instructions for the VSP allow for this.

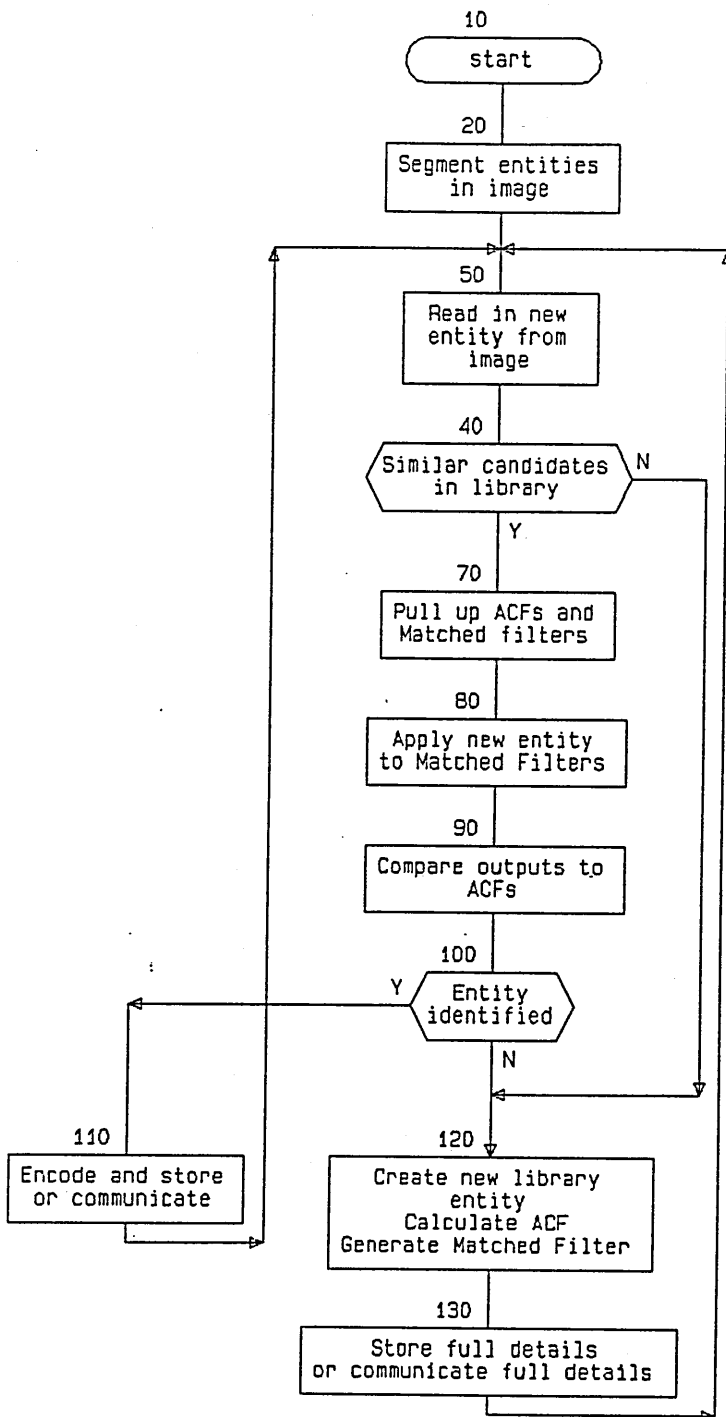
The process of frequency domain convolution is used to create the library ACFs. This job is invoked when a segmented character is not recognised as having been seen before at reference 120 in flowchart 5.3.

Details of the programming are illustrated by figure 5.5 and listing 5.1. Proof of the method is available in Gonzalez and Wintz^[12] and Zoran^[13]. The listing shows the characteristics of the horizontal architecture of the device, since each instruction has uncommon length with many fields, as discussed in Section 3.0.

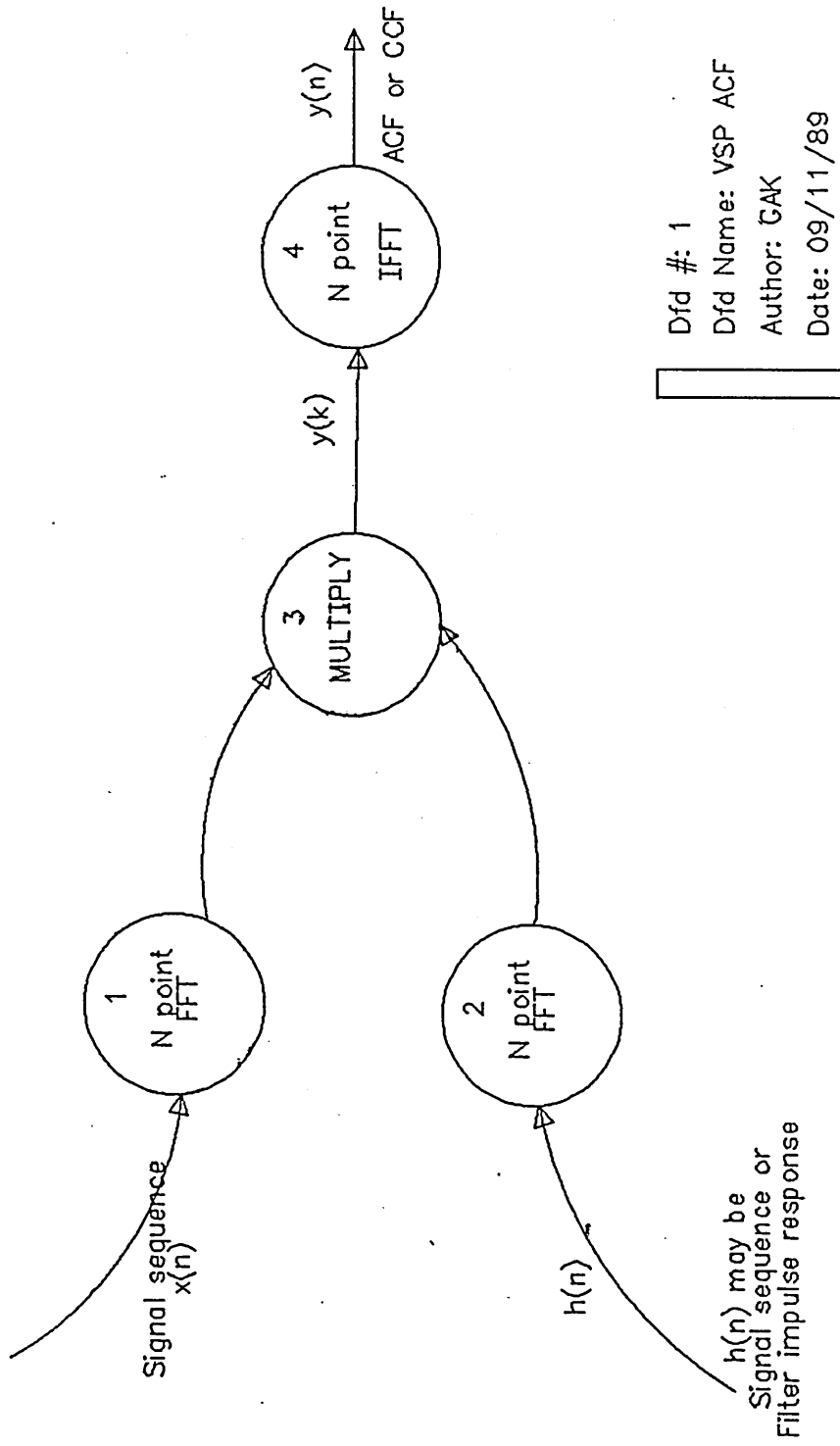
5.3 The Screening Stage

Before the main matching process proceeds, a rough comparison or screening stage takes four horizontal slices from the segmented character. In the original form of the screening stage a majority vote algorithm noted that if, say,

research



Implementation



Dfd #: 1

Dfd Name: VSP ACF

Author: GAK

Date: 09/11/89

Figure 5.5

```

LOC  OBJ  LINE  SOURCE
      1  /* Fast Correlation for VSP */
      2
      3  /* Cooley-Tukey Algorithms for X(N) and H(N) R:0 *
      4
      5  /* Sande-Tukey for IFFT R:1,FPS:1,LPS:64 */
      6
      7  /* Fixed divide by 2 scaling--each pass AS:1 */
      8
      9  /* G.A.King Data Compression version 1.00 11/10/
     10
     11  /* SET UP INSTRUCTION FIELD DEFAULT VALUES */
     12
     13  DEFAULT EI:0,INTRP:0,RV:0,ZR:0,ZP:0,MBS:128,MSS:
     14
     15  DEFAULT OR:0,CN:0,MDF:3,RS:0;
     16
     17  DEFAULT AS:0,FSIZ:128,FFT.RBA:0,ADF:3,SB:0,LN:1;
     18
     19
     20  /* OTHER DECLARATIONS */
     21
     22  equ SCRATCH=0xFFFF,XN=0x200;
     23
     24  equ N=128,HZ=0x300,F=0x400,SCALE=0xFFC;
     25
0000 0040 26  M: dw 64;
     27
0001 0000 28  PROGSTART: dw 0;
     29
     30  equ OUT=0x500;
     31
     32  org 0
     33
     34  /* SET UP SINGLE RAM SECTION MODE */
     35
0000 0010 36  LDSM NMPT:1,MD:1,UP:0,MBA:SCRATCH;
0001 A170
0002 0FFF
     37
     38  /* COMPUTATION OF H(Z) STORED AS H(N-K)
     39
     40  AND STORED AS H(Z) AS THE TWO ELEMENTS FOR AU
     41
0003 0000 42  LD NMPT:N,MDF:2,ZR:1,MBA:HZ;
0004 E050
0005 0300
     43
0006 9406 44  FFT NMPT:N,R:0,FPS:64,LPS:1,I:0;
0007 0030
0008 0020
     45
0009 0800 46  STB NMPT:N,MBAB:F;
     47
000A E068
000B FBFF
     48
000C 0800 48  ST NMPT:N,MBA:0x500;
000D E060
000E 0500
     49
     50  /* RESET SCALE REGISTER POINTER AND MAX SCALE RE
     51
000F 0010 52  LDSM NMPT:1,MD:0,UP:1,MBA:SCALE;
0010 A270
0011 0FFC
     53
     54  /* PERFORM VECTOR MULTIPLICATION */
     55
0012 0000 56  LD NMPT:N,MDF:2,MBA:F;
0013 E040
0014 0400
     57
0015 1006 58  MLTC NMPT:N,SH:1,MBA:0x500;
0016 E068
0017 0500
     59
     60  /* NOW PERFORM INVERSE FFT */
     61
0018 9406 62  FFT NMPT:N,R:1,FPS:1,LPS:64,I:1;
0019 8180
001A 0023
     63
     64  /* STORE THE SCALE REGISTER IN EXTERNAL MEMORY *
     65
001B 0811 66  STI NMPT:1,STR:5,EI:1,MBA:SCALE;
001C 4050
001D 0FFC
     67
     68  /* STORE THE ACF IN EXTERNAL MEMORY */
     69
     70  /* REMEMBERING TO EXCLUDE INCORRECT RESULTS OF C
     71
     72  /* number of valid points is N-M-1 */
     73
001E 0BF0 74  ST NMPT:63,MDF:1,MBA:OUT;
001F E020
0020 0500
     75
0021 C000 76  HLT;
0022 0000
     77
     78  end;
     79

```

S Y M B O L	TYPE	SEG	VALUE/SIZE	AT
F	CONSTANT	C	0x0400	
HZ	CONSTANT	C	0x0300	
M	LABEL	C	0x0000	
N	CONSTANT	C	0x0080	
OUT	CONSTANT	C	0x0500	
PROGSTART	LABEL	C	0x0001	
SCALE	CONSTANT	C	0x0FFC	
SCRATCH	CONSTANT	C	0x0FFF	
XN	CONSTANT	C	0x0200	

ASSEMBLY COMPLETE
 CODE SIZE: 25H, 37D
 79 LINES READ
 0 WARNINGS DETECTED
 0 ERRORS DETECTED

three out of four slices compare within the set tolerance to a library symbol, then the item is regarded as a likely candidate and is tested with the main matching technique.

A simulation of the screening process was created by programs written in a high level language. Starting with a library of characters defined by figure 5.8 each character was screened against the complete set including itself. The results are shown in table 5.1. The average number of characters selected as "possibles" needing to be checked by the main technique was 3.73, representing just under 15% of the library size. It is likely in practice for the library to contain a number of "versions" of the same character and this is expected to increase the number of "possibles".

There is a trade off between the screening and main techniques. If the screening is rigorous the main technique need only be invoked for a few candidates. The faster the processing for the main technique, the less demanding the screening need be. Results support the view that screening may be cursory when compared to more traditional pattern matching, certainly less exhaustive than the height/width/internal black run suggested by Holt and Xydeas^[3].

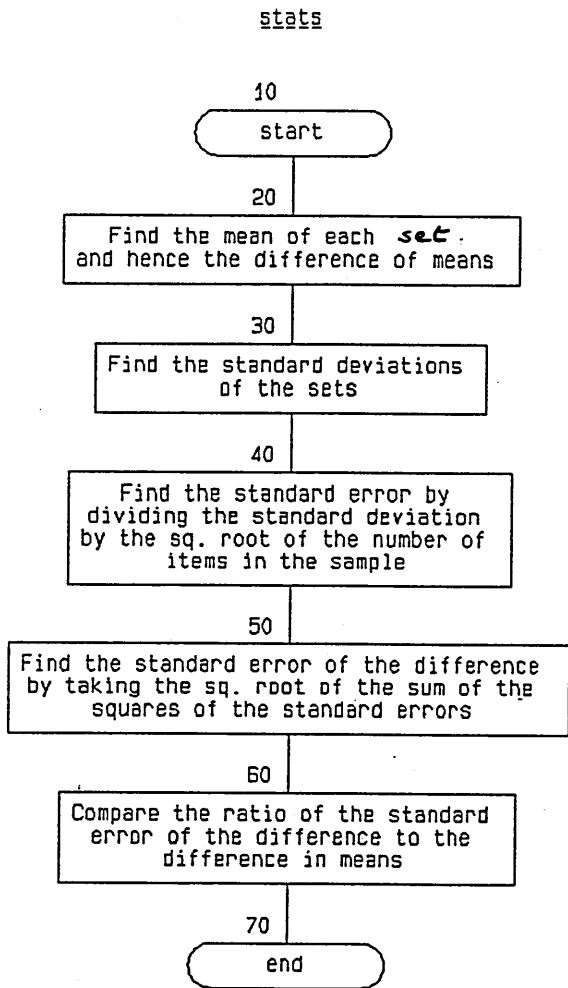
5.4 The Main Statistical Technique.

The essence of matching is to detect significant differences. Kain^[6] proposed several measures of similarity, the most useful is a figure derived by comparing the standard deviation of the filter output and the library ACF and then dividing by the mean for each sequence.

Table 5.1

<u>Character</u>	<u>Selected by screening</u>
a	a,g,q
b	b,f,k,m,p,w
c	c,e,s,v,x
d	d,s
e	e,c,o,v,x
f	f,b,k,m,n,p,u,w
g	g,a
h	h,r
i	i,l
j	j,z
k	k,b,f,m,n,p,u,w
l	l,i
m	m,b,f,k,b,w
n	n,f,k,p,u
o	o,e
p	p,b,f,k,m,n,u,w
q	q,a
r	r,h
s	s,c,d,v,x
t	t
u	u,f,k,n,p
v	v,c,e,s,x
w	w,b,f,k,m,p
x	x,c,e,s,v
y	y
z	z,j

Characters to be checked with main technique averages 3.73



It was felt that a better measure could be devised. The problem is the need to be able to determine whether the difference between sample sequences is significant or whether it arises from noise or other random effects. A method for doing this is described by Harrison^[14] and originates in statistical estimation techniques, such as those described in Chatfield⁽¹⁵⁾.

The mechanism is explained as follows :-

Two statistical measures are Standard Deviation, and the Mean.

A set of samples will have values for these two parameters. If there are several sets of samples then the Standard Error of the Means is the Standard Deviation for the set of means.

If a series of sample values is not significantly different then its mean will be within the Standard Error predicted.

If two sets of samples are possessed. One selected set should have its mean and standard deviation calculated, knowing the number of samples it is possible to predict the Standard Error which would apply to a group of similar sets. The same procedure may be repeated for the other set and then the Standard Error of the difference of the means must be calculated.

The actual difference between the two means is then compared with the Standard Error of the difference.

The results are interpreted in accordance with the normal distribution, so that :-

If two sets of data differ only by random effects, then

their means have a 68% chance of falling within a range defined by the standard error of the difference of those means. Other probabilities follow the normal distribution, so that there will be a 99.73% probability of being within three times the standard error of the difference.

The pre-requisite is the ability to calculate the standard error of the difference, starting with two sample sequences.

The final match figure of merit is obtained by dividing the difference of the two sample means by the standard error of the difference. Low values (<1) indicate a high level of confidence in the match.

Figure 5.6 demonstrates the method for two sampled waveforms. The resulting figure of merit (modulus of r), being 2.544 indicates a very low match probability.

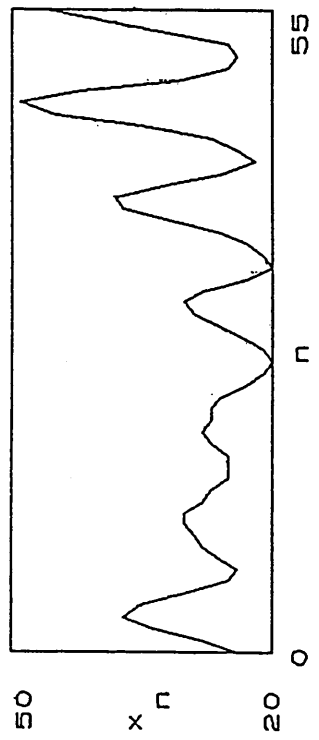
5.5 Experiments and Results

Kains method and that developed from Harrison were the subject of a comparative study. Complying with the original experiments by Kain, only the alphabetic character set was used. The character set is defined by figure 5.8. ACFs were generated for each symbol and then stored in a library.

To test the main matching technique ACFs were generated using an HLL(high level language) program. A typical ACF run is shown by figure 5.7.

Each symbol to be recognised was screened against all the rest in order to determine which, according to the screening algorithm, were candidates for the main matching technique. The results are shown in table 5.1. CCFs were generated for each candidate using a

Library ACF - candidate



mean(x) = 28.75
 stdev(x) = 6.52

b := mean(x)
 f := stdev(x)

Find the standard error in each case

$$i := \frac{f}{\sqrt{56}} \quad j := \frac{h}{\sqrt{56}}$$

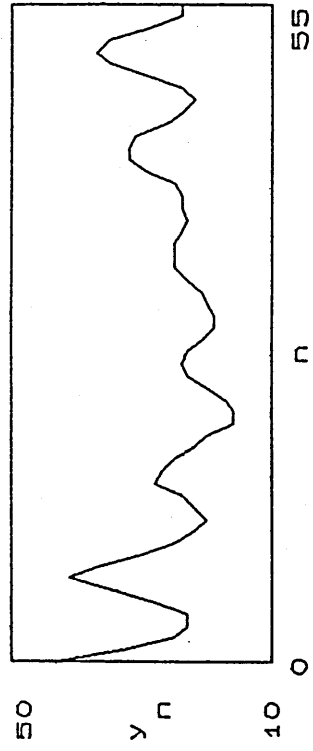
Find the standard error of the difference

$$k := \sqrt{i^2 + j^2}$$

Calculate the ratio of the standard error of the difference to the difference of the means

$$r := \left[\frac{d - b}{k} \right] \quad |r| = 2.544$$

Output from matched filter



mean(y) = 25.768
 stdev(y) = 5.868

d := mean(y)
 h := stdev(y)

Figure 5.6

matched filter set up for the original subject symbol.

If the original subject symbol was "a", then the screening selected "a,g,q". The correlation functions derived were "ag","aq", and "aa". This last ideally being the same as the ACF from the library. The correlation function "aa" should yield the highest match probability, i.e. the lowest numerical output from the statistical match routine. The statistical process would have, as inputs :-

- (1) The ACF of "a"
- (2) The correlation functions "ag","aq","aa" in turn.

The outputs for a representative series from the tests are shown in table 5.2. In the table an entry such as "bw" means that the ACF for character "b" was compared with the cross correlation of "b" and "w" (the output that would have resulted from "w" being applied to the matched filter configured for "b").

The results in the column relating to the method due to Harrison were derived as shown in figure 5.6. The table does not list the result of "bb","ww","ff" etc since in all cases this would yield 0, which means a "perfect" match. The results column entitled method due to Kain is derived by comparing the standard deviation of the matched filter output to that of the library ACF and then dividing by the mean. This was the best of the techniques suggested in Kains original research and the figures are provided to allow a comparison.

Table entries with numerical elements in their names such as "i2" or "e5" refer to symbol variations due to corruption, noise or other distortions. In the experimental work symbol patterns with such variations were added to the ACF library. Table entries

match figures of merit

Subject symbol and comparisons	Method derived from Harrison	Method due to Kain
-----------------------------------	---------------------------------	-----------------------

b		
	bw	18.10
	bf	16.33
	bk	13.80
	bm	10.61
	bp	7.94
		9.80
		1.90
		0.97
		4.74
		0.45
w		
	wf	40.26
	wk	34.03
	wm	6.15
	wp	28.10
		0.86
		1.86
		0.52
		0.10
f		
	fk	1.89
	fm	17.50
	fn	5.91
	fp	5.38
	fu	6.52
	fw	21.91
		0.52
		4.24
		0.41
		1.26
		0.32
		5.69
i		
	ij	6.76
		0.74
j		
	jz	7.83
		3.42
x		
	xc	0.53
	xs	1.71
	xv	2.11
	xe	4.38
		1.34
		2.24
		0.87
		2.57
a1		
	a2	3.34
		0.81
i1		
	i2	0.34
		1.20
b1		
	b2	0.75
		0.14
e1		
	e5	0.68
	e4	11.63
	e2	6.97
		0.97
		0.16
		2.21

in this class were a1,i1,b1,e1. Symbols containing greater distortion were matched against these in order to evaluate the relative tolerance of the two statistical match methods in cases of "same character but corrupted". Examples of the characters used are given in figure 5.8.

Although Kain reported 87% correct recognition in his original work there was a serious problem in differentiating between "i" and "j".

This is confirmed by table 5.2 entry "ij". Note that the new match procedure has no difficulty and clearly rejects this case, although in an integrated system as proposed the problem would not arise because the screening process only selects "i" and "l" for further matching. The figures for "ij" imply that the probability that they match is that associated with being 6.76 standard deviations from the mean in a Gaussian distribution. Kains match procedure generates 0.74, an indecisive value.

For the purposes of experiment it was decided to set the match figure of merit at 1.6. Outputs from the match procedure having values less than this would be declared the "same" symbol. This threshold is adjustable but 1.6 represents 3 or 4 uncompensated error pels. The threshold would normally be selected to define the acceptable distortion provided the match procedure is sufficiently discriminating and consistent. The method due to Kain produced poor results in this respect. Where "b" was the subject, screening suggested w,f,k,m,p,b and Kains method gives probable matches with "k" and "p" as well as "b". The new method has no such problem. Other examples are provided,

AUTOCORRELATION FUNCTION FOR LETTER A1

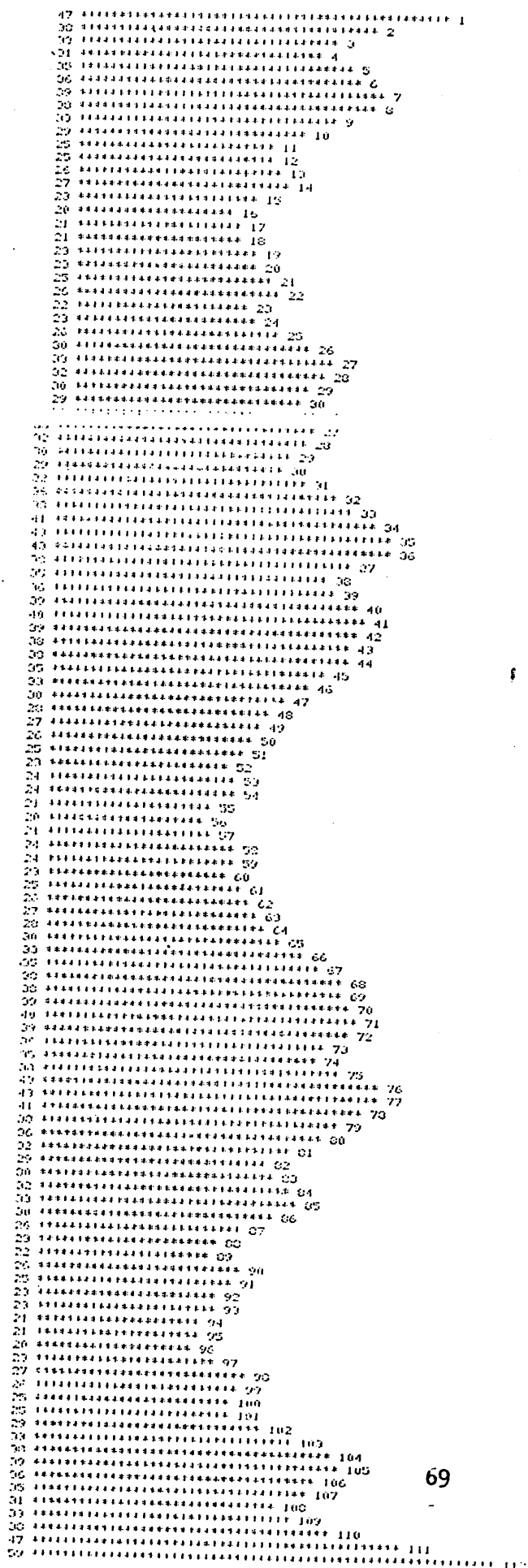


Figure 5.7

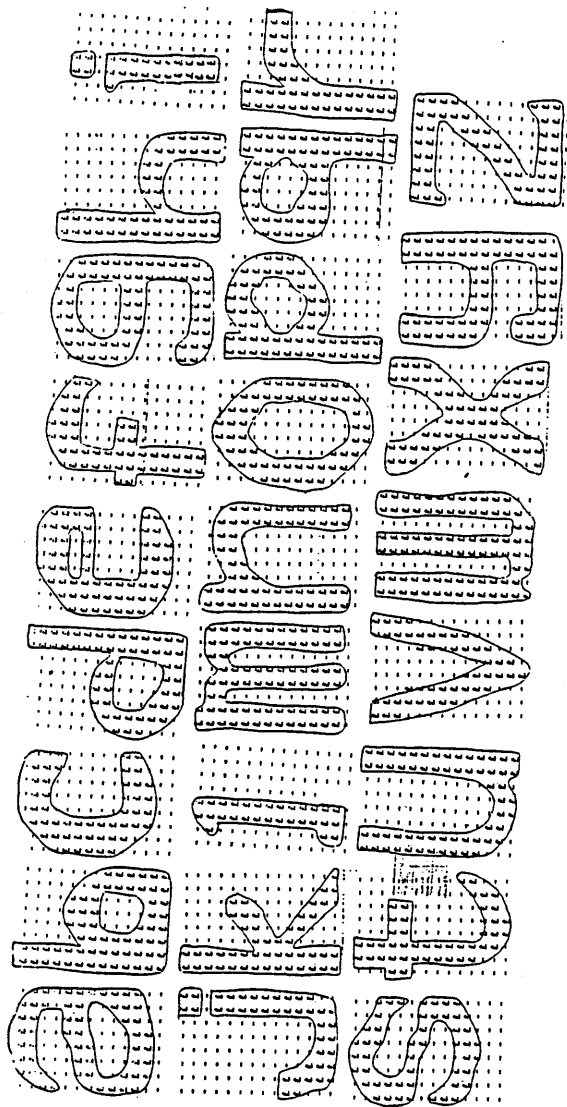
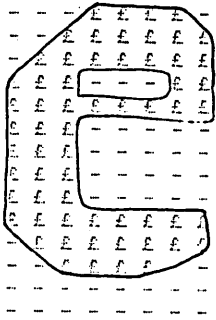
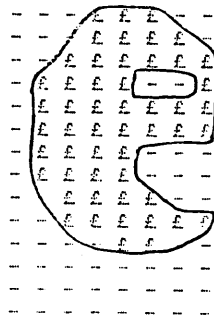


Figure 5.8(a)

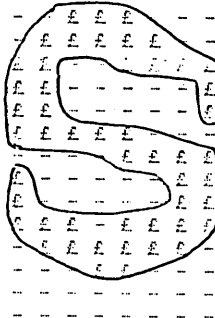
ENTITY ARRAY e1



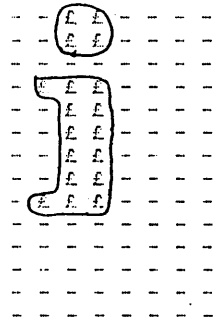
ENTITY ARRAY e2



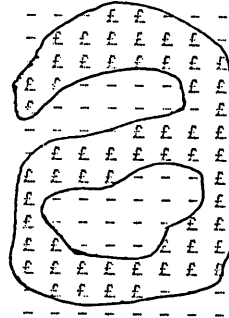
ENTITY ARRAY s1



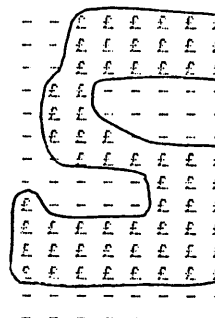
ENTITY ARRAY i1



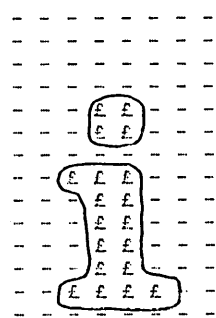
ENTITY ARRAY a1



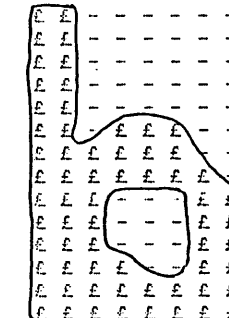
ENTITY ARRAY s2



ENTITY ARRAY i2



ENTITY ARRAY b1



ENTITY ARRAY b2

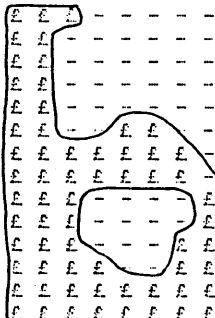


Figure 5.8(b)

but it is clear that the new method is much more consistent and discriminating. Table entries I1I2, B1B2, and E1E5 were for symbols differing only by small amounts of random noise. Entries A1A2, E1E2, and E1E4 all involved seriously distorted characters.

With the font used only "x" and "c" have yielded possible false match decisions and this has now been eliminated by the addition of vertical slicing at the screening stage. Instead of four horizontal slices the improved screening system uses three horizontal and one vertical slice, with the vertical slice centred in the character cell. Some examples of the character versions are shown in figure 5.8(b).

5.6 Secondary Encoding

The process described for pattern matching and substitution does not specify the "short code". Data compression will occur if the symbols are signalled/stored as 8 bit codes, say ASCII, but this does not take advantage of a further compression which is possible. Inherent to Section 4.2.1 was the notion of frequency analysis for particular pattern occurrences and it was considered that this principle might be usefully applied.

The difference between the textual case and the line drawings is the relative sparseness of the line drawings and the relative lack of repeat patterns in the scan line.

A number of schemes were considered for their suitability for improving the compression by getting away from ASCII representations of the matched symbols. The first option looked at was Binary Non-consecutive One code described by Wai Hung Ng^[16]. This code is a run length type developed from Elias^[17] but did not offer a good performance because it

relies on one symbol being significantly more common than the other.

Run length options were discarded and the Lynch-Davisson code^[18] examined. The Lynch-Davisson code is a time sequence code which is detailed in Section 2.0.

It is a total information time code and consists of two parts "q", which defines the number of non-redundant samples and "T" which carries the pattern of the sequence. A binary string is scanned and q,T determined as detailed in Section 2.7.

A number of examples were worked through and average compressions for English language phrases were found to be of the order 1.2 to 1.4:1. This was not considered impressive and it was decided to try to devise an alternative.

A new approach was to use the innovative Morse coding principle discussed in Section 2.8. A high level language program was written to encode English phrases in this way and a large number of examples produced an average compression of 1.5. Examples of program runs are given in figure 5.9.

This result is better than that yielded by Lynch-Davisson but not sufficiently so to be remarkable.

Pattern matching methods were applied by Downton and Kabir^[22] in the context of complete words of handwriting. The thought was triggered that greater compression could be achieved by substituting special bytes for the most common text words. Assuming that the segmentation process was able to detect spaces between words, the strings of character symbols

? THESAURUS

KEY CODE
001100001011010011011011011
CHARACTER CODES
10000000001001010001000
TOTAL BITS KEYCODE+CHARACTER CODE
50
NUMBER OF BITS AS ASCII
72
COMPRESSION RATIO
.6944445

? IN BOTH THESE SENSES THE QUOTATION IS APPOSITE
KEY CODE

01001010010001100110010000110000101100110001100101001100101110000110000110010001
1011001010001010011010100010011100010100100011011010001001
CHARACTER CODES

00101110100000010000111010000000001110000010000000011101000001110110100100010110
000010111000000111001011001100000000010

TOTAL BITS KEYCODE+CHARACTER CODE
257
NUMBER OF BITS AS ASCII
360
COMPRESSION RATIO
.6983696

? ROYAL ASSENT TO THE BILL WILL BE GIVEN
KEY CODE

01101110001010010001001101100101000110000101110000110000110010001010010001101
0100100100100001100011010100001010
CHARACTER CODES

01000011010101001110010000000101111010001110100000111010000001000100111001100010
00100111010000111011000001010

TOTAL BITS KEYCODE+CHARACTER CODE
224
NUMBER OF BITS AS ASCII
304
COMPRESSION RATIO
.7368421

Figure 5.9

THE SLEREXE COMPANY LIMITED

SAFORS LANE . BOOLE . DORSET . BH 25 4 EX

TELEPHONE BOOLE (943 13) 31617 . TELEX 123456

Our Ref. JSO/PJC/EAC

18th January, 1972.

Dr. P.M. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Barks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Registered in England: No. 2004
Registered Office: 20, Abchurch Lane, London, E.C. 4

Figure 5.10

Table 5.3 CCITT document 1 character frequency/occurrence

A	3	a	58
B	1	b	6
C	4	c	37
D	2	d	30
E	1	e	78
F	1	f	12
G	1	g	11
H	1	h	23
I	1	i	68
J	2	j	2
K	0	k	2
L	2	l	29
M	1	m	21
N	1	n	48
O	2	o	55
P	6	p	11
Q	0	q	0
R	5	r	48
S	3	s	51
T	3	t	66
U	0	u	27
V	0	v	9
W	0	w	3
X	0	x	0
Y	1	y	15
Z	0	z	0

coming from the pattern matching process could be analysed for the number of occurrences in the file or document. It was thought that a dictionary encoding scheme would be appropriate and that a 2 pass procedure would apply. On the first pass the words are stored in a hash table. The occurrences are kept track of by checking if it is already in the hash table and if not it is added. If it is already present the "number of occurrences" field is incremented.

On completion of the first pass the hash table is complete and the table is sorted in descending order of "number of occurrences". The top 128 occurrences are entered into the dictionary and the dictionary sorted into alphabetical order. The dictionary has two main fields, the word, and the single byte to be substituted.

On the second pass of the source data words extracted are searched for in the dictionary and substituted for by the single byte code. There will be many symbols forming part of other than the top 128 words and also some non-word symbols. All the entities :-

- 1) Code bytes for common words
- 2) Uncoded symbols/characters from relatively rare words
- 3) Uncoded symbols/characters that are non-word sourced

were then subjected to the Morse encoding scheme. It will be recalled that in the Morse scheme a 3 bit time/position indicates how to deal with the variable length code. Having not previously allocated any 7 bit patterns there were 128 available. Some were used as upper case characters. A deficiency of the original Morse code is that it does not allow for "case". Some of the

remaining codes were used as control codes to delimit the different categories of encodement above. This enabled the Morse for "a" for example to represent a popular word or simply the recognised single character "a" depending on how its meaning was switched by control codes.

The dictionary is specific to a particular document or file and must therefore be stored/transmitted for use in the recovery phase.

Programs to test the performance of these ideas were designed using the Axion Kindra CASE tool and auto-coded in "C". The programs were tested on "C" source programs and resulted in compression ratios between 1.8 and 2.0. Kindra flowcharts for the structured and modular code are contained in Appendix D.

The development of these ideas was not carried further since the implementation was not particularly applicable to Vector Processors, and was not central to the research.

The overall performance of the proposed pattern matching and substitution system then needed quantifying and in order to arrive at an estimate typical short business letters exemplified by CCITT document number 1 were undertaken.

Table 5.3 illustrates by defining occurrences of symbols. This work was also useful in estimating the number of autocorrelation functions likely to be generated. For this document 42 ACFS would need to be produced.

Figure 5.10 is the text of document CCITT 1. For this document calculations implied that pattern matching and substitution provided a compression ratio of 4.3:1 compared to raw pixel data.

This would be further improved by a secondary encoding achieving around 1.8:1. This leads to an overall compression of 6.5:1 which compares favourably with the benchmarks arrived at in Section 4.0. A typical average quoted there was 3:1, but less than 2:1 where congested text was used.

5.7 Implementing Fast Correlation

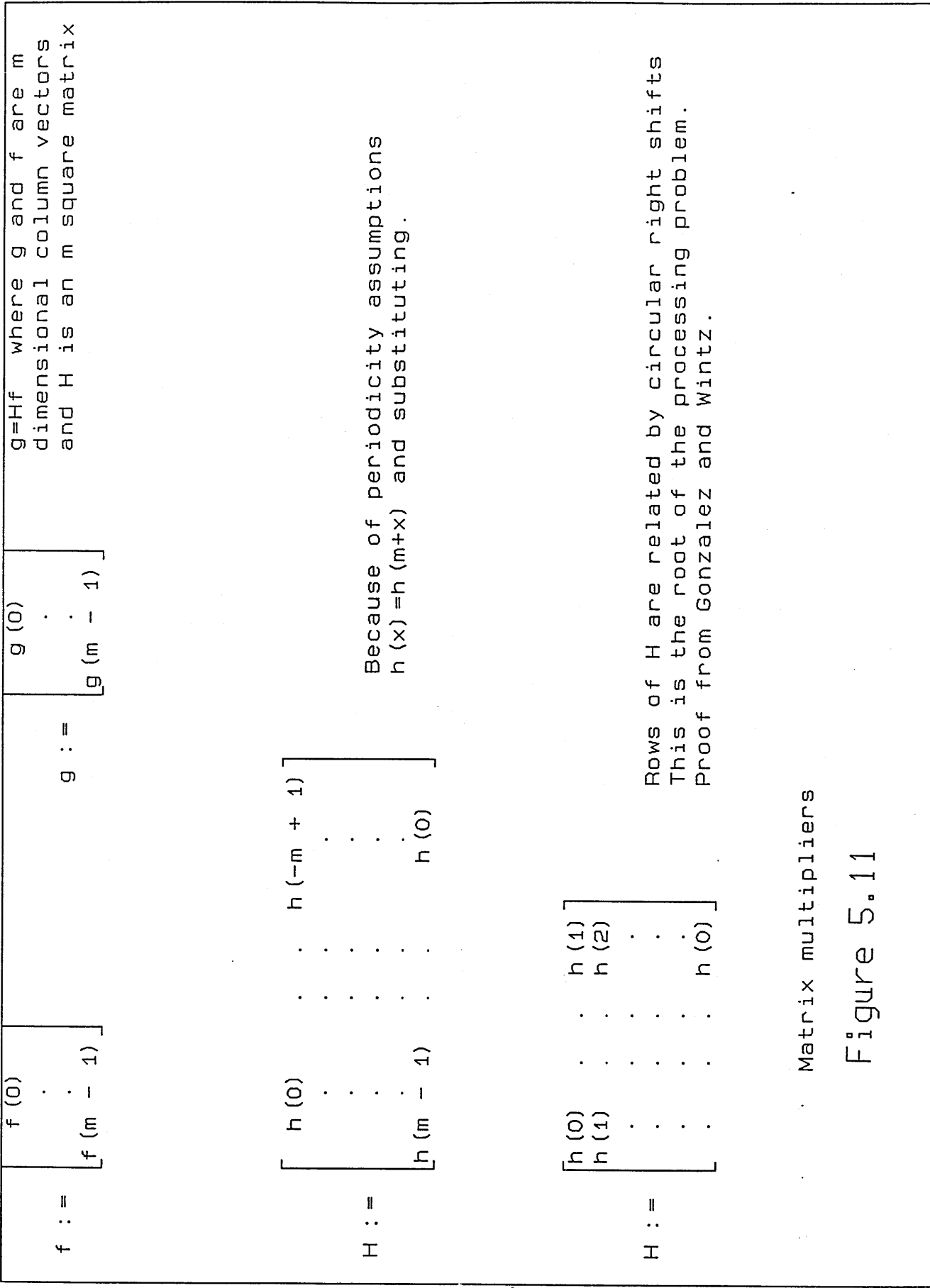
Convolution and Correlation between two sequences are very basic processes which find wide application in signal processing. This study has the requirement to implement correlation with very high computational rates and the straightforward evaluation of a correlation or convolution using n multiply/add operations per output point (n =length of shorter sequence) is computationally too intensive.

An alternative scheme is to operate in the frequency domain because convolution in the time domain is equivalent to multiplication in the frequency domain.

The process must begin with the application of the Fourier transform to both input sequences. Array multiplication is then followed by the inverse Fourier transform to produce the result. Fast convolution results and is illustrated by figure 5.5.

Since only sampled values are used, the discrete transform applies. The Discrete Fourier transform is periodic with period M . To operate with the Vector Signal Processor it is necessary to understand the principle of cyclic convolution.

If two functions $f(x)$ and $h(x)$ have ranges of $0,1,2,\dots,A-1$ and $0,1,2,\dots,B-1$ respectively, and if M is chosen to be greater than or equal to $A+B-1$ and the two variables are padded



to length with zeroes, then the padded convolution is :-

$$f(x) * h(x) = \sum_{m=0}^{M-1} f(m) \cdot h(x-m) \quad \text{for } x = 0, 1, \dots, M-1$$

according to Gonzalez and Wintz [12].

An alternative expression for this can be constructed by direct matrix multiplication as shown by Brickell [20]. Figure 5.11 demonstrates that rows of H are related by circular right shifts. This provides the root of a processing problem which can be appreciated by comparing linear and cyclic convolution in terms of samples as shown in figure 5.12.

5.7.1 Overlap and Discard

Zoran [13] discusses two methods for overcoming the incorrect results of cyclic convolution. The first, which is Overlap and Add, seeks to zero pad the sequences in positions where cyclic convolution would have produced erroneous values. Recombining the sequences involves extra add operations which incurred unnecessarily long execution times. For this reason, the method was rejected.

The alternative, Overlap and discard arranges to discard all the outputs affected, but produces the correct output sequence by segmenting the input sequence and ensuring overlap by M-1 points.

The method is illustrated by figure 5.13, and a VSP program for this algorithm is shown in listing 5.1 earlier in the Section.

5.7.2 Convolution or Correlation ?

The expressions for convolution and correlation given

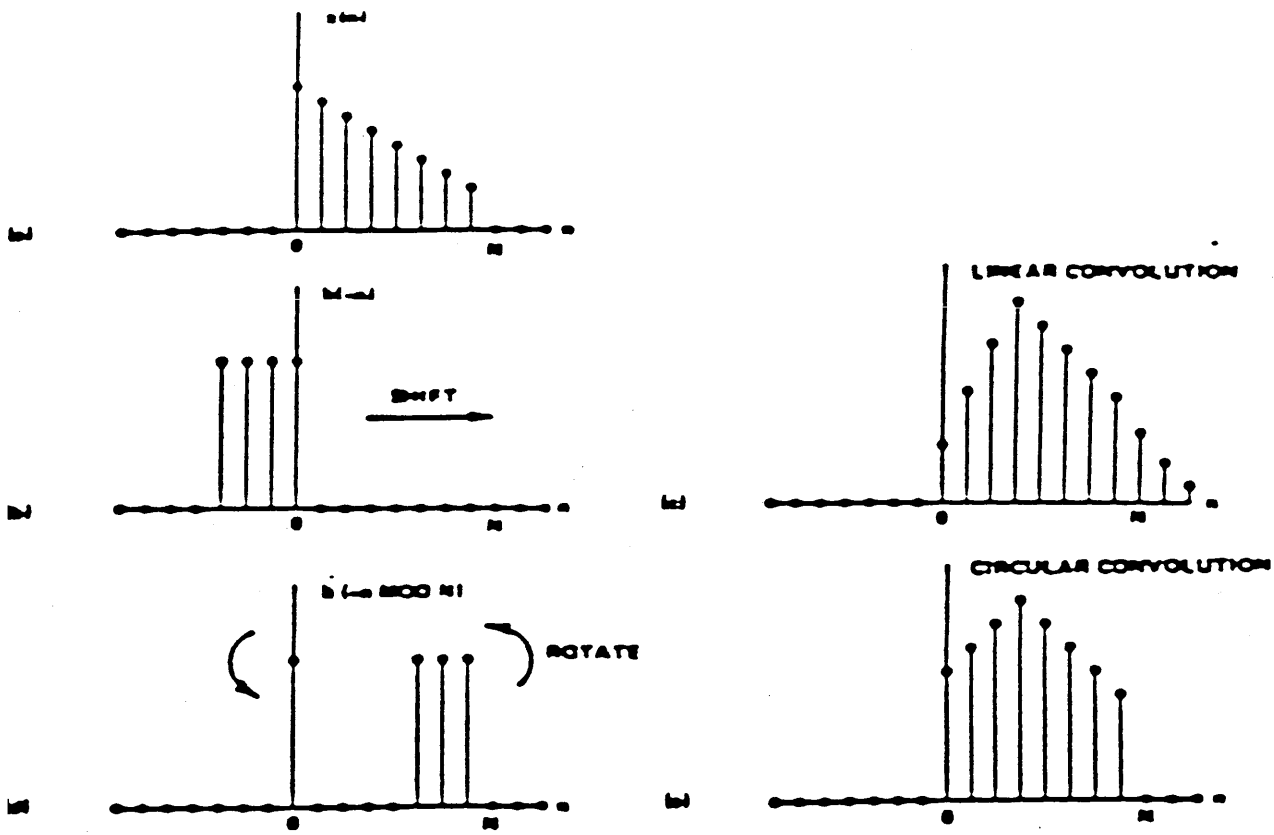


Figure 5.12

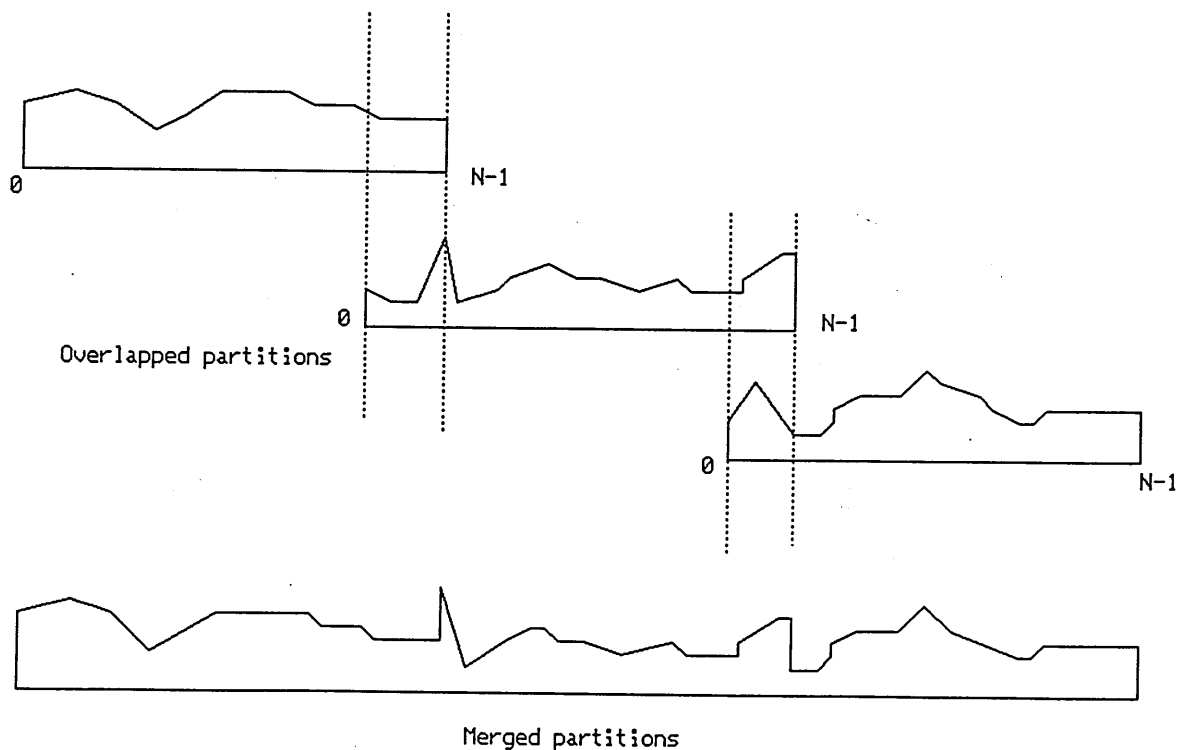


Figure 5.13

previously demonstrate that the only substantial difference between the two is a time reversal of one of the components.

Consider correlation through the use of the frequency domain.

If DFT[x(n)] is [X(k)]
and DFT[h(n)] is [H(k)]
then DFT[R(m)] is $S(k) = X(k) \cdot H(k)$ ^{*}
Proof is available in Zoran [13].

Considering convolution :- $Y(k) = X(k) \cdot H(k)$

The difference is that one of the components is used in a reversed way. e.g. H(k) might be substituted for by H(N-k).

The VSP provides a store backward instruction (STB), which makes the reversal a trivial operation.

5.7.3 The VSP program

The program operates on 128 points and a summation of the number of VSP machine cycles for the listed instructions yields a total execution time of 5079 clocks. With the processor running at 10 MHz the program runs in 508 microseconds.

References

- [1] Johnson, O., Segen, J., Cash, G.L., "Coding of two level pictures by pattern matching and substitution", Bell System Technical Journal, Vol 62 No.8 pp2513-2520, October 1983.
- [2] Wilcox, L.D., Spitz, L., "Automatic recognition and representation of documents", in Document manipulation and

- typography,Ed. J.Van Vliet (Proc Int. Conf on Electronic Publishing, Nice, France)),1988.
- 20-22 , Ed J.Van Vliet. 1988.
- [3] Holt,M.J.J.,Xydeas,C.S., "Recent developments in image data compression for digital facsimile", ICL Technical Journal pp123-146,May 1986.
- [4] Boyle,R.D.,Thomas,R.C., "Computer Vision ", Blackwell Scientific, 1988.
- [5] Horowitz,L.P.,Shelton,G.L., "Pattern recognition using autocorrelation", Shelton G.L., Proc IRE 49 pp175-185,1961.
- [6] Kain,R.Y., "Autocorrelation pattern recognition",Proc IRE 49 pp 1085-1086,1961.
- [7] Lynn,P.A.,"An introduction to the analysis and processing of signals", Methuen, 1983
- [8] Open University,"T326 Electronic Signal Processing", 1984.
- [9] Blandford,D.K.,"The Digital Filter Analyser", Addison Wesley, 1988.
- [10] Hayes , "Computer Architecture and Organisation" ,
- [11] Taylor,D.M.,"Single commands for complex DSP functions", Electronic Product Design,pp31-37, November 1987.
- [12] Gonzalez,R.C.,Wintz,P.,"Digital Image Processing", Addison Wesley 1982.
- [13] Zoran,"Fast Convolution", Technical Note TN 92045-0187.
- [14] Harrison.R., "Statistics and Reliability", Open University 1976.
- [15] Chatfield,C.,"Statistics for Technology", Chapman and Hall, 1978.

- [16] Ng Wai-hung, "Binary non-consecutive one code for time tag data compression" IEE, vol 118, No. 10, pp 1358-1360, October 1971.
- [17] Elias, P., "Predictive coding", IRE Trans on Inf Theory IT-1 No. 1, pp16-33, 1955 .
- [18] Lynch, T.J., "Sequence time coding for data compression", Proc IEEE, 54, No.10, pp1490-1491, 1961.
- [19] Fano, R.M., "The transmission of information"., Technical report No.65, MIT Research Lab of Electronics, 1949.
- [20] Brickell, F., "Matrices and Vector Spaces", Allen and Unwin, 1972.
- [21] Lynch, T.J., "Data Compression - Techniques and Applications", Van Nostrand Rheinhold, 1985.
- [22] Downton, A.C., Kabir, E., "Verification techniques for high performance OCR of hand printed postcodes", IEE Colloquium digest No.1989/109, pp7/1-7/7, 1989.

6.0 Transform Coding

6.1 Introduction

This section deals with the methods available for compressing grey scale image data. A characteristic of raster scanned n-valued image pixels is that they are usually strongly correlated. The purpose of transform coding is to involve a linear transformation in which the samples are mapped into a transform space. In doing this, the process results in a set of samples that are more independent. The transformation does not of itself provide data compression but the processes that do, such as quantisation, can be more effective with more independent samples.

6.2 Principles and Rationale

Consider a function of time such as a line of a raster scanned image. A typical case is shown in figure 6.1. Taking any sample as a starting point, the next sample will not be entirely independent. That is, a sample in the vicinity of the previous one will most probably be close in value rather than wildly different. This indicates that there is correlation between them.

Gonzalez and Wintz^[1] consider two consecutive pixels x_1 and x_2 . They produce a scatter plot of one against the other, see figure 6.2. Chatfield^[2] explains a measure of association between two variables. This measure is Covariance and is a statistic obtainable from the scatter plot mentioned above.

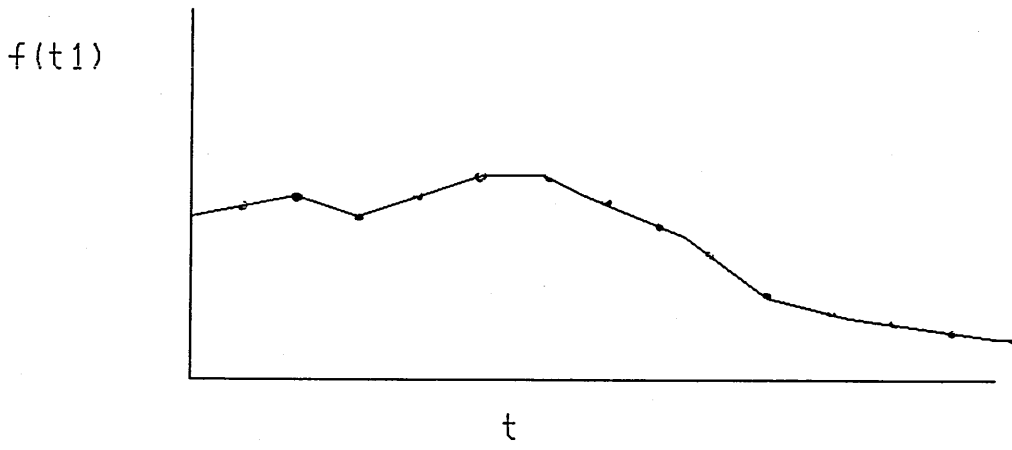


Figure 6.1

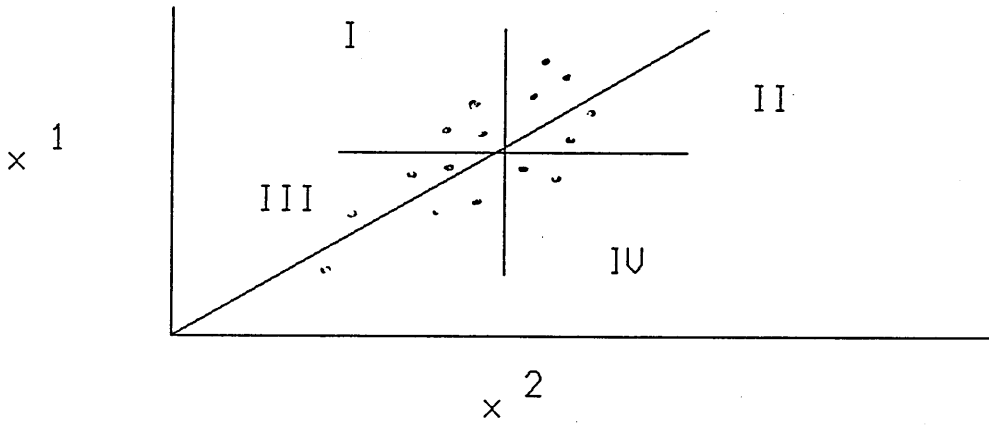


Figure 6.2

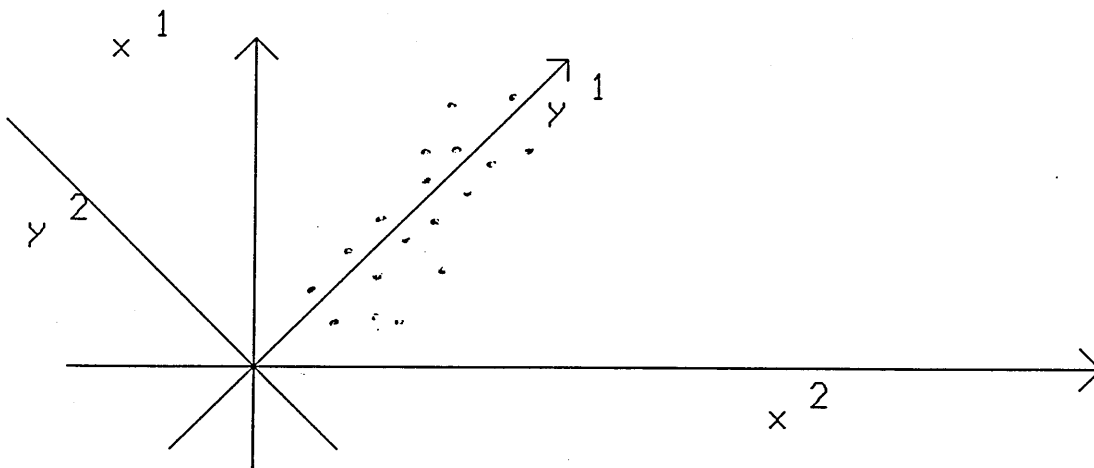


Figure 6.3

Covariance is defined as :-

$$S_{x_1, x_2} = \frac{1}{n-1} \sum_{i=1}^n (x_{1,i} - \bar{x}_1)(x_{2,i} - \bar{x}_2) \quad (1)$$

If most points are in the quadrants II and III, the covariance will be large and positive. Alternatively, most points in I and IV yield large negative covariance. If there is no correlation between the variables the points will be evenly scattered in the quadrants and the covariance approaches 0.

6.3 Transform Coding and the Vector Signal Processor

A number of transforms have been considered and evaluated for their ease of implementation with a Vector Signal Processor.

6.3.1 The Hotelling Transform

Hotelling^[3] describes a transform that is based on the statistical properties of an image. Karhunen^[4] and Loeve^[5] developed Hotellings discrete variable approach into an analogous process for continuous variables. Since the Vector Signal Processor (VSP) operates within a sampled data system environment Hotellings approach will be discussed.

Suppose the existence of a set of sample images denoted $f(x, y)$. Each image may be treated as a vector comprising of individual sample values x_{ij} , which denotes the j th sample value of vector x_i which represents the i th image.

A row, column array may be constructed by assigning the first n components of x_i to the first row, the next n components to the next row and so on.

The covariance between all the x vectors may be expressed as a matrix. The covariance matrix of the x vectors is :-

$$C_x = E \left[(x - m_x)(x - m_x)' \right] \quad (2)$$

$$m_x = E [x] \quad (3)$$

Where m is the mean vector, E is the expected value and the prime indicates transposition. Gonzalez and Wintz^[1] explain that equations (2) and (3) can be approximately derived from :-

$$m_x = \frac{1}{M} \sum_{i=1}^M x_i \quad (4)$$

$$C_x = \frac{1}{M} \sum_{i=1}^M (x_i - m_x)(x_i - m_x)' \quad (5)$$

which relates to equation (1).

The procedure then requires the derivation of the eigenvalues and eigenvectors of C . An array whose rows are the eigenvectors of C is then constructed and denoted A .

$$A = \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1N} \\ e_{21} & \dots & \dots & \dots \\ \vdots & \dots & \dots & \dots \\ e_{N1} & \dots & \dots & e_{NN} \end{bmatrix} \quad (6)$$

Where e_{ij} is the j th component of the i th eigenvector. The Hotelling transform is complete when the image vector $(x - m)$ is multiplied by array A .

This yields a transformed image vector :-

$$y = A(x - m_x) \quad (7)$$

In order to examine the effect of this transformation it is necessary to consider the covariance matrix of the y vector. By applying similar equations to (3) and (5) :-

$$C_y = E[(y - m_y)(y - m_y)'] \quad (8)$$

and

$$m_y = E[y] \quad (9)$$

but substituting (7) into (9) gives :-

$$m_y = E[A(x - m_x)] \quad (10)$$

$$= E[Ax - Am_x] \quad (11)$$

The estimated value of $(Ax - Am_x)$ is the same as the estimated value of x , multiplied by array A , and minus the value of Am_x hence :-

$$m_y = AE[x] - Am_x \quad (12)$$

but $m_x = E[x]$ and substituting :-

$$m_y = A.m_x - A.m_x = 0, \text{ a zero vector.}$$

It is now possible to write C_y in terms of C_x by substitutions.

Remembering $m = 0$ and $y = A(m - m_x)$ equation (8) becomes :-

$$C_y = E[y \cdot y'] \quad (13)$$

$$\begin{aligned} &= E[(Ax - Am_x) \cdot (Ax - Am_x)'] \\ &= E[A(x - m_x)(x - m_x)' \cdot A'] \\ &= A \cdot A' \cdot E[(x - m_x)(x - m_x)'] \end{aligned}$$

but $C_x = E[(x - m_x)(x - m_x)']$

therefore $C_y = A \cdot C_x \cdot A'$ (14)

Gonzalez and Wintz [1] show that C_y is a diagonal matrix with elements equal to the eigenvalues of C_x .

$$C_y = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}$$

Since the terms off the main diagonal are 0, the elements of y are uncorrelated. To arrive at this point it is necessary to have derived the eigenvalues of C_x .

The eigenvalues of C_x are the solutions to the matrix equation :-

$$C_x \cdot \phi = \lambda \cdot \phi \quad (15)$$

where λ represents the eigenvalues.

Firstly, the characteristic equation :-

$$\det [C_x - \lambda I] = 0 \quad (16)$$

is solved for the eigenvalues λ . I is the unit matrix. The

eigenvalues are then arranged in descending order of magnitude and substituted into $(C_x - \lambda I)\phi = 0$ to solve for the eigenvectors.

One of the basic concepts underlying the Hotelling transform is that rows of [A] point in the direction of maximum variance of the data as shown in figure 6.3. . The transform is reversible and does decorrelate the samples. It packs the maximum amount of variance into the first n coefficients, and the quantising process can discard the higher order coefficients to obtain a data compression.

In summary, the operations required to implement the transform are :-

- (i) Means of vectors
- (ii) Deriving covariance matrices as illustrated by equation (5)
- (iii) Solution of equations such as (16)

In these contexts the VSP does not offer any advantages over standard microprocessors and its lack of conditional jump and branch instructions would demand interactive reliance on the host. Useful functions which would improve the applicability of the VSP are DIVIDE, SUBTRACT, and CONDITIONAL JUMP.

Although this transform is optimal the present development state of the VSP does not allow it to be a useful tool.

6.3.2 The Fast Fourier Transform

Boyle and Thomas^[6] correctly observe that Fourier theory plays an important part in image processing. This is also true of signal processing and Lynn^[7] starts by analysing repetitive

waveforms in terms of the Fourier series and then extends the principle to cover the non-repetitive waveform case.

In general the Fourier transform allows a signal that is a function of time to be represented in the frequency domain.

For the purposes of this research only sampled data signals will be considered. Such signals are modelled as a series of weighted Dirac functions :-

$$f(t) = x_0 \cdot \delta(t) + x_1 \cdot \delta(t - T) + x_2 \cdot \delta(t - 2T) \dots$$

The Fourier Transform of this function is :-

$$G(j\omega) = x_0 \cdot e^{-j\omega t} + x_1 \cdot e^{-j\omega(t-T)} + \dots$$

This transform of a sampled data signal is generally known as the Discrete Fourier Transform (DFT).

As opposed to the Hotelling transform, which has to be calculated for each data set, the Fourier Transform is fixed in form whilst its performance is nearly as good. For a detailed quantitative comparison see Pearl [8]

Specifically, the DFT is given by the expression :-

$$X(k) = \sum_{n=0}^{N-1} X(n) \cdot W_N^{kN}, \quad k = 0, 1, \dots, n-1$$

where X(n) is a signal sample value

X(k) are frequency domain values
 $W = e^{-j2(\pi)/N}$

Direct computation requires N^2 complex additions, which can produce a large overhead.

The basic idea behind the fast fourier transform is that the

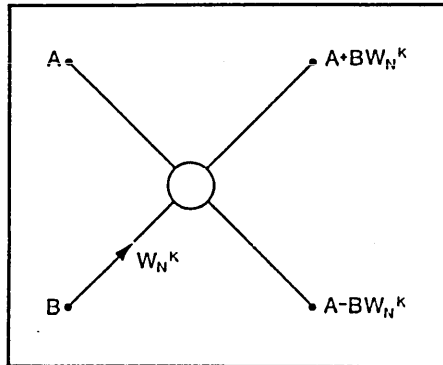


Figure 6.4

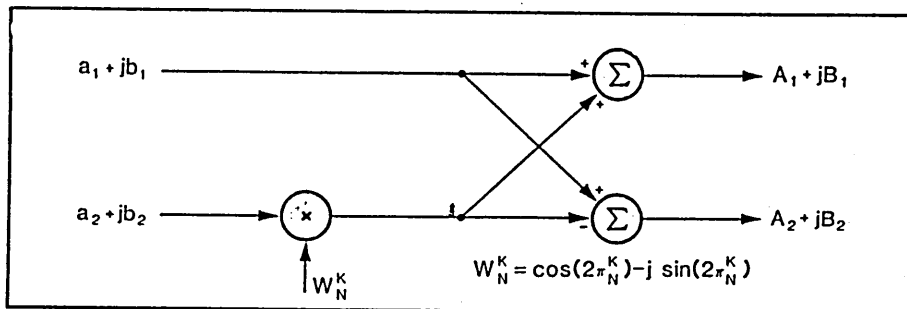


Figure 6.5

summation over N samples can equally well be achieved by a combination of summations over N/2 samples. In the case where N is a power of two, the sample grouping principle can be applied repeatedly, culminating in a final summation of only two values.

Hutton^[10] discusses an algorithm to accomplish the basic operation needed. This algorithm is called the Radix-2 DIT (Decimation in time). The technique breaks down the DFT into two smaller transforms, thence into two smaller transforms and so on.

Eventually, the DFT is broken down into a number of two point transforms. A two point transform is known as a BUTTERFLY. The origin of the name derived from the configuration of the signal flow diagram which may be used to describe it, as shown in figure 6.4.

The operation accepts 2 complex words, A and B and delivers two transformed words. Taylor^[11] describes the operation in a more detailed way as shown in figure 6.5.

Figure 6.5 indicates that any system attempting FFT action will involve a complex multiplication and two complex additions. The complex multiply may be achieved by four real multiplies and two real additions. Complex addition can be provided by two real additions.

This leads to a total number of computations required to implement the butterfly of :-

4 multiplications and 6 additions

Hutton^[10] derives the total number of butterflies required to execute an N point transform as :-

$$\frac{N}{2}(\log_2 N)$$

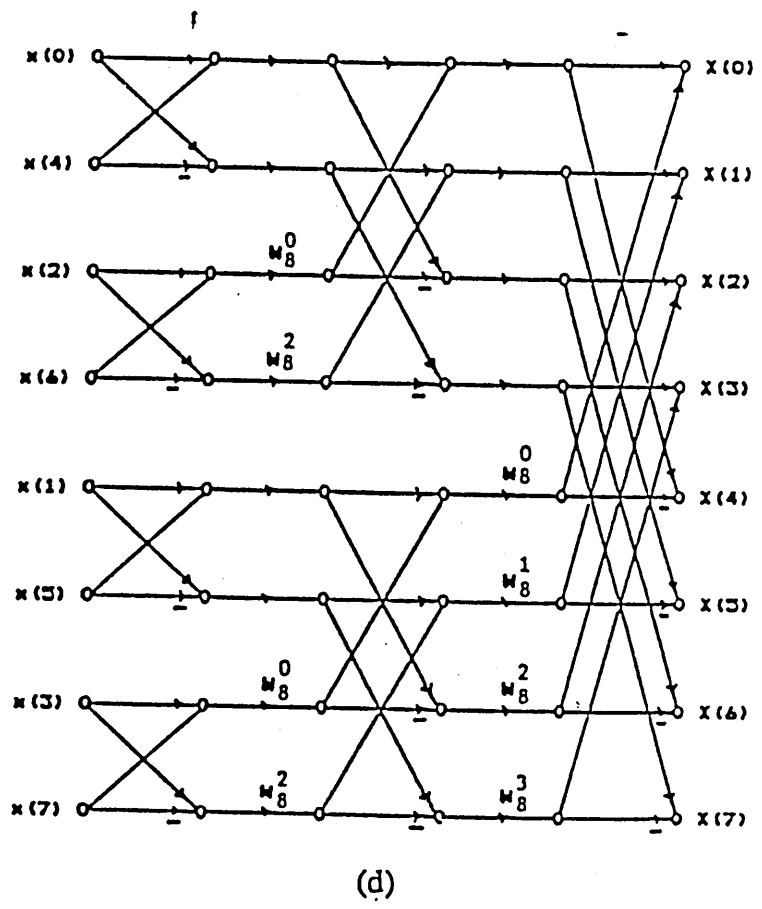
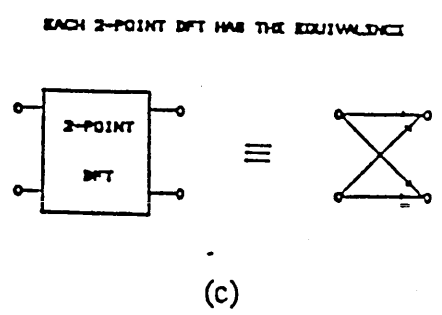
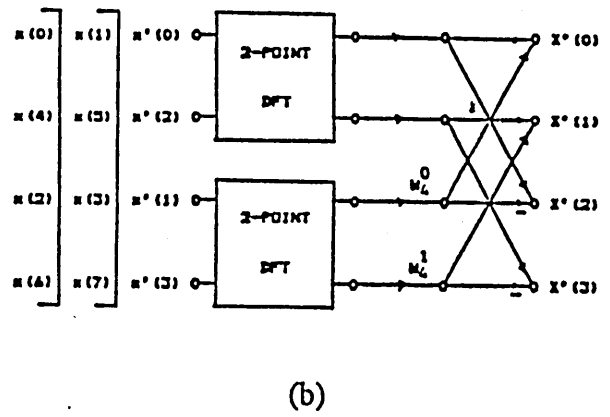
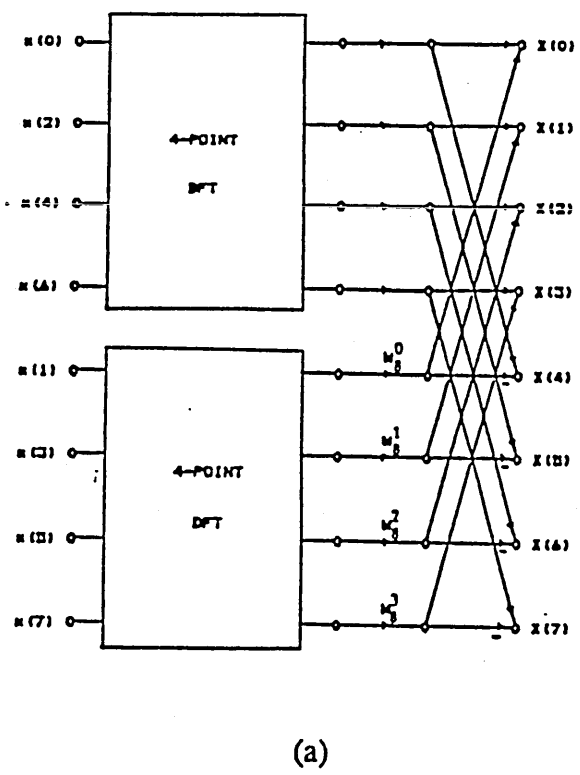


Figure 6.6

Thus a 1024 point operation is achieved by 5120 butterflies, and the computational advantage increases with N.

The detailed algorithm must be analysed in order to detect any special computational requirements.

The Sande-Tukey^[12] factorisation explains the necessary steps.

It should be noted that an 8 point FFT is achieved by two 4 point DFT followed by 4 butterflies. The 4 point DFTs can be decomposed so that a single 4 point DFT may be realised as two 2 point DFTs or butterflies, as shown in figure 6.6 (b),(c).

The full decomposition into butterflies is shown in figure 6.6 (d) .

If the butterflies are executed in 3 passes it should be noted that the bit separation of the input is 1 for the first pass and 4 for the last pass. These are important parameters and will be denoted to comply with the Vector Signal Processor assembly language field names "FPS", and "LPS" (the first and last pass separations).

Sande-Tukey factorisation proceeds by treating the time samples as two groups, odd and even indexed. It is noticeable that the results are subject to a bit reversed re-ordering of the original time samples.

x(0)
x(4)
x(2)
x(6)
x(1)
x(5)
x(3)
x(7)

This means that the data must be loaded in the re-ordered way before the FFT is performed. The output is then correctly ordered.

The VSP uses an RV (reverse data field) with its LD (load) array instruction. The instruction sequence is :-

```
LD      RV=1      load reverse ordered data
```

```
FFT
```

```
ST      RV=0      store normally ordered
```

An alternative algorithm, the Cooley-Tukey ^[9] form has the same "addition and subtraction before or after multiplication by a complex weighting" structure. The index groupings of the Cooley-Tukey technique mean that input in normal order results in a bit reversal output. Thus for the VSP the instruction sequence in this case is :-

```
LD      RV=0
```

```
FFT
```

```
ST      RV=1
```

A VSP program for an n point FFT is in Appendix B.

6.3.3 Architectural Factors

The internal RAM capacity of the Zoran VSP co-processor is 128x38 bit complex words. The most significant 19 bits are allocated to the real component. Only 17 bits are used for storage and two bits are guard bits coping with temporary overflows. 16 bits of the word are accessible. From the users point of view the memory is effectively 128x32.

The Zoran VSP achieves concurrent action with separate execution and input/output units. This is a common feature in

peripheral vector processors (Karplus^[13]), but for it to be an advantage it is necessary to be able to partition the RAM. In this case the RAM can be partitioned into 2x64 locations. This contributes a 13% speed advantage for small FFTs.

A number of processors optimised for signal processing support both fixed point as well as block floating point operations. When using fixed point operation the ALU results are scaled by two at the end of each pass. Block floating point results are only scaled by two if an overflow has occurred.

6.3.4 Two Dimensional Fast Fourier Transform

Gonzalez and Wintz^[1] claim two main advantages for the two dimensional transform. The first advantage is concerned with interpretation, in that the Fourier spectra are often usefully displayed as an intensity function.

Many image spectra have a dramatic amplitude reduction as a function of frequency, and the usual processing procedure is to display the function modified by a Logarithmic operation. e.g.

$$D(u,v) = \log[1 + |F(u,v)|]$$

A further adjustment is often made to the centre of the display. The properties of conjugate symmetry and periodicity then give a characteristic display.

The second advantage is best explained by considering the coding performance for a transform encoder.

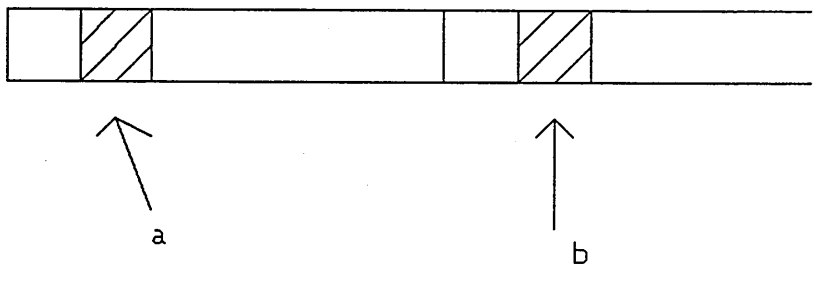
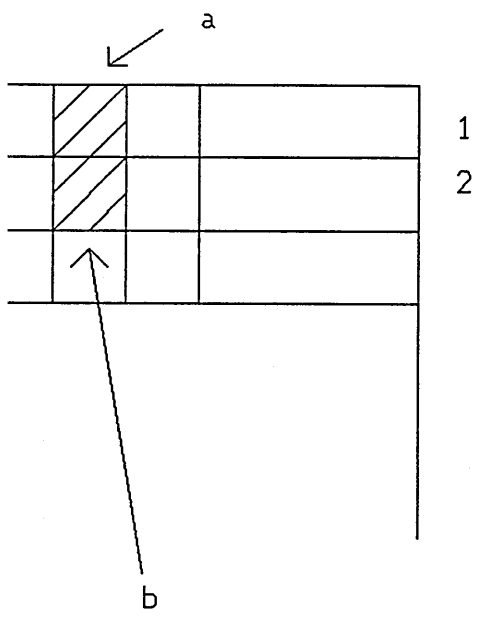


Figure 6.7

Adjacent pixels a and b are separated
 this subpicture shape strategy

Performance depends on :-

- 1) The transformation
- 2) The quantisation strategy
- 3) The sub-picture size and
- 4) The sub-picture shape

A one dimensional transform tends to be implemented with a concatenation strategy, in which an entity having adjacent pixels in a two dimensional sense has then "separated" as shown in figure 6.7, in order to form a long single dimensional array.

As an alternative, the two dimensional method maintains the adjacency. According to Gonzalez and Wintz Transforming an $n \times n$ array yields better performance with an advantage of the order of 0.2 bits per pixel when using the same quantiser and transfer coding scheme.

6.3.5 Two dimensional FFT implementation

The FFT has a characteristic "separability" property which allows a specific and particular advantage. This is that the two dimensional transform, or its inverse can be achieved in two steps.

- 1) Application of the 1-D transform
- 2) A further application of the 1-D transform

If the discrete Fourier transform pair are expressed as follows, the method becomes clear.

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} F(x, v) \exp[-2\pi u x/N] \dots \dots (1)$$

where $F(x, v) = N \left(\frac{1}{N} \sum_{y=0}^{N-1} f(x, y) \exp[-2\pi v y/N] \dots (2) \right.$

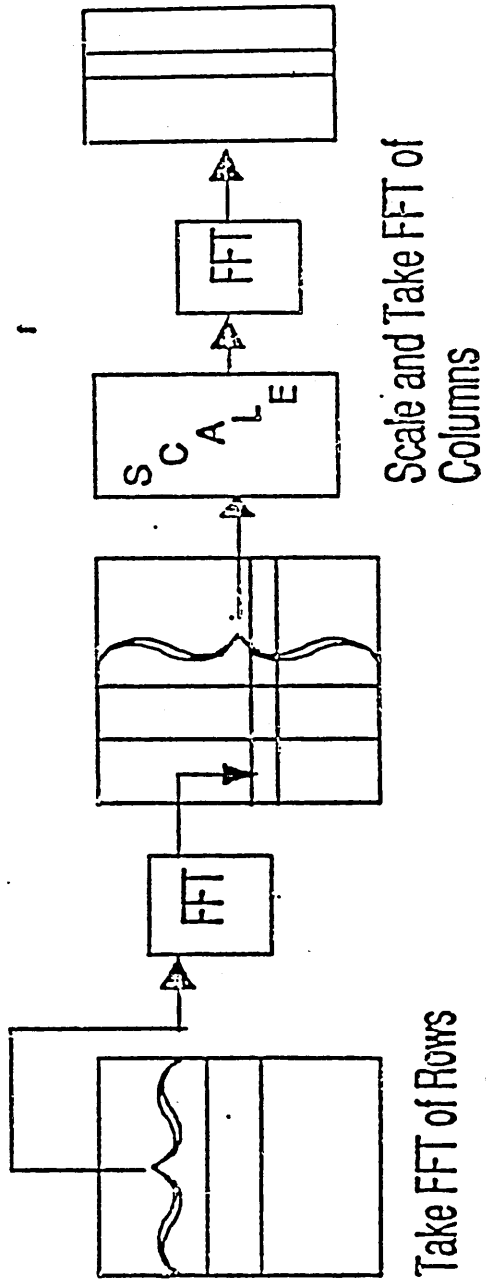


Figure 6.8

In (2), each value of x inside the brackets is a 1-D transform with frequency values $v=0,1,2,3,\dots,N-1$.

The function $F(x,v)$ is inferred to mean that it can be derived by taking the 1-D transform along the x or row axis and then multiplying by N .

The final result $F(u,v)$ is then obtained by taking the 1-D transform along the columns of $F(x,v)$ as expressed by (1). The computational sequence is found in Zoran^[14], and figure 6.8 gives a diagrammatic illustration.

6.3.6 FFT Instruction Parameters

In order to execute the algorithm in figure 6.8 it is first necessary to consider the bit order problems associated with Sande-Tukey and Cooley-Tukey options and make an appropriate option.

An assumption of an 8x8 sub-image will be made, and the first operation requires the loading of VSP internal RAM. The Load instruction gives an opportunity to manipulate the order of the data.

```
LD NMPT:64,MDF:2,ZR:1,RV:3,MBS:8,MSS:8,MBA:SUB-PICTURE;
```

This loads 64 points, and $RV:3$ results in data of size MBS being accessed in bit reversed order whilst blocks are accessed in normal order. Making this decision implies a bit reversed start condition for the FFT and this in turn has fixed the option as Sande-Tukey.

Other terms in the fields of the instruction are defined as follows :-

MDF:2 - memory data format- sequential data in external memory is loaded into the REAL part of internal memory.

The normal expectation is that image data will be all real i.e. grey scale for pixels and all values will be non-negative.

MBA - The address of the sub-picture in external RAM

MSS - Memory step size8

MBS - Memory block size ...8

These last two parameters load all 8 pixel values in an 8 word block.

The next step in the algorithm is to calculate the FFT of the rows, leading to the instruction :-

```
FFT NMBT:64,R:1,FSIZ:8,FPS:1,LPS:4;
```

R:1 selects the Sande-Tukey option, NMBT:64 determines that 64 butterflies are to be performed per pass.

FSIZ:8, is the size of the FFT. Making this parameter less than the NMBT allows multiple SIMULTANEOUS FFT's to be performed in this case.

8 simultaneous row FFT's are performed to complete the row FFT operation. The result of this method is to produce normally ordered data at this point.

The difficulty with the columnar FFT's is that the

x_0	y_0	--	x_1	y_1	--	x_2	y_2	etc
-------	-------	----	-------	-------	----	-------	-------	-----

x_0	y_0	
x_1	y_1	
x_2	y_2	

Figure 6.9

individual elements of each column are spaced at 8 bit intervals. Furthermore, the concept of these elements as an array means that each column array is interleaved, as shown in figure 6.9. The flexibility in defining the instruction parameters means that this is manageable. The interleaved data demands just 3 passes of a 64 point FFT to achieve the equivalent of the necessary three passes of an eight point FFT.

The key instruction field values are :-

```
FSIZ:64,FPS:32,LPS:8;
```

yielding an instruction as follows :-

```
FFT NMBT:64,R:0,FSIZ:64,FPS:32,LPS:8;
```

Note that starting with normally ordered data requires the use of the Cooley-Tukey algorithm. FSIZ:64 indicates the size of the FFT, R:0 selects Cooley-Tukey.

The resulting prescribed action is illustrated by figure 6.10.

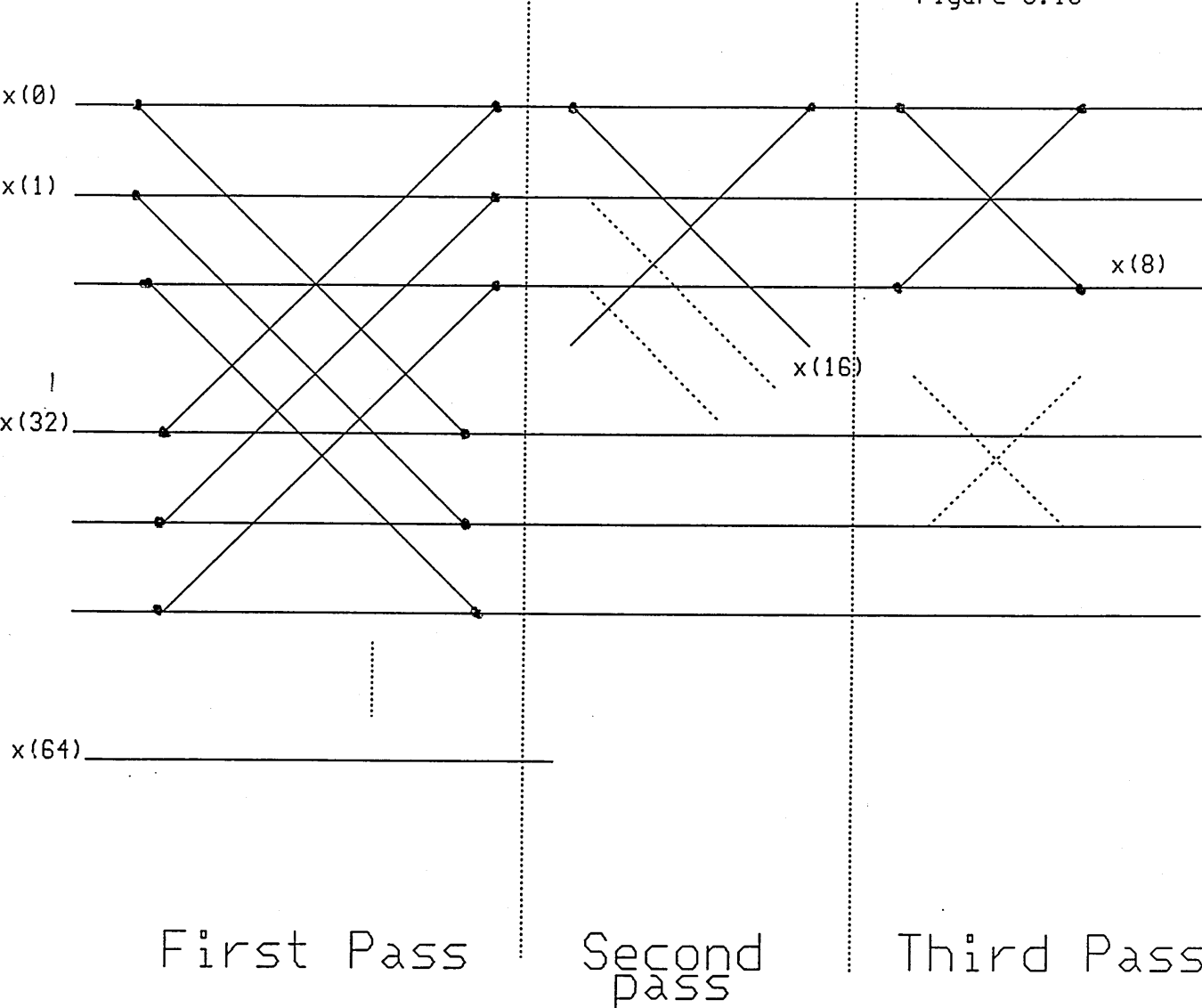
After the third pass, elements of the array separated by 8 bits are subjected to the butterfly action. The parameter LPS:8; stops the action at this point.

After this process the results are grouped in sequential order of the interleaved vector elements. The original organisation is restored by the instruction :-

```
ST NMPT:64,MDF:3,RV:2,MBS:8,MSS:8,MBA:SUBPICTURE;
```

MDF:3 writes real and imaginary internal RAM locations to sequential external RAM.

Figure 6.10



RV:2 Reverses internal bits within blocks of size MBS, whilst preserving the order of the blocks.

Larger FFT's than 8x8 e.g. 16x16 cannot be achieved by a single FFT instruction. Special techniques are available if necessary.

6.3.7 The Fast Cosine Transform

Gonzalez and Wintz^[1] show that the performance of the Hotelling Transform is approached by the Cosine Transform when the original data is strongly correlated as is usually the case with image data.

The Vector Signal Processor is designed for array operations and for the Fast Fourier Transform, but the decision to use the Cosine Transform prompted the investigation into which of several fast discrete options which proceed via the FFT are most appropriate.

Alaul Haque⁽¹⁴⁾ proposes a 2-D algorithm working directly on 2-D data sets. The method involves the partitioning of matrices and the subsequent regrouping of submatrices. For large arrays the computational and floating point overheads are great.

Ghanbari and Pearson⁽¹⁵⁾ suggest a Fast Cosine Transform (FCT) algorithm based on Hadamard sparse matrices, the bonus being that coefficients assume only 1,0,-1 states, allowing relatively simple hardware implementations. Since the VSP is quite happy with floating point operations this is an unnecessary limitation.

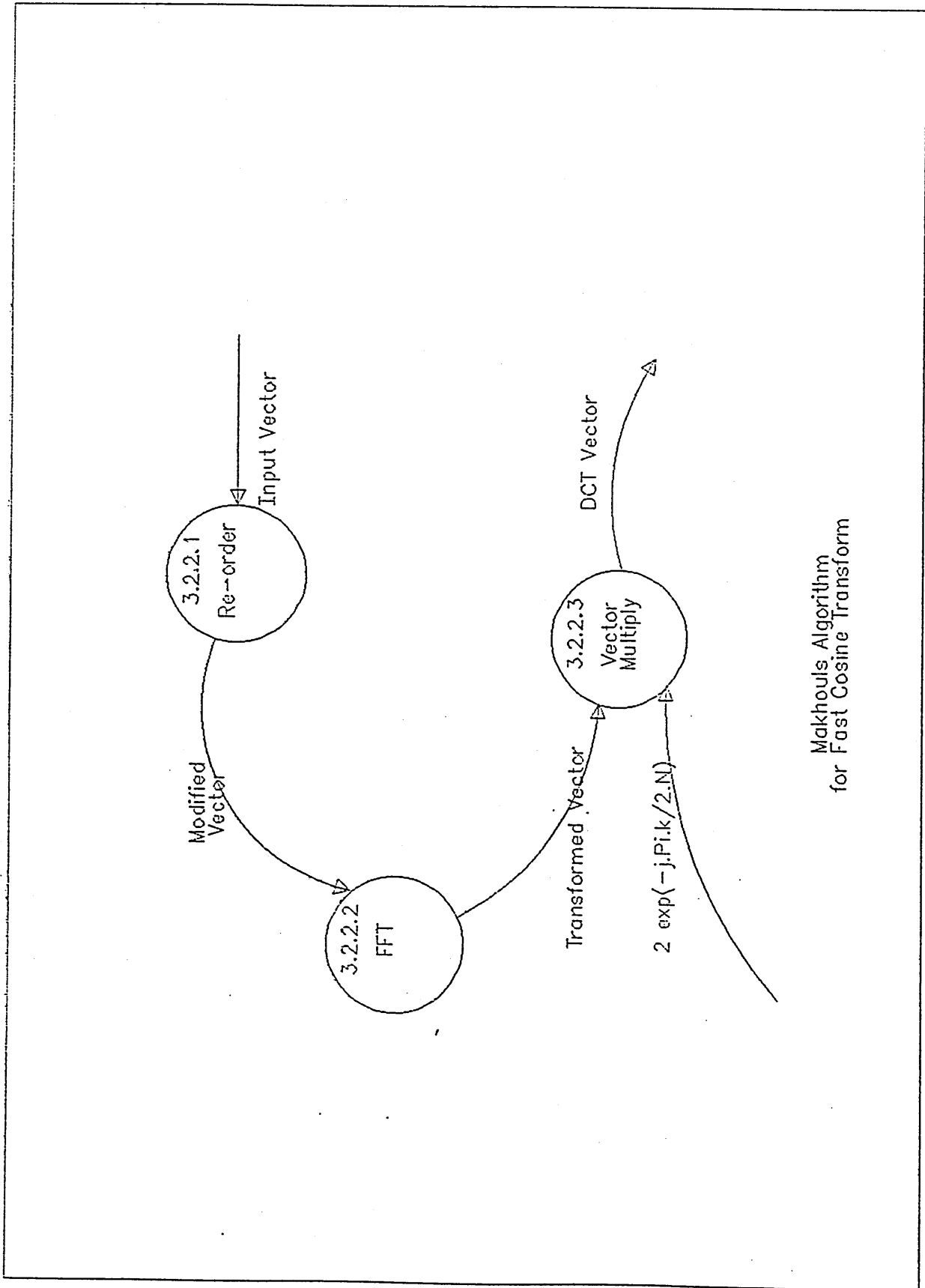


Figure 6.11

Byeong Gi Lee⁽¹⁶⁾ and Makhoul⁽¹⁷⁾ describe FCT implementations based on the Fast Fourier Transform (FFT). Both of these methods are used in a procedure which re-orders the input data.

The development choice was Makhoul's algorithm, which is defined by figure 6.11.

If the input data sequence is $x(n)$, then the re-ordered data $v(n)$ is derived as follows :-

$$v(n) := \begin{cases} x(2n) & 0 \leq n \leq \left\lfloor \frac{N-1}{2} \right\rfloor \\ x(2N-2n-1) & \left\lfloor \frac{N+1}{2} \right\rfloor \leq n \leq N-1 \end{cases}$$

Where $\lfloor \quad \rfloor$ indicates "the integer part of". This means that the re-ordered sequence is obtained by taking the even points in $x(n)$ in order, followed by the odd points in reverse order. The method then calls for the DFT of $v(n)$ to yield $v(k)$. This may be executed as an FFT. The final step is to multiply the array $v(k)$ by $2 \exp[-j\pi k/2N]$.

6.3.8. VSP implementation of the FCT

Assuming a normally ordered array of samples of size N residing at memory base address (MBA), $MBA:SUBPICTURE$, the first task is to load the array into the internal memory of the VSP system - even points first.

```
LD NMPT:N/2,MBS:1,MSS:2,RV:0,MBA:SUBPICTURE+1;
```

This instruction loads half the number of points selecting every other point beginning with point 2, normal order. Next the odd points must be loaded in reverse address order. The RV field

is capable of prescribing bit reverse operations on addresses or data, but what is required here is a re-ordering which can be achieved by loading in normal order, storing backward, and then loading again in normal order. The bit reversal operations are provided for certain FFT operations and are not appropriate to Makhoul's algorithm. The instruction sequence needed is :-

```
LD NMPT:N/2,MBS:1,MSS:2,RV:0,MBA:SUBPICTURE;  
STB NMPT:N/2,MBS:N/2,MSS:N/2,MBAB:SCRATCH+N/2;  
LD NMPT:N/2,MBS:N/2,MSS:N/2,MBA:SCRATCH;
```

The STB instruction stores data arrays whilst decrementing the memory base address. MBAB defines the base address when storing backward. The results are that data is loaded into VSP memory as follows: $x(2), x(4), x(6), \dots, x(n/2), \dots, x(7), x(5), x(3), x(1)$.

It is necessary to have precalculated $2\exp(-j k/2N)$ for $0 \leq k \leq (N/2)$. This task is best done by the VSP and to this end the instruction set provides a special instruction.

The DEMO instruction multiplies a complex vector in internal memory by a series of complex coefficients generated from a look up table which contains 256 Cosine values ranging from 0 to 90 degrees. After multiplying each element of the specified Cosine vector with corresponding array in internal RAM, the products are placed in the internal RAM also.

There are a number of microword control fields unique to the DEMO instruction. These are :-

RBA - ROM base address

This parameter defines the starting angle of the Cosine vector, and is set to 10x the value in degrees. i.e. to start at

90 degrees RBA=900. The internal look up table contains 1024 discrete values and this makes only some values for RBA legal. The rule is :-

If "i" is the angle, $i*3600/1024$, where $0 \leq i < 1024$

RDA - ROM decrement address.

This address is used to define the incremental angles for successive Cosine coefficients. As with RBA the parameter is specified as 10x the required value in degrees. Legal values belong to the set ;

$(i+1)*1800/512$, $0 \leq i \leq 511$.

VSIZ - Vector size

Specifies the number of samples beginning with RBA to be addressed from the internal look up table. After this value the look up table address reverts to the RBA value. Legal values are 4,8,16,32,128 points.

The sequence of instructions required is as follows :-

FFT

DEMO (multiply by $2\exp(-j k/2N)$)

ST (leave result back in external memory)

A full program listing appears in Appendix B.

The inverse FCT developed by Makhoul requires that the input data be first complex conjugated, rotated by the MODLT instruction and then inverse FFTed. The MODLT instruction is exactly analogous to the DEMO instruction except that the frequency is translated up in frequency instead of down. Complex conjugation

is specifically provided by the CMCN VSP instruction. The CMCN is not required for the forward transform and so the inverse transform takes about 10% longer to execute.

6.4 Transfer Coding

A spatial domain array and its transformation by the FCT are shown in figure 6.12 and 6.13.

Merely transforming into the frequency domain does not generate a compression of itself. As can be seen by figure 6.13 the transformation has produced strong compression potential with all the high valued coefficients bunched together followed by a long run of similarly valued ones. This long run consists of zero or near zero valued coefficients which may be allocated zero.

These contribute very little and may be ignored or compressed because of their redundancy. Zero valuing small but non-zero coefficients will create some small distortion when the data is reconstituted. The amount of distortion acceptable for a given system will determine what coefficients must be considered significant, and which may be zeroed. The allocation of bits to coefficients either zonally, i.e. only allocating significant values to a particular portion of the frequency spectrum - in this case equivalently low pass filtering, or by threshold i.e. zeroing all values below a given amplitude, is a function of quantising.

Whichever quantising equivalent scheme is chosen, and the effects of the threshold type are illustrated in figure 6.14, the

Normally ordered data samples
 65, 34, 44, 55, 66, 77, 128, 233, 112, 23, 15, 43, 71, 99, 100, 222

```

945
557
233
243
992
100
171
115
128
160
445
  
```

```

p :- f :- fft(p)      j :- 0 .. 8      i :- 0 .. 15
  
```

```

h_j := f_j * [ 2 * exp [ -1 * pi * j / 2.15 ] ]
  
```

```

544
400
607
128
112
99
100
222
  
```

```

> :-
  
```

Figure 6.12

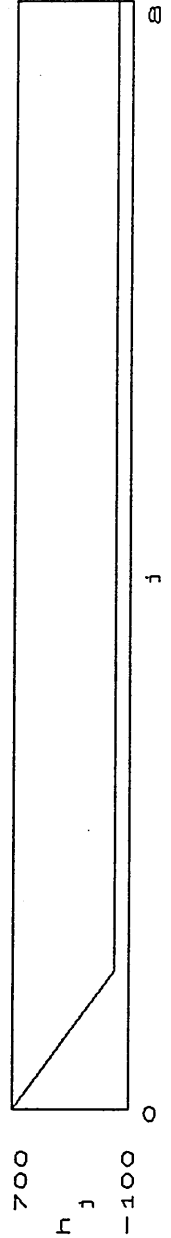
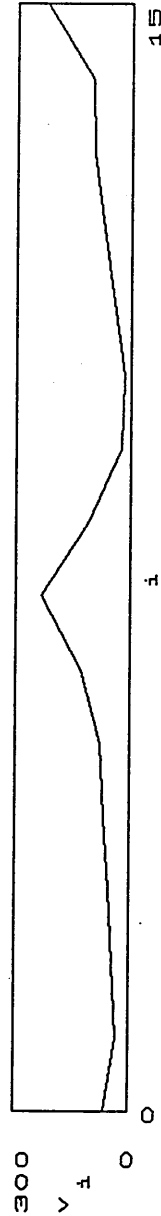
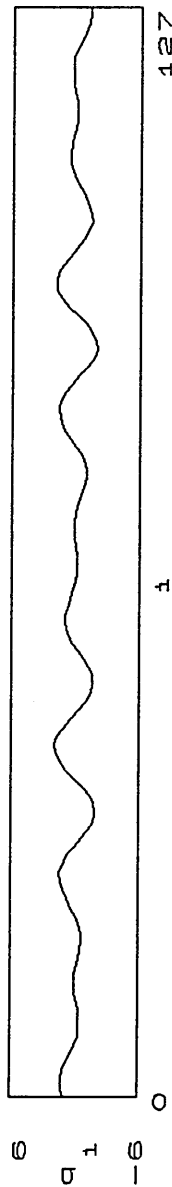


Figure 6.13

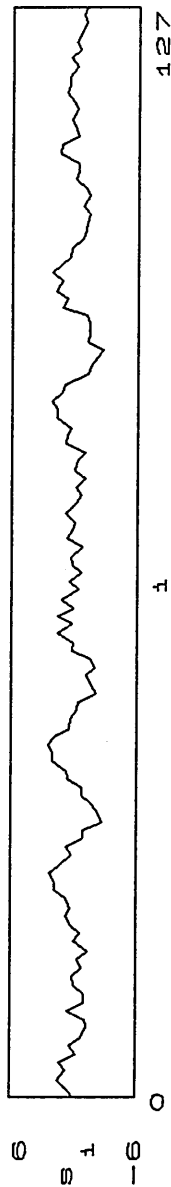
Define the signal: `i := 0 .. 127`

$$a_i := \sin \left[\frac{i}{128} \cdot 14 \cdot \pi \right] + \cos \left[\frac{i}{128} \cdot 19 \cdot \pi \right]$$



Add some noise:

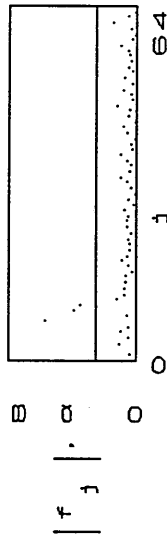
$$s_i := a_i + \text{rnd}(2) - 1$$



Take its discrete fourier transform:

$$f := \text{fft}(s) \quad j := 0 .. 64$$

$\alpha := 2.5$...define threshold for spectral noise rejection



<--- Signal

<--- Noise

Filter, and take the inverse transform:

$$g_j := f_j \cdot \mathbb{I} \left[|f_j| - \alpha \right] \quad h := \text{ifft}(g)$$

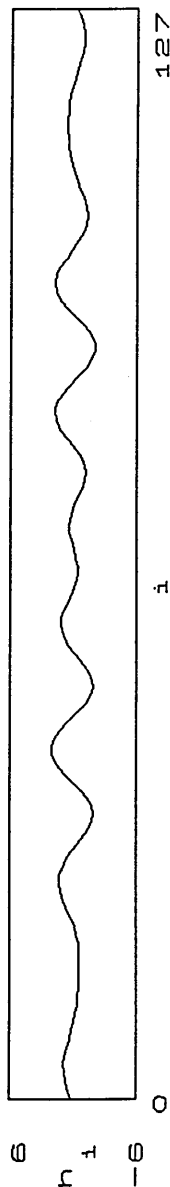


Figure 6.14

result is that a Run length or Huffman scheme can provide compression. Zonal quantising is particularly easily implemented with the Vector Signal Processor as is demonstrated in Section 7.0 during a discussion on edge detection by filter.

The threshold quantising is more difficult and best achieved by the host processor. This is because the VSP does not have instructions capable of testing for value.

References

- [1] Gonzalez, R.C., Wintz P., "Digital Image Processing" Addison Wesley, 1987.
- [2] Chatfield C., "Statistics for Technology", Chapman and Hall, 1978.
- [3] Hotelling H., "Analysis of a complex of statistical variables into principal components", J. Educ. Psychol, vol 24, pp417-441 and 498-520, 1933.
- [4] Karhunen, translation by Selin I., "On linear methods in Probability Theory", T-131 The RAND Corp, 1960.
- [5] Loeve M., "Fonctions aleatoires de seconde ordre", Processus Stochastique et mouvement Brownien, Hermann, 1948.
- [6] Boyle R.D. and Thomas R.C., "Computer Vision", Blackwell Scientific, 1988.
- [7] Lynn P.A., "An introduction to the analysis and processing of signals" Methuen, 1983.
- [8] Pearl J., "On coding and filtering stationary signals by discrete Fourier transforms", IEEE Trans. on Info Theory, pp229-232, March 1973.

- [9] Cooley, J.W., Tukey, J.W., "An algorithm for the machine calculation of complex Fourier series", Math. of Comp., vol-19, pp297-301, 1965.
- [10] Hutton S., "Handling complex data in digital signal processing", Electronic Product Design, pp61-65, November 1987.
- [11] Taylor D.M., "Single commands for complex DSP functions" Electronic Product Design, pp31-42, November 1987.
- [12] Zoran, Technical Notes TN 92040-0187, pp3-5
- [13] Karplus W.J., "Architectural and software issues in the design and application of peripheral array processors", Computer vol 14. No.9 pp 11-17, 1985.
- [14] Alaul Haque M., "A two dimensional fast cosine transform", IEEE Transactions on Acoustics, Speech, and Signal Processing. Vol ASSP-33 No6. pp 1532-1539, 1985.
- [15] Ghanbari M., D.E. Pearson, "Fast cosine transform implementation for television signals", IEE Proc Vol 129 Pt F, No.1, pp 59-68, 1982.
- [16] Byeong Gi Lee, " FCT - a fast cosine transform", IEEE Proc, pp 28A.3.1-28A.3.4, 1984.
- [17] Makhoul J., " A fast cosine transform in one and two dimensions", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol ASSP-28, No.1, pp 27-34, 1980.

7.0 Quadtree encoding + Delta modulation

7.1 The application

Section 5.0 is concerned with textual characters and Section 6.0 with grey scale photographic pixel data. There is a considerable class of images not covered by these sections, namely line drawings.

Characteristically, line drawings are often sparse and consequently the potential for redundancy reduction and the analysis and run length coding in Section 4.0 could be used. Figure 4.1 contains line drawings of increasing sparseness, for example.

The best results achieved are listed in arrays E and F which show compression ratio maxima of the order of 30:1. The average compression for a range of images was approximately 5:1.

An alternative scheme was devised and only when its performance was quantified was the applicability of the VSP considered. The origins of the alternative are well known but its use in combination with a form of Delta modulation and the detailed encoding are thought to be original.

7.2 Quadtree representation

The quadtree encoding principle was the subject of early work by Tanimoto^{[1],[2],[3]}, and Chen^[4] and workers in this area often return to it, for example Zhongqiang Li and Telfer^[5].

Gonzalez and Wintz^[6] use quadtrees under the general heading of Region Orientated Segmentation, specifically defining

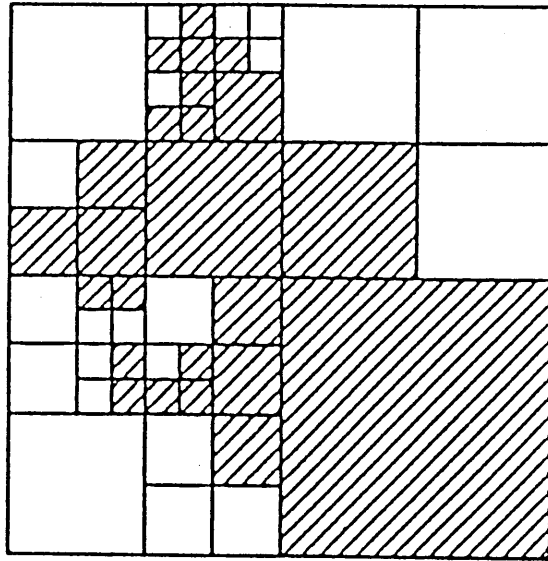


Figure 7.1

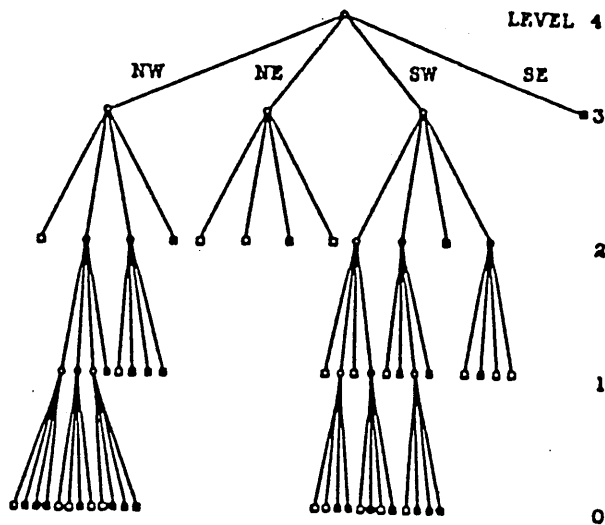


Figure 7.2

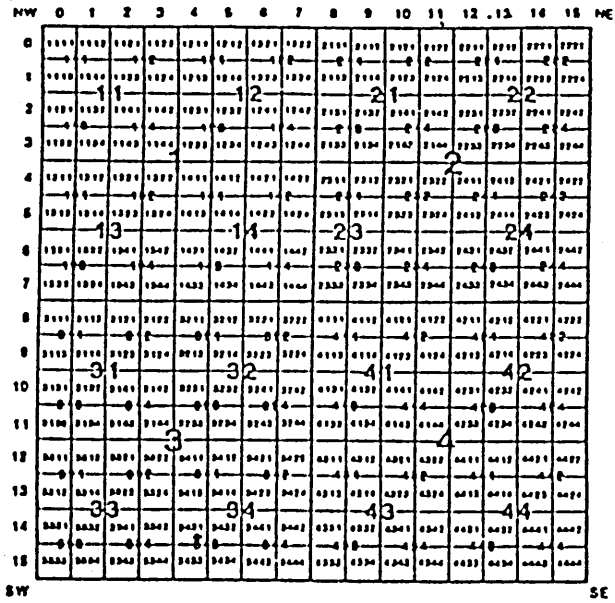


Figure 7.3

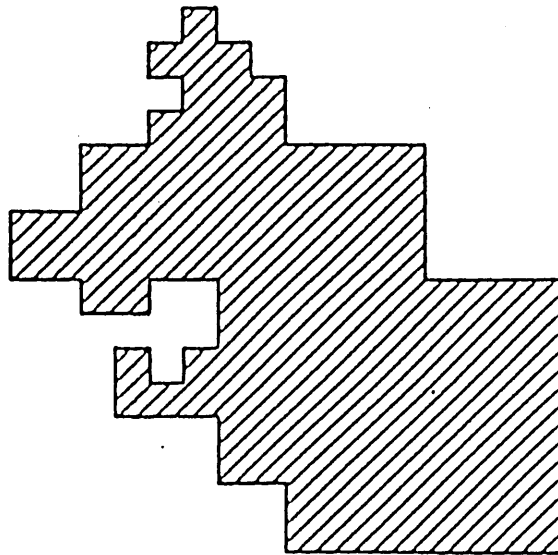


Figure 7.4

a region splitting and merging algorithm.

Clearly, quadtrees have been found to be useful in several areas of image processing, but the interests of this research are particularly data compression.

Consider figure 7.1. A quadtree may be constructed from a square binary array of pixels which represent an image. A set of connected black pixels in the image are referred to as a region.

If it is assumed that an image is comprised of a $2^n \times 2^n$ binary array of pixels, then a quadtree encoding represents the image by recursively sub-dividing it into four quadrants until no further subdivision is necessary. This process is illustrated by figure 7.2 which is the pyramid form of representation that spawned the name "quadtree".

7.2.1 Quadtree addressing scheme

Figure 7.3 shows a possible addressing scheme for quadtrees. The major quadrants are number 1,2,3,4 corresponding to the north-west,north-east,south-west,south-east quadrants of the sub-picture. The redundancy reduction technique of only recording the black pixels is used, but each node or pixel has a unique key.

This key can be processed to show the level at which it was formed and also the x,y position within the complete subpicture. The list of keys is, by convention, given in order from the origin.

As an example, the region shown in figure 7.4 is encoded as follows :-

Key 4000 refers to the whole south-east quadrant, 4400 to the south-east quadrant of 4000, and so on.

Using this principle the region in question is encoded as :-
1212,1213,1214,1223,1232,1233,1234,1240,1320,1330,1340,1400,2300
3121,3122,3142,3144,3220,3232,3233,3234,3240,3420,4000

0's are used to fill all undefined points.

7.3 Delta Modulation

The addressing scheme above does not produce any worthwhile data compression, for instance figure 7.1 in its raw data state comprises of 16X16 bit words. The encoding sequence above calls for 24 words of at least 13 bits resulting in negative compression in this case. If there were far fewer black pixels to define a small compression could be achieved. On average the overall compression for a practical picture is, as stated, hardly worthwhile.

To obtain data compression a significant saving in word size is necessary and to achieve this Delta modulation was applied, using the rule that the prediction would be the same as the previous sample. This is justified by the fact that the encoding sequence exhibits a "locality of reference", that is samples are very close in value to the one which preceded them.

In this way the north-west quadrant of the example previously encoded :-

1212,1213,1223,1232,1233,1234,1240,1320,1330,1340,1400

becomes :-

1212,+1,+10,+9,+1,+1,+6,+80,+10,+10,+60

This has not reduced the number of elements representing the region, but it has allowed 8 bit words to be used in each case with the exception of the opening sample. This problem was tackled by utilising a "never used" digit e.g. 5. This number was chosen arbitrarily from the range of digits 5 to 9.

Thus the opening element 1212 becomes a two part entity :-
51,+212 and the full sequence for the example region is :-

51,212,1,10,9,1,1,6,80,10,10,60

The code 5 means "first digit of literal starting value for a quadrant follows". The second word, in this case 212 could have a maximum value of 444. To accommodate this within 8 bits an alternative starting code 8 is used so that 4444 becomes 84,+244. Code 8 means the same as code 5 except that 200 must be added to the following word.

The result of this scheme is that the original 16X16 subpicture is signalled or stored as 28X8 bit words. This represents a small compression, 1.14:1.

The notion of additional "special codes" was extended to include :-

60 - meaning new sub-picture starts

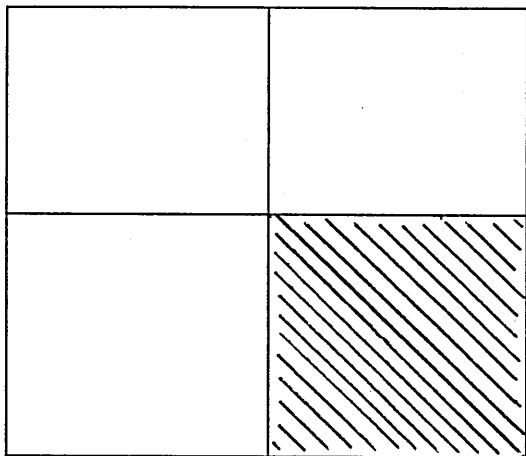
70 - meaning delta modulation sequence repeats

66 - meaning complete sub-picture blank or devoid of 1's

Using the full encoding options figure 7.5 would be encoded as 60,54 and figure 7.6 which has a single line one pixel wide horizontally across the sub-picture would produce the sequence :-60,51,131,1,9,1,89,1,9,1

52,70

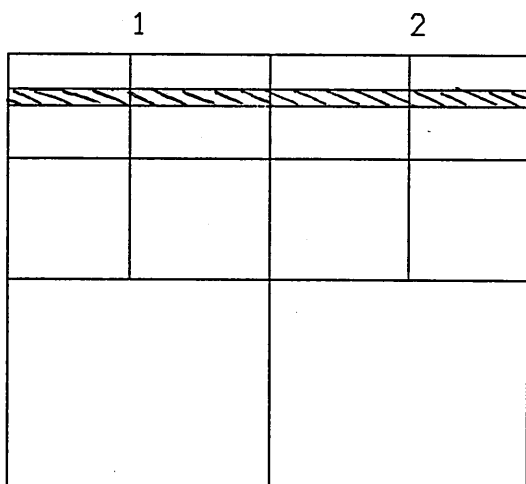
figure 7.5



4000

+60,+54

figure 7.6



+60,+51,+131,+1,+9,+1,+89,+1,+9
+52,+70.

key:
+60 start
+5X quadrant (X)
+70 repeat sequence

A total of 12 bytes. The sub-pictures being used are 16x16, thus the sub-picture represented is equivalent to 32x8 bit words and a compression of 2.67 is achieved.

The maximum compression possible will occur if a sub-picture is completely blank, as might be expected. In this case it is necessary to signal "new sub-picture", i.e. prefix 6, but it also necessary to signal "all blank". The special code word suggested is 66.

For this circumstance the compression ratio will be 32:1. When compared with the run length methods described in Section 4.0 this result appears surprisingly good. The derivation of the scheme is purely theoretical and no implementation was done. The theoretical performance cannot therefore be confirmed.

The scheme offers two possible advantages not easily obtained from Huffman style operation.

These are :-

- (1) That it provides a high level view of spatial entities in images.
- (2) That it allows spatial entities to be analysed.

Many line graphics images contain a large proportion of vertical or horizontal lines. The "repeat" delta modulation sequence commands are applicable when lines cross any major quadrants. Figure 7.6 shows an example of this.

The effects of errors on run length coding are to corrupt a single line of pixels and in READ coding the error would propagate throughout the drawing as described by Fuchs [8].

Errors in the quadtree system will only corrupt a localised 2-dimensional 16x16 block.

7.4 VSP implementation

The array instructions provided by the VSP are not particularly applicable to the encoding process which is more suited to bit operations available with a standard microprocessor. Alternatively, the bit operations within the C high level language are ideal. This scheme was not proceeded with past the conceptual stage because of the inapplicability to the VSP.

References

- [1] Tanimoto,S.L., "A pyramid model for binary picture complexity", Proc.IEEE Comp.Soc. Conf on Pattern Recognition, pp25-27,1977.
- [2] Tanimoto,S.L., "Image transmission with Gross information first", Computer Graphics and Image Processing No.9,pp72-76,1979.
- [3] Tanimoto,S.L., "A hierarchical data structure for picture processing", Comp. Graphics and Image Processing, vol14, No.2, pp104-119 ,1975.
- [4] Chen C.H.,"Pattern recognition and artificial intelligence" Academic Press pp452-471,1982.
- [5] Zhongqiang Li,Telfer,D., "Primitive quadtree and type code quadtree approaches for the representation of binary regions", IEE digest no.1989/53,pp3/1-3/7,1989.
- [6] Gonzalez,R.C.,Wintz,P., "Digital image processing", Addison Wesley,1987.

[7] Fuchs, P.M., "Compressing data conserves memory in bit mapped displays", EDN, pp173-183, October 1986.

8.0 Image Manipulation using the Vector Signal Processor

8.1 Introduction

Previous sections have assumed that entities can be isolated from the background in graphics images. In Section 5.0 it was suggested that a chain coding scheme could be used for this purpose. The algorithm for tracing an entity boundary by chain code is rehearsed in order to determine the compatibility of the process with the VSP. If objects have been isolated they may be operated on for compression purposes as in Section 5.0, but it is a short step to considering other manipulative operations such as might be appropriate to desk top publishing. Once again it is compatibility with the VSP which is a prime consideration and suitable ideas are developed.

8.2 Object representation

This research is concerned with the low level operations performed by specific hardware. In this context the representation of objects encompasses detecting their outline once they have been segmented out of the overall image.

When the outline has been detected it will need to be described in a manner suitable for operating on the object in order to manipulate it.

8.2.1 Edge Detection

Boyle and Thomas^[1] explain that although primitive edge detection is much less ambitious than template matching as discussed in Section 5.0, there is, nevertheless a similarity between the ideas.

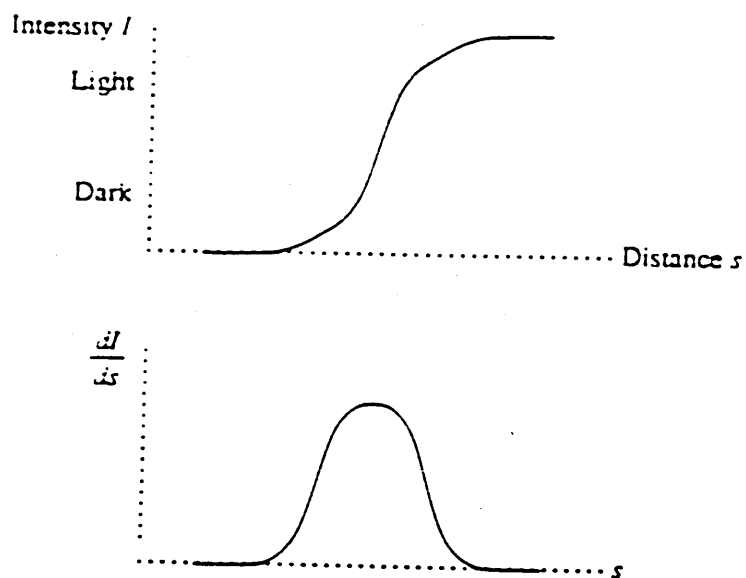


Figure 8.1

0	1
-1	0

1	0
0	-1

Figure 8.2

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Figure 8.3

Typically, primitive edge detection consists of passing a template or templates across the image in an attempt to detect sharp changes of intensity. When found these are good evidence of an edge or boundary of an object.

What is required is to examine the rate of change between adjacent pixels. Figure 8.1 shows first order detection. Special templates have been developed by Roberts^[2] and Sobel^[3] which enable the gradient magnitude at any point to be calculated. These templates are shown in figure 8.2 and 8.3. It should be noted that these templates, often called "edge operators", are provided in pairs with sensitivities of each adjusted to give a measure of intensity change in two orthogonal directions. Generally, two numbers are generated and then used to calculate the gradient magnitude. The process for the Roberts case is shown below :-

$$\begin{aligned} \Delta_1 &= f(i,j) - f(i+1, j+1) \\ \Delta_2 &= f(i+1, j+1) - f(i, j+1) \\ \text{gradient magnitude } S(i,j) &= (\Delta_1^2 + \Delta_2^2)^{\frac{1}{2}} \end{aligned}$$

The Vector Signal Processor is appropriate for these kinds of operation, being particularly fast at passing the template over the image and producing a new array upon which the numerical processes could be applied.

8.2.2 Filtering

The basic formulation of edge detectors calls for localised derivative operators. The template methods approximate this by differences. One of the premises underlying the use of the Vector Signal Processor is that the image pels are treated as a

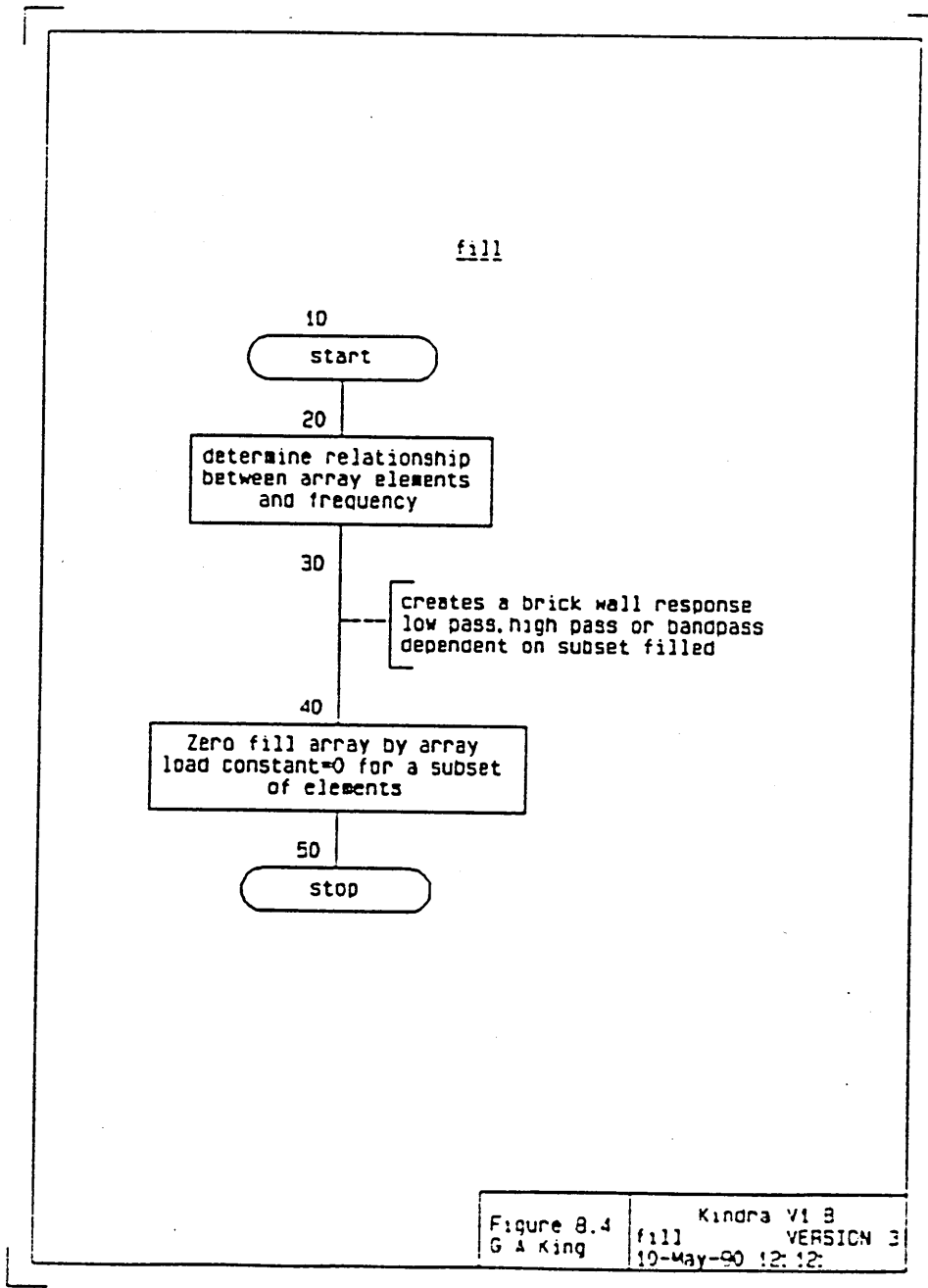


Figure 8.4

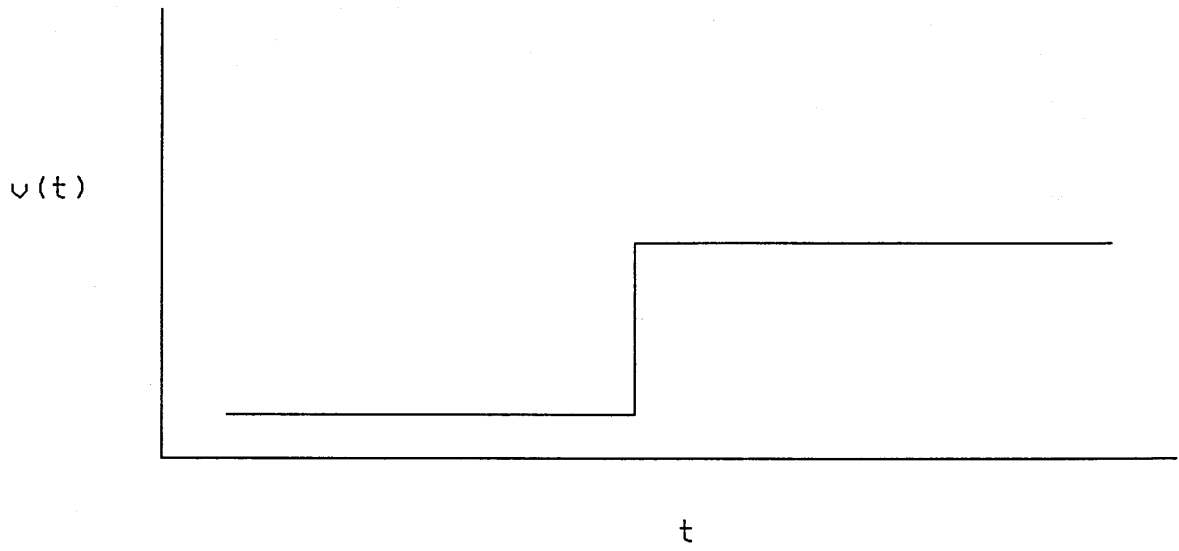


FIGURE 8.5

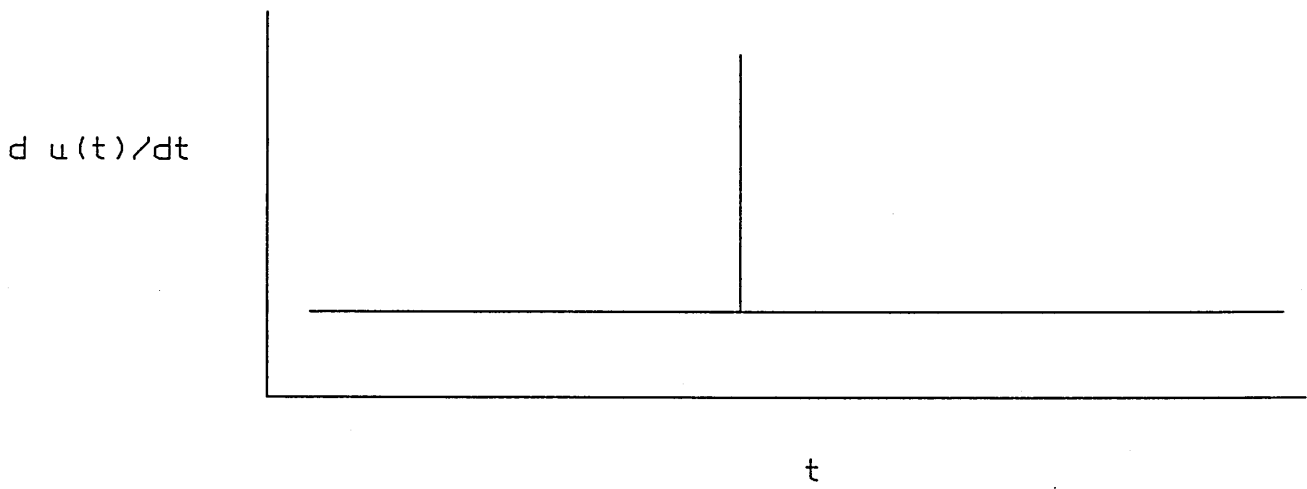


FIGURE 8.6

signal. The classic signal differentiator is the high pass filter.

Edges and other high rate discontinuities are associated with the high frequency components in the signal. Edge detection may be achieved by attenuating the low frequency components whilst retaining the high frequency components.

The ideal highpass filter has a transfer function valued at zero below a cut-off frequency. The ideal "brick wall" characteristic cannot be realised physically, but it is a simple matter to create these within a computer.

The basic algorithm is shown in figure 8.4 . The action is described by considering figure 8.5 a sample line traversing an image object. The result of ideal filtering is shown in figure 8.6.

In common with the templating methods two passes would be necessary to identify vertical and horizontal edges. The vertical edges are detected by a 1-D transform. The method described in Section 6.0 illustrated by figure 6.9 shows the interleaved column data upon which a further FFT is performed in order to detect horizontal edges.

Underlying the differentiated image is an implicit co-ordinate system. What is then needed is a means of defining the edge demarcating the object boundary.

8.3 Chain Codes

The basic idea behind the chain code is that from a known starting point the direction followed by an edge is indicated as a straight line segment pointing in any of eight possible

Figure 8.8

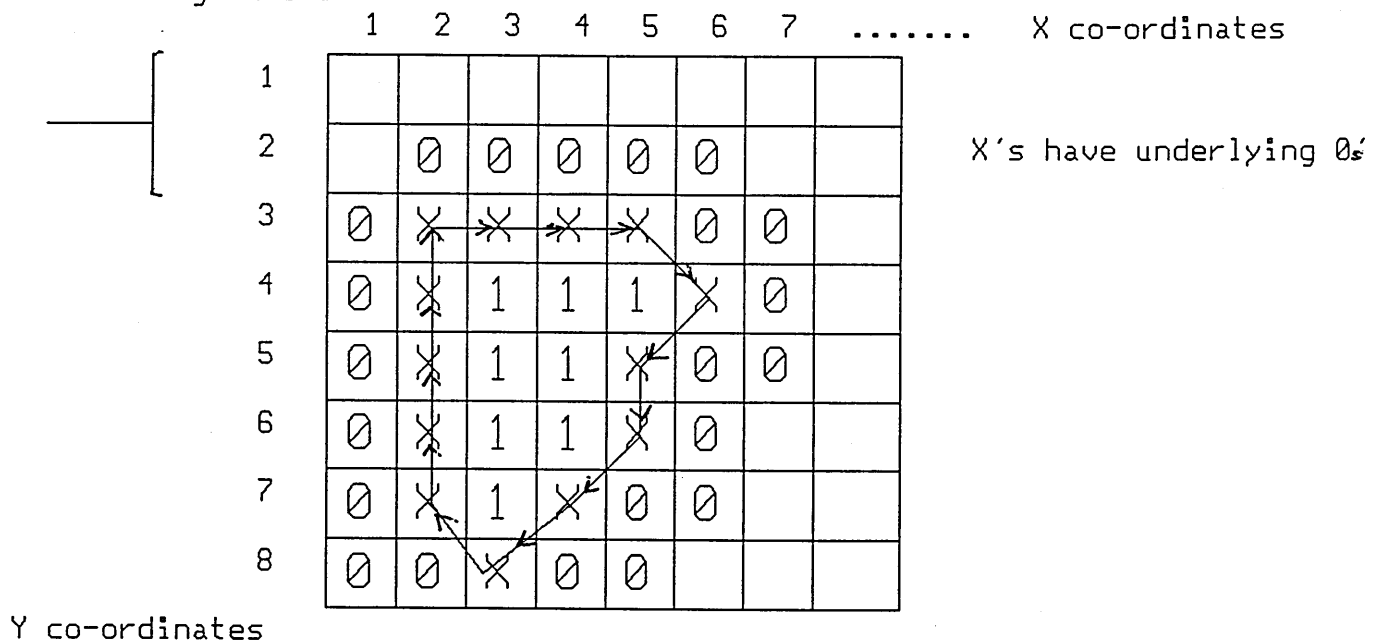
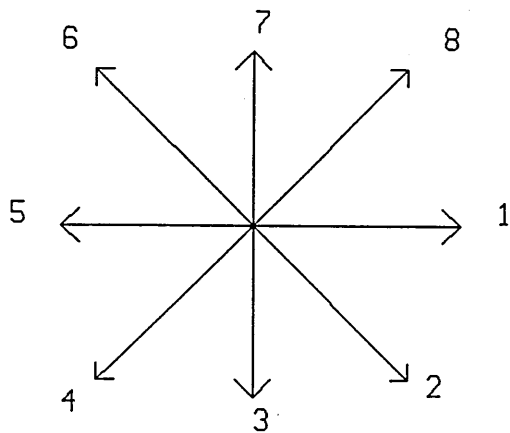


Figure 8.7



directions. The directions are shown in figure 8.7.

The algorithm required to trace the boundary starts by first finding, by means of an array search, the upper left neighbour of a non-zero mark. This is illustrated by pixel grid reference 2,2 in figure 8.8. Once having found this starting point, the algorithm continues with the following steps :-

- 1) Call the starting point L and allocate xy co-ordinates
 $X = 2$ $Y = 2$ $pres$
 sk sk $s=start$ $k=kth$ chain (since

each object defined will have its own chain code).

- 2) Identify the next link candidate by advancing one square in each of the directions in the following order 1,2,3,4,5,6,7,8 as defined in figure 8.7.

- 3) For each candidate check RIGHT NEIGHBOUR for "1", if not advance to next in order. If so, $L = L$ where
 $pres$ $cand$
 $cand=candidate$.

- 4) Make a note of the series of co-ordinates.

To check the squares in the correct order the following sequence is necessary :-

- (i) $X = X$
 sk $sk+1$
 $Y = Y$
 sk sk
- (ii) $X = X$
 sk $sk+1$
 $Y = Y$
 sk $sk+1$
- (iii) $X = X$
 sk sk

- (iv) $Y = Y_{sk, sk+1}$
 $X = X_{sk, sk-1}$
- (v) $Y = Y_{sk, sk+1}$
 $X = X_{sk, sk-1}$
- (vi) $X = X_{sk, sk}$
 $Y = Y_{sk, sk-1}$
- (vii) $X = X_{sk, sk}$
 $Y = Y_{sk, sk-1}$
- (viii) $X = X_{sk, sk+1}$
 $Y = Y_{sk, sk-1}$

In the case of figure 8.8, the series of xy co-ordinates describing the boundary are :-

X	Y	
sk	sk	
2	2	start
3	2	
4	2	
5	2	
6	3	
5	4	
5	5	
4	6	
3	7	
2	6	
2	5	
2	4	
2	3	
2	2	return to start point

These positions derived from the directional sequence 1,1,1,2,4,3,4,4,6,7,7,7 for the object of figure 8.8.

It is purely a conceptual matter to regard the absolute values for the co-ordinates to be in the all positive quadrant of a real+imaginary system. i.e. the references are $2+j2$, $3+j2$, $4+j2$, $5+j2$ etc. This $a+jb$ form will be useful for operations involving Fourier Descriptors. The chain code representation was first proposed by Freeman ^[4].

8.3.1 VSP Chain code implementation

The VSP is not well suited to this task which is easily implemented with the host processor in either high level language or assembler.

8.4 Fourier Descriptors

Gonzalez and Wintz ^[5] discuss taking the Fourier Transform of the sequence of complex numbers representing an object boundary. They correctly refer to the result as a Fourier Descriptor and point out that because the DFT/FFT is a reversible linear transformation, no information is lost.

Contributions to the principles may be found in Persoon and Fu ^[6].

Certain operations can be carried out on the object by operating on its frequency domain representation. These operations are :-

- 1) Scaling
- 2) Rotation

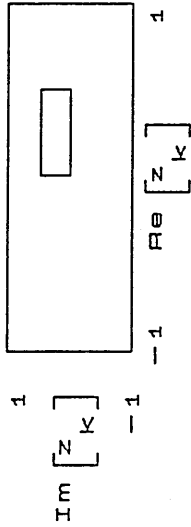
Fourier Descriptors

Define points as a complex vector

Object is a box $k := 0 \dots 4$

$z_0 := 0 + 0j$ $z_1 := .5 + 0j$ $z_2 := .5 + .5j$ $z_3 := 0 + .5j$ $z_4 := 0 + 0j$

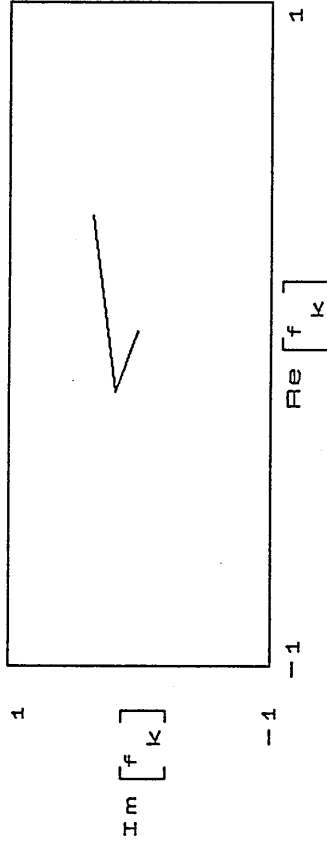
$z_6 := 0 + 0j$ $z_7 := 0 + 0j$



$z = \begin{bmatrix} 0.5 + 0.5i \\ 0.5 + 0.5i \\ 0.5 + 0.5i \\ 0.5 + 0.5i \\ 0.5 + 0.5i \end{bmatrix}$

$f := \text{cfft}(z)$

$f = \begin{bmatrix} -0.354 + 0.354i & 0 \\ -0.177 + 0.177i & 0 \\ -0.073 + 0.073i & 0 \\ -0.177 + 0.177i & 0 \\ -0.354 - 0.354i & 0 \\ 0.427 - 0.427i & 0 \end{bmatrix}$



$q := \text{icfft}(f)$

$q = \begin{bmatrix} 0.5 + 0.5i \\ -4.799 \cdot 10^{-13} - 1.6 \cdot 10^{-13}i \\ -3.2 \cdot 10^{-13} - 3.2 \cdot 10^{-13}i \\ -1.6 \cdot 10^{-13} - 7.999 \cdot 10^{-13}i \end{bmatrix}$

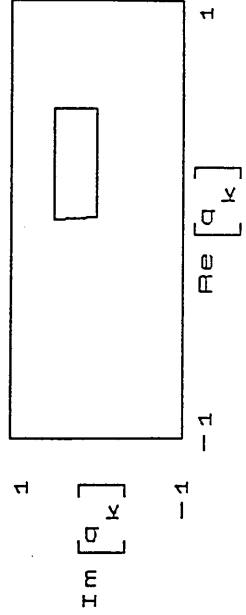


Figure 8.9

3) Change of position

Figure 8.9 demonstrates the generation of a Fourier Descriptor for an object and its subsequent reconstitution. A simple rectangle is described in terms of its four corner positions expressed using $a+jb$ form. The corner references form the elements of array z . The array f is obtained by taking the complex FFT of z . Array q is the reconstituted z obtained by inverse complex FFT of array f .

To scale an object it is only necessary to multiply its FFT by a factor. This is simply an array multiply by a real number. The process is shown in figure 8.10. The same procedure as that used in figure 8.9 is repeated but the FFT, once obtained as array f is multiplied by 8 in this case. After the complex IFFT is performed the original rectangle has been scaled up by 8, as can be seen by comparing the first four elements of arrays z and q .

Rotation is achieved by array multiplying by an exponent as illustrated by figure 8.11. Within $\text{Exp}(jx)$, x is the angle of rotation and is a constant for any particular calculation. Although the rotated object appears distorted in figure 8.11, this is not due to a discrepancy in the process but to a deficiency in the laser printer scaling and resolution.

8.4.1 Fourier Descriptor manipulation with the VSP

Movement of the position of an object entails adding constants to the real and complex parts of each reference point on the boundary. Vector add, real and complex are standard instructions. The task is performed in two steps.

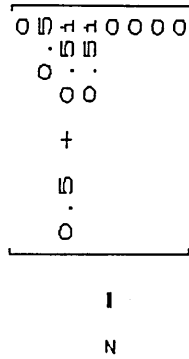
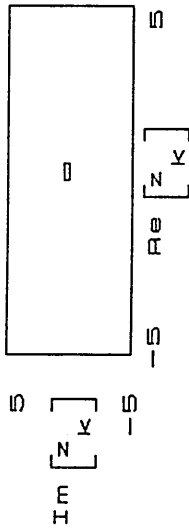
Fourier Descriptors - Scaling

Define points as a complex vector

Object is a box $k := 0 \dots 4$

$z_0 := 0 + 0j$ $z_1 := .5 + 0j$ $z_2 := .5 + .5j$ $z_3 := 0 + .5j$ $z_4 := 0 + 0j$

$z_5 := 0 + 0j$ $z_7 := 0 + 0j$



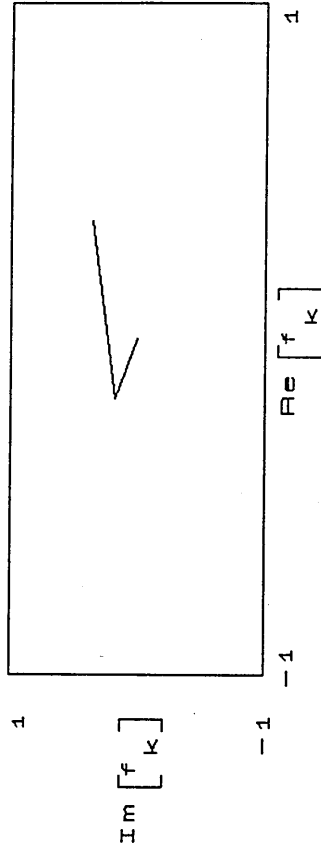
$f := \text{cfft}(z)$

```

0.3541 + 0.3541i
-0.1771 + 0.1771i
-0.0731 + 0.0731i
-0.1771 + 0.1771i
-0.3541 - 0.3541i
0.4271 - 0.4271i

```

$f :=$



$f := f \cdot 8$

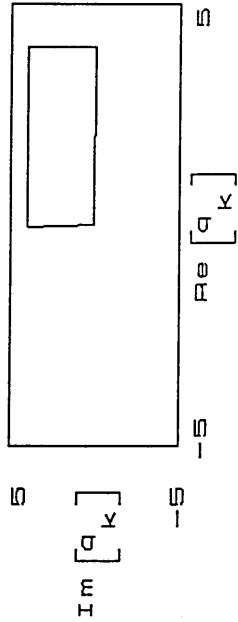
$q := \text{icfft}(f)$

```

0.4
4 + 4i
-3.83910 - 1.2810i
-2.5610 - 2.5610i
-1.2810 - 6.39910

```

$q :=$



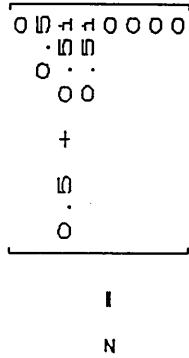
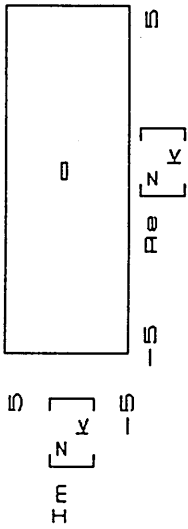
Fourier Descriptors - Rotation

Define points as a complex vector

Object is a box $k := 0 \dots 4$

$z_0 := 0 + 0j$ $z_1 := .5 + 0j$ $z_2 := -.5 + .5j$ $z_3 := 0 + .5j$ $z_4 := 0 + 0j$

$z_5 := 0 + 0j$ $z_6 := 0 + 0j$ $z_7 := 0 + 0j$



$f := \text{cfft}(z)$

```

0.354 + 0.3541i
-0.177 + 0.1771i
-0.073 + 0.0731i
-0.177 + 0.1771i
0.354 - 0.3541i
0.427 - 0.4271i

```

f

$f := f \cdot 6 \cdot \exp(.39j)$

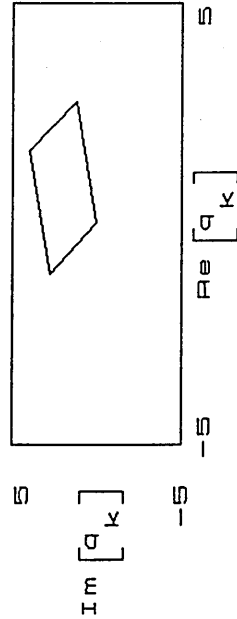
$q := \text{icfft}(f)$

```

2.775 + 1.1411i
1.634 + 3.9151i
-1.141 + 2.7751i
-2.298 \cdot 10^{-12} - 1.983 \cdot 10^{-12}i
-1.046 \cdot 10^{-12} - 2.505 \cdot 10^{-12}i
9.369 \cdot 10^{-13} - 4.804 \cdot 10^{-12}i

```

q



ADDR - Vector add real

ADDC - Vector add complex

Both of these have a CN field which, when set allow a constant to be used. The constants address is specified in the MBA field.

Scaling involves array multiplication by a constant. The appropriate instruction is :-

MLTC - Vector multiply complex

Rotation can use the DEMO instruction used in section 6.0 during Fast Cosine Transform operations. The difference in this case would be that the RBA field would indicate the starting (and only) angle, whilst the RDA would specify a zero incremental angle. In this way the same angle from the internal look up table would multiply each of the array elements of boundary points.

Further details of VSP instructions can be found in Appendix C.

References

[1] Boyle, R.D., Thomas, R.C., "Computer Vision", Blackwell Scientific, 1988.

[2] Roberts, L.G., "Machine perception of three dimensional solids", Optical and electro-optical information processing, Tippett (Ed), MIT Press, 1965

[3] Ballard, D.H., Brown, C.M., "Computer vision", Prentice Hall, 1982.

[4] Freeman, H., "Computer processing of line

drawings", Comput. Surveys, vol6, pp57-97, 1974.

[5] Gonzalez, R.C., Wintz, P., "Digital Image Processing", Addison Wesley, 1987.

[6] Persoon, E., Fu, K.S., "Shape discrimination using Fourier descriptors", IEEE Trans. Systems Man Cyb, vol SMC-7 No.2, pp170-179, 1977.

9.0 An alternative classifier - The Neural Network

9.1 Introduction

This section is not an exhaustive review of neural net theory or performance but is an investigation of how the VSP (Vector Signal Processor) may find a role in substitution or simulation of neural net actions. Part 9.2 is intended to identify the basic parameters and principles, whilst 9.3 onward is a speculative approach to the application of the VSP.

9.2 Basic concepts

Clarkson^[1] provides the historical perspective of the origin of the Neural network and its relationship to neurology.

The characteristic feature of Neural nets is that they are composed of many non-linear elements operating in parallel. It is recognised that human equivalent performance in pattern recognition will require enormous distributed computing power and Neural nets offer one technique by which the required capacity may be achieved.

9.2.1 Structure and components

The nets are made up of computational elements or nodes having several inputs and a single output. Each input carries a weighting factor and the node produces a summation of the weighted inputs. The node then passes the result through a non-linearity involving an internal threshold. The arrangement is illustrated by figure 9.1.

A large variety of interconnection schemes have been devised

Figure 9.1

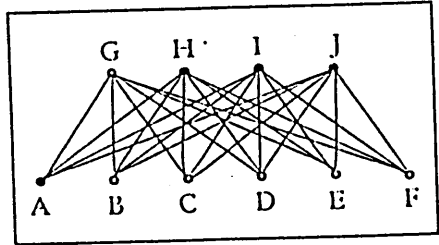
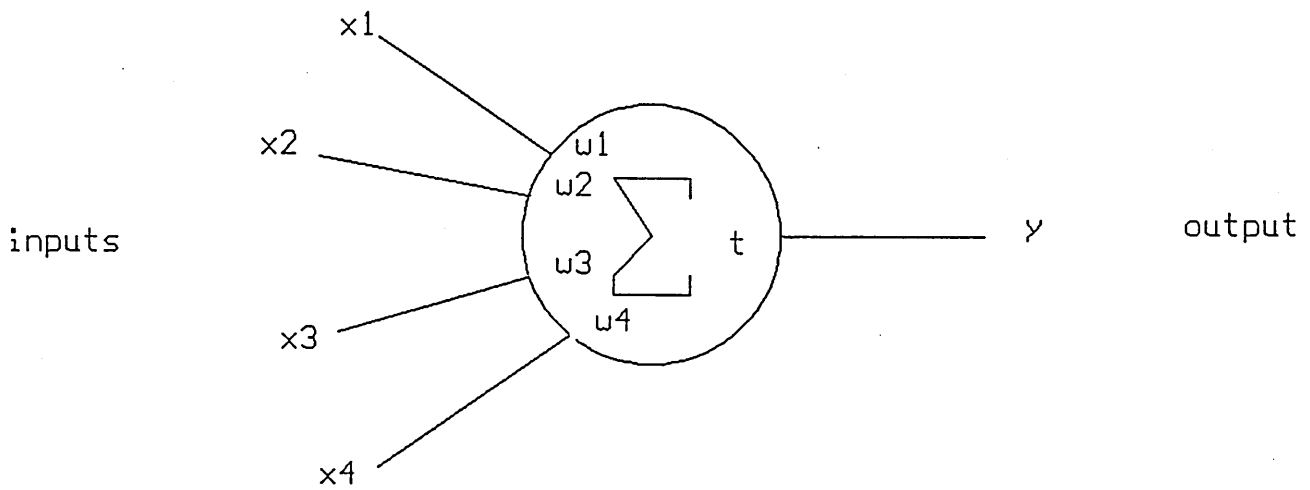
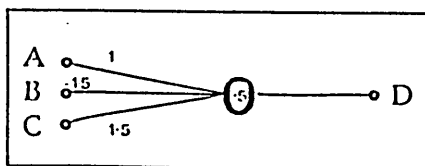


Figure 9.2



Inputs			Sum	Output Logic
A	B	C		D
0	0	0	0	0
1	0	0	1	1
1	1	0	-0.5	0
1	1	1	1	1
0	0	1	1.5	1
1	0	1	2.5	1
0	1	1	0	0

Figure 9.3

but for the purposes of description figure 9.2 shows a simple two layer network. Figure 9.3 provides an example set of weights and a node with a threshold of 0.5, and a truth table of outputs.

The "intelligence" of the network is in the combined interaction of the weighting factors and the summing values.

9.2.2 Operation

Lippman^[2] provides a taxonomy of six neural network structures that can be used as classifiers. What was needed in the context of this research is a net appropriate to the case when exact binary representations are possible, as with black and white pixels that have suffered from noise or other random distortion as described in Section 5.0. For these reasons the Hopfield^[3] type net was chosen as being most appropriate.

Lippman^[2] also provides a description of general neural net classifier action.

"In the simplest type of net the inputs are provided in parallel. The first stage computes matching scores and outputs these in parallel over n lines. These are input to the second stage which selects and enhances the strongest output. After classification is complete only one output will show a significant value. In this design each class must have a unique output allocated to it. "

Of course, more sophistication is possible but it is clear that the weights will be a unique set for each class. For use as a classifier there are immediate similarities to the operation of the matched filter classifier. The weights in the neural net case are analogous to the configuration data (reverse impulse

response) applied to the matched filter.

Obviously there will be a need to arrive at the weights for each pattern to be recognised. The process of arriving at these is termed "training" the net.

The general training method known as the "backwards propagation" strategy is described by Clarkson^[1] as follows :-

" Initially, when a network is being trained all the weighting factors are set to random values. A particular weighting factor is selected and for a given set of input data the actual output values are compared to what they are specified to be for that set of inputs. The value of the weighting factor is then adjusted to yield minimum error between actual and specified output. The process is then repeated for each weighting factor. Usually the networks stabilise at a satisfactory solution."

Operation then proceeds as follows. The net is initialised with the weights of the exemplar classes. The candidate pattern is applied and the output is examined for class identity.

One of the more sophisticated options is the adaptive neural network. In this case the classifier outputs are fed back to adapt the weights. This is essentially the importation of the learning or training process into the classifier. The result of this strategy is that differences between the exemplars and the actual applied data are adjusted for so that adaption will make a correct response more likely for succeeding input patterns that are similar to the current pattern.

9.2.3 The Hopfield Net

The variant used in this research has N nodes containing hard limiting non-linearities and binary inputs and outputs taking on the values $-1, 1$. The output of each node is fed back to all other nodes with a weighting factor denoted t_{ij} , where the outputs are never fed back to the same node. The arrangement is shown in figure 9.4.

First, weights are set using a recipe from exemplar patterns from all classes. A candidate pattern is then forced onto the output lines. The net then iterates or converges toward an unchanging state. On reaching this state the output should be the exemplar that the input pattern most resembles. There is a link between this style of operation and associative memories. The principles of associative memory are discussed by Willis [4].

This study is not an exhaustive review of neural net theory or performance but is an investigation of how the Vector Signal Processor may find a role in substitution or simulation of neural net actions. The foregoing background information is intended to identify the basic parameters and principles.

9.3 Net simulation with the VSP

Traditionally neural nets have been modelled in computer memory and first thoughts considered using the VSP in a similar way but using the potentially very fast array operations for enhancement.

The array approach means using three arrays for each net node. An input array, a weighting array, and an output array.

Figure 9.5 shows the arrangement. The weighting and

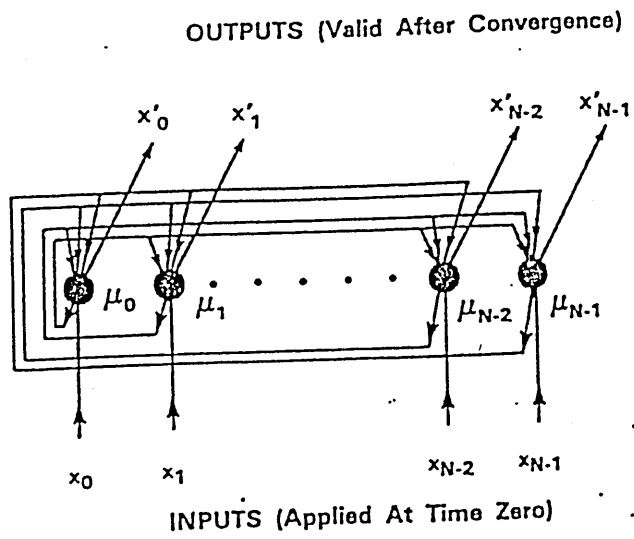
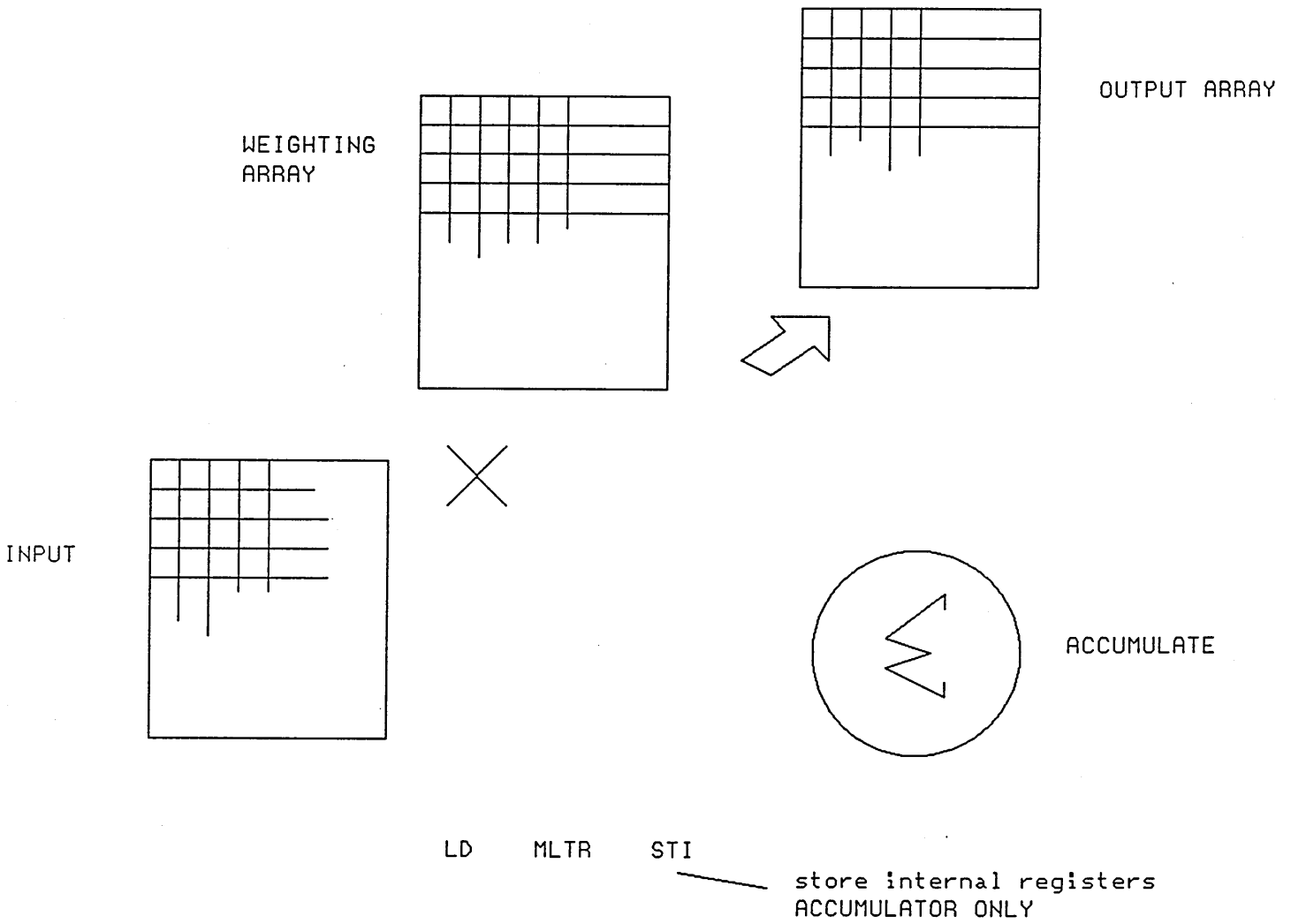


Figure 9.4

Figure 9.5



summation functionality is provided by array multiply and accumulate features leading to a sequence :-

LD - Load input array
MULTR - Multiply real by weighting array
STI - Store internal register (accumulator)

The last instruction performs the summation storage into the output array. One real problem is the non-linearity. No VSP action could be devised to satisfy the requirement and the thresholding or hard limiting tests would have to be performed by the host processor.

Another daunting feature is the number of arrays that would have to be worked with. A typical Hopfield net for eight exemplars might involve 120 nodes, implying at least 360 arrays.

9.3.1 Matched filter models

This option is proposed by Selviah, Midwinter et al ^[5]. A new modelling formalism is proposed for the purpose of allowing a more intuitive grasp of the internal actions of neural nets than has been hitherto available.

The neural network is represented by an equivalent network of matched filters which may be analysed by standard correlation techniques.

9.3.2 Matched Filter Equivalence

A correlating matched filter implementation replaces the weighted interconnection net with a network of matched filters. The general arrangement is shown in figure 9.6.

The input filter array has each filter having an impulse response of the signal complex conjugate associated with one of

the patterns being searched for.

The output of this set of filters is a series of correlation functions with a variety of amplitudes for the main correlation spike. The amplitudes are measures of the similarity between the offered signal and the memorised (as time reversed impulse responses) codes.

Only the central spike of the correlation function is useful, as it will be used as a multiplier. The side lobes and the noise will need to be removed by what Selviah et al call a "noise gate".

The noise gate output of a weighted Dirac function for each channel is applied to another array of matched filters. This time the matched filters have an impulse response that is a NON reversed version of the memorised signals. The result is that from this second array will emerge the range of memorised signals, but that which has the greatest similarity to that applied will be largest in amplitude. In short, the output of the first array is used to scale the output of the second.

Selviah et al provide proof that the action up to this point is mathematically identical to the vector matrix multiplication neural net. This proof is reproduced as figure 9.7.

Remembering that the binary input to neural nets is expected to be bipolar e.g. +1,-1, and therefore the memorised signals are expressed in this form, the action of the summation and thresholding in figure 9.6 are evident.

The output codes of the second matched filter set are summed. When several bipolar binary codes are added, if all

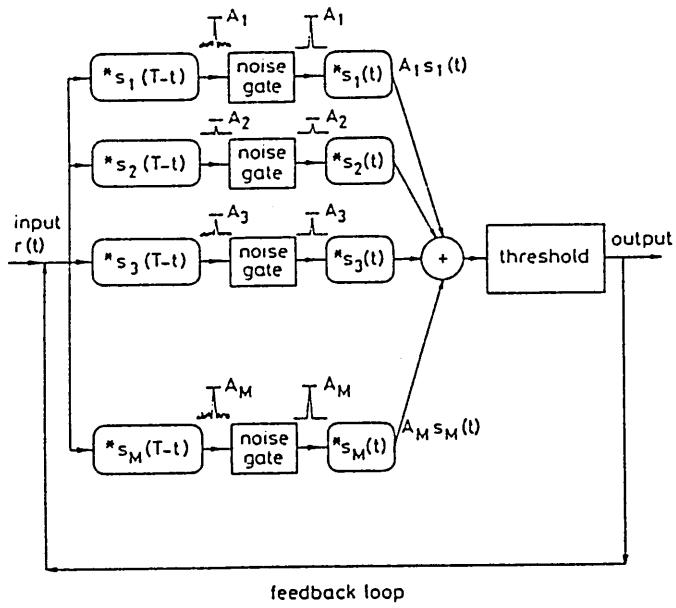


Figure 9.6

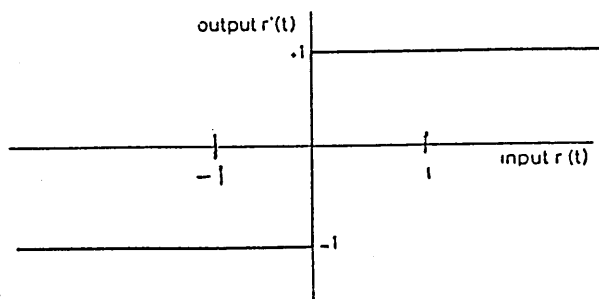


Figure 9.7

the kth bits are of the same polarity then the sum code will have the largest amplitude. Differing kth bits will have a reduced sum amplitude.

The action of a hard threshold, response shown in figure 9.7, will be to suppress the similarities and enhance the differences. This is because large outputs from the summation unit are clamped to a maximum of 1 or -1 whilst outputs that are less than 1,-1 are amplified up to 1,-1.

The overall effect of the summation and hard threshold is to act as a feature extractor.

Adding a feedback path allows iteration of the process. This feedback creates a system which iterates toward the strongest memorised code. There are problems when a number of correlations produce strong outputs. This may cause convergence to an incorrect or spurious code. Selviah et al^[5] explain the reason for this as being the strength of the non-linearity and show that a threshold function having a lower rate of change reduces incorrect convergence at the expense of speed.

9.4 Implementation with the Vector Signal Processor

It has been demonstrated in previous sections that the Vector Signal Processor can be programmed to perform as a non recursive Finite Impulse Response filter. The development of the design proceeding via the impulse response to the recurrence formula has again been demonstrated with the use of the FIR instruction.

By these means it is possible to have the input signal stored as an array, and each of the memorised impulse responses

stored as arrays. The Matched Filter program module may be called for each filter action, either loading the impulse response backward (LDB), for the time reversed cases or forward (LD) for the non reversed second array of filters.

The filter outputs can be stored as a signal array set.

The noise gate and side lobe removal is achieved by zero filling the time domain filter output arrays and is a simple array store action.

The summation is catered for by a series of array add instructions.

The only difficulty is the threshold action which involves a test of individual array elements and replacement. This is not a feature provided for within the VSP and would have to be undertaken by the host processor.

References

- [1] Clarkson, D., "Neural networks", Electronic Technology Vol 23 pp150-152, September 1989.
- [2] Lippman, R.P., "An Introduction to Computing with Neural Nets" IEEE ASSP Magazine pp4-22, April 1987.
- [3] Hopfield, J.J., "Neural Networks and Physical Systems with emergent collective computational abilities", Proc Nat Acad. Sci, U.S.A. Vol 79, pp 2554-2558, April 1982.
- [4] Willis, N., "Computer Architecture and Communications", Paradigm, 1986.
- [5] Selviah D.R., Midwinter J.E., Rivers A.W., Lung K.W., "Correlating Matched Filter model for analysis and

optimisation of Neural Networks", IEE Proceedings Pt F, Vol 136,
No.3,pp143-148,June 1989.

10.0 Error Control Coding

10.1 Underlying principles and issues

Error control coding theory has been built up over a thirty year period and has been applied to telecommunications and to applications involving computers. This study is concerned only with ideas appropriate to computing. The uses to which error control coding has been put in this field include designs for memory which is error correcting (within certain bounds), arithmetic processors, and data communication.

The subjects of data compression and error control coding are interesting when studied together because of the opposite effects they have on data size.

[1]
Lynch points out that the function of data compression is to remove natural redundancy, whilst error control coding makes specific use of artificial redundancy.

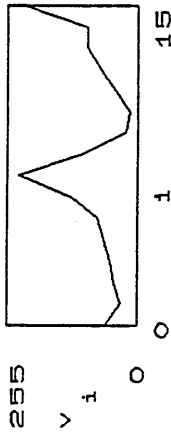
It is legitimate to question the validity of an exercise which on the one hand compresses data, and on the other expands it. Supportive arguments may suggest that many applications rely on significantly error free action and if that is the main purpose then compression is a secondary affair which mollifies to some extent the negative compression caused.

Another argument might be advanced in the case of an error coding scheme which only partly offsets the gains of compression, especially where errors would cause exaggerated distortion.

The use of error control coding in a given case would need to be the subject of careful evaluation in terms of "cost-benefit" analysis.

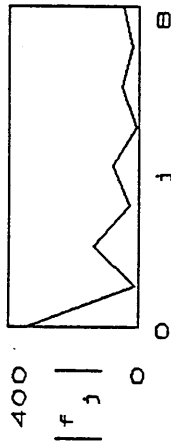

```
v :=
  65
  34
  44
  55
  66
  77
  128
  233
  112
  23
  15
  43
  71
  99
  100
  222
```

```
i := 0 .. 15
```



```
f := fft(v)
j := 0 .. 8
```

```
f_j
  346.75
 -8.32 + 8.6211i
 52.073 - 126.3961i
 -19.78 + 17.3641i
  6.75 - 80i
 -4.074 - 5.2881i
 -32.073 - 41.8961i
 -14.826 - 9.0311i
 -45.25
```



```
f_0 := 346
f_4 := 7.75 - 80i
h := ifft(f)
```

```
h_i
  65.313
 33.813
 43.312
 54.812
 66.312
 76.813
 127.313
 232.813
 112.312
 22.812
 14.312
 42.812
 71.313
 98.813
 99.313
 221.813
```

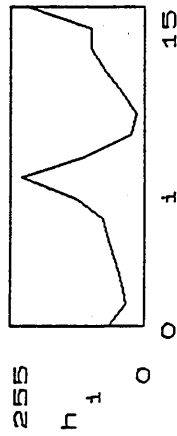


Figure 10.1

It should also be borne in mind that some data compression techniques have a "built in" error averaging action. Prominent among these techniques is Transform coding.

The effect is clearly demonstrated in figure 10.1. A vector $v[16]$ is transformed by FFT into the frequency domain, and denoted $f[16]$. Elements f_0 and f_4 are corrupted and after the inverse transform vector $h[16]$ displays evidence of the distributed or averaged error.

A similar effect can be found in the use of Delta modulation and it may be that the system has a tolerance for such errors e.g. photographic data, making error control coding less crucial.

There are many other techniques which are seriously prone to errors. A striking example is Two-dimensional Relative Address Designate coding commonly used in high compression FAX.

The nature of 2-D READ coding is discussed by Fuchs^[2] who explains that the algorithm codes a scan line by relating to the black/white transitions in the previous line. An error can propagate through an entire document but the potential compression is quoted as 50:1, with results showing a range between 7:1 and that figure.

Bodson, Urban et al^[3] also identify the deficiency and suggest that every n-lines the 2-D algorithm is suspended and a 1-D encodement such as Huffman is used. In this way an error "striation" through the document is stopped.

Clearly, this scheme is a candidate for error control coding.

10.2 Appropriate Error Control Methods

Included in the generic term "error control" are both error detecting and error correcting schemes. Simple parity methods tend to be error detecting, but to a poor level of efficiency and will not be considered further.

The use of block codes, i.e. where the data is segmented into blocks is relevant but the error correction capability inherent to parity block check characters will also not be considered because of the limitations imposed by being able to consider only character orientated data.

To be appropriate to the hardware upon which this research is rooted schemes must be amenable to the use of arrays, shift registers and other signal processing tools. The Vector Signal Processor is optimised for arrays and any algebraic requirements can be met either by the VSP or by the host processor.

The choice is restricted to Linear Block Codes, preferably cyclic. After a review of error trapping, majority logic, Hamming and Golay coding/decoding, these options were rejected. The selected method is Bose, Chaudery, Hocquenghem (BCH) [4][5].

BCH codes appear to be the most extensive and powerful option commensurate with the conditions defined above.

To correct t errors for a given artificial added redundancy, m ($t < 2^{m-1}$), then n , the overall block size, $= 2^m - 1$ and $n - k < mt$. Where k is the size of the information field of the block, as examples consider a code where $n, k = 31, 11$. With an overall block size of 31 and an information field of 11 there are 20 parity digits (mt). $2^m = 32$ therefore $m = 5$ and since $mt = 20$, $t = 4$.

A 31,11 code can correct 4 errors. Consider a code 63,45. In this case $2^m = 64$ and thus $m=6$. With $mt=18$, in integer terms $t=3$. A 63,45 code corrects 3 errors.

BCH codes are decoded by using the technique of representing code digits as coefficients of a polynomial and carrying out algebraic operations on those polynomials. As a result, an error location polynomial can be developed which pinpoints the locations of the errors. BCH is a binary system and a sub-class involving n-ary operations is referred to as Reed-Solomon code.

The details included have been derived from Bose and Ray-Chaudery [4][5], Shu Lin [6] and Rao and Fujiwara [7].

10.3 Essential Background

It will be useful to describe terms and concepts generally applicable to block codes.

10.3.1 Galois Fields

In many instances applications of algebraic thinking are required to be implemented by digital computer. In this context it is important that, whatever is done, the rules of ordinary arithmetic must apply in order that algebraic techniques can be used.

A Galois field defines the constraints in terms of number symbols and operations within which this is true. For binary work the alphabet 0 and 1 are defined along with the basic arithmetic operations of modulo 2 addition and multiplication. This set or field of elements is usually denoted $GF(2)$.

Galois fields must have 2^n symbols, where n is a positive integer. An arithmetic with two symbols can be developed into

one involving 2^n and this is done by Shu Lin [6] eventually arriving at a definition of a "primitive" polynomial.

10.3.2 Cyclic Codes

Cyclic codes are those where if an n-tuple

$$v = v_0, v_1, v_2, \dots, v_{n-1}$$

is a code vector, then the n-tuple

$$v = v_{n-i}, v_{n-i+1}, \dots, v_0, v_{n-i-1}$$

obtained by end around shift (one place right) is also a code vector.

10.3.3 Polynomial representations

It is possible to treat each code vector as having a one to one correspondence to a polynomial of degree n-1 or less.

Shu Lin [6] shows that with polynomial representation it is possible to develop some important properties for a cyclic code which make implementation of encoding and syndrome generation easier. Also provided are a number of theorems and proofs.

Included is :-

In an n,k cyclic code there exists only one polynomial g(X) of degree n-k.

$$g(X) = 1 + g_1 X + g_2 X^2 + \dots + g_{n-k-1} X^{n-k-1} + g_{n-k} X^{n-k}$$

Every code polynomial v(X) is a multiple of g(X) and every polynomial of degree n-1 or less which is a multiple of g(X) must be a code polynomial.

10.3.4 Generator Matrix

To generate each code word then a generator matrix G could be used instead of a look up table. The code word is arrived at by matrix multiplication e.g.

$$[v]=[m][G]$$

where v is an $n \times 1$ column matrix and m is a $1 \times k$ matrix, thus G is an $n \times k$ matrix. This is best illustrated by example taken from Shu Lin^[6], who is concerned with linear block codes that are systematic. A systematic code is one where the codeword has two concatenated parts. The first part is the message word and the second part is a group of redundant digits which are a parity function of the message bits. Assume a message word 101. The redundant bits, say three, might be defined as follows :-

- bit 1 even parity bit for 1st and 3rd message bits
- bit 2 even parity bit for 1st and 2nd message bits
- bit 3 even parity bit for 2nd and 3rd message bits

This yields a redundant portion :- 011 and therefore a complete code word of :- 101011

The code contains 3 message bits within a total of six and would be described as a 6,3 code.

Consider an encoder segmenting messages into 3 bit blocks, it then transforms each block into a code vector of 6 digits.

Message	Code
000	000000
001	001101
010	010011
011	011110
100	100110
101	101011
110	110101
111	111000

Because $k=3$ there are $2^3=8$ possible distinct messages. Each is transformed into a unique codeword. The list of 8 6-bit words is a subspace of the total vector space of all 6-tuples. It is possible to find a set of linearly independent 6-tuples such that each 6-tuple is a linear combination of others. The whole code can then be described by a matrix and a codeword can be found by a linear combination of the rows of this "Generator" matrix.

Using the 6,3 code example above a generator matrix for it is:-

$$G = \begin{matrix} v_1 & 100110 \\ v_2 & 010011 \\ v_3 & 001101 \end{matrix}$$

Element v_1 corresponds to the message word 100, v_2 corresponds to message word 010 and v_3 corresponds to message word 001. All other codewords are found by modulo-2 addition of these.

Thus if the message is 101 then the codeword is :-

$$1.v_1 + 0.v_2 + 1.v_3 \\ = 1.(100110) + 0.(010011) + 1.(001101) = 101011 \quad (\text{modulo-2})$$

Clearly the first 3 digits of the code word are the message bits, and the last $n-k$ bits are linear functions of the message bits. These $n-k$ bits are the parity check bits and the code is systematic. Instead of storing the full 8 codeword options at the encoder only the 3X1 generator matrix is necessary in order to generate the full set. The code is specified by the generator matrix.

10.3.5 Parity Check Matrix

The transpose of the generator matrix is the parity check matrix. e.g. for the generator matrix above the parity check matrix H is :-

101100
110010
011001

This is obtained by the transposition being specified as below:-

$$\begin{array}{l}
 V_1 = \begin{array}{cccccc} & & \text{bit} & & & \\ & 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 0 & 1 & 1 & 0 & \\ \end{array} \\
 V_2 = \begin{array}{cccccc} 0 & 1 & 0 & 0 & 1 & 1 \\ \end{array} \\
 V_3 = \begin{array}{cccccc} 0 & 0 & 1 & 1 & 0 & 1 \\ \end{array}
 \end{array}$$

$$\begin{array}{l}
 V'_{1n} = V_3(5-n) \quad n=0 \dots 5 \\
 V'_{3n} = V_1(5-n) \\
 V'_{2n} = V_2(5-n)
 \end{array}$$

yielding

$$H = \begin{array}{l}
 V'_1 = 101100 \\
 V'_2 = 110010 \\
 V'_3 = 011001
 \end{array}$$

The usefulness of the parity check matrix is to allow the derivation of the syndrome.

10.3.6 The Syndrome

Consider the case of a message word 111 from the generator matrix the code word is 111000. If the transpose of H i.e. H^t is derived by converting rows into columns it is :-

110
011
101
100
010
001

If the code word 111000 is denoted u then a matrix multiplication can be carried out as follows :-

$$s = u.H^t$$

$= 110 + 011 + 101 = 000$ (modulo 2). s is called the "syndrome" and will be 0 for all genuine codewords, but not if corruption has taken place. For example consider the case where codeword 100110 had been corrupted into 101110.

The syndrome will be :-

$$s = 110 + 101 + 100 + 010 = 101 \text{ and is non-zero.}$$

Having detected the errors it is necessary to identify them in order to correct. One way of doing this is to use the standard array method which allocates a one to one correspondence between non-zero syndromes and the correctable set of error patterns.

Each formulation of this type of code has a different error correcting capability. The 6,3 example is only capable of correcting single bit errors. Using the notion that a correct codeword can be changed into a corrupted codeword by the modulo-2 addition of an error vector then the error vectors that can be corrected by the 6,3 code above are :-

000001
000010
000100
001000
010000
100000

These are the "correctable set" of error vectors, otherwise

known as the coset leaders. Six non-zero syndromes could be associated with these coset leaders and simple interpretation will allow error correction for these error vectors.

If a corruption occurs for which a coset leader is not responsible then an incorrect decode will result. Systems should be devised so that the coset leaders are the most likely errors to occur.

The details above are general and apply to any n,k linear block systematic code. The term "distance" is used to define the number of components by which two code words differ. The "minimum distance" for a code is the smallest distance between all possible pairs of codewords. The notion of minimum distance is important in that it determines the error correcting capability of a linear code. Greater than 1-bit error correction is possible but an n,k of more than 6,3 is needed. For instance a 15,5 code corrects up to 3 error bits.

10.4 Bose-Chaudery-Hocquenghem Codes

The code is generated by means of a generator polynomial. This is derived by letting "a" be a primitive element of Galois field (2^m) . A primitive element is any element whose powers generate all the non-zero elements of $GF(2^m)$. The least common multiple of the minimum polynomials of a^i .

The form of the generator polynomial for $GF(2^4)$

is as follows :-

$$g(X) = 1 + X^4 + X^6 + X^7 + X^8$$

yielding a 15,7 cyclic code able to error correct 2 bits. Other

possibilities can be created by using more or fewer of the minimum polynomials. Thus using the three minimum polynomials generates a 15,5 code that corrects 3 bits, whereas using the single minimum polynomial is single error correcting.

The decoding of BCH codes assumes that the code vector has been added to by a noise vector. The first step of the decoding process is to calculate the syndrome which is a vector with $2t$ components, using the received vector. The set of equations are solved for each of the vector elements and then a technique is applied which finds an error location polynomial. Finally, the roots of this last polynomial are found which identifies the error positions within the received vector. Details are available in Rao and Fujikawa [7].

10.5 BCH Implementation

Implementation of binary cyclic codes may be achieved by a shift register involving modulo 2 addition between delay elements, additional control logic and switching. The examples given by Rao and Fujikawa [7] together with Shu Lin [6] suggest architectures reminiscent of those used for programmable matched filters earlier in this study.

This allows consideration of the use of programmable Digital Signal Processors or even the imaginative use of the FIR instructions in some versions of the VSP. In addition a purely hardware solution could be developed, but would be relatively inflexible.

One problem remains for the BCH option and the use of the

DSP or VSP processors. Neither are structured for the finding of roots of polynomials, even using the iterative methods suggested by Berlekamp [8], and since this is a vital decoding function, the host processor would have to be used, ideally with an arithmetic co-processor.

10.6 Error Detection - CRC

As has been pointed out, and quantified above, very large overheads are incurred in error correcting systems. These overheads are so large that considerable data compression is needed to offset them. In many cases the necessary compression is not available, and bandwidth is limited.

In some of these situations it may be that the detection of errors may be enough and correction be made by retry or retransmission, especially in data communications.

To satisfy the need for a reliable error detection one option is the CRC algorithm. Computationally it is relatively trivial, and the overhead can be very small, perhaps 16 bits added to a information field of 1024 bits. It is suitable in cases of minimal compression availability.

A CRC generator is based on a PRBS (pseudo random binary sequence). A shift register with feedback is used to operate on the data within a block. A shift register may be defined using an operator D defined such that :-

$$X(t) = DX(t-1)$$

Multiplying by D is equivalent to delay by 1 bit. The feedback equation may be described by :-

$$D^4 X(t) + DX(t) + X(t) \text{ or simply } X^4 + X + 1$$

or similar. Thus the system is able to be represented as a polynomial.

Frohwerk^[9] correctly points out that feeding a data block into a PRBS generator of this kind is equivalent to dividing the data stream by the characteristic polynomial of the generator.

At the end of the process a residue or remainder is left in the shift register. The remainder is shown by Frohwerk^[9] to be characteristic of the data stream and is defined by him as a "signature".

Gordon and Nadig^[10] prove that if when data blocks are created they are tagged with this signature and if on recovery the same algorithm is used, then the probability of error detection is given by :-

$$\text{Prob}(\text{PRBS fail}) = 2^{-(m-n)} \cdot 2^{-m}$$

where m is the message length and n is the number of bits in the shift register.

A 16 bit register dealing with 256 bit blocks of data would give a system where the probability of failure to detect any error is $1.526 \cdot 10^{-5}$, and proof is also provided that single bit errors are always detected.

Arguments regarding implementation are the same as for BCH coding.

References

- [1] Lynch, T.J., "Data compression-techniques and applications", Van Nostrand Rheinhold, 1985.

- [2] Fuchs, P.M., "Compressing data in bit mapped displays", EDN, pp173-183, October 1986.
- [3] Bodson, D., Urban, S.J., Deutermann, A.R., Clarke, C.E., "Measurement of data compression in advanced group 4 facsimile systems", Proc. IEEE, vol73, No.4, pp731-739, April 1985.
- [4] Bose, R.C., Ray-Chaudery, D.K., "On a class of error correcting binary group codes", Inf. and Control, 3, pp68-79, 1960.
- [5] Bose, R.C., Ray-Chaudery, D.K., "Further results on error correcting binary group codes", Inf. and Control, 3, pp279-290, 1960.
- [6] Shu-Lin, "An introduction to error correcting codes", Prentice Hall, 1970.
- [7] Rao, T.R.N., Fujiwara, E., "Error Control coding for computer systems", Prentice Hall, 1989.
- [8] Berlekamp, E.R., "Algebraic coding theory", McGraw Hill, 1968.
- [9] Frohwerke, R.A., "Signature analysis a new digital field service method", Hewlett Packard application note 222-2, pp9-15, October 1980.
- [10] Gordon, G., Nadig, H., "Hexadecimal signatures", ibid pp1-8.

11.0 Conclusions

11.1 Scope

The conclusions derived are in two categories, firstly there are those associated with the relevance, currency and validity of the research and secondly the summary of the value and results of the work to test the original hypotheses.

11.2 Relevance, currency and validity

The main thrusts of this work have been :-

- 1) Matched filters and Correlation techniques
- 2) Transform coding
- 3) Quadtree derived ideas

Certain authors, such as Marshall^[1] advocate that these methods are inelegant because it is often the case that analysis and coding techniques require different principles and algorithms. This is particularly true of transform coding, but true or not technological innovation is only fruitful where it is possible to proceed to a viable product.

Recently, large resources have been allocated to solving the computational problems of transform coding, especially the DCT. VLSI has been developed to the point where signal processing algorithms are implemented with an efficiency that allows for real time application. Many useful products are strongly reliant on real time operations.

Systems that are universally used must comply to standards and the subject of low bit rate coding is no exception. Specification of a technique within a standard is a vote of

confidence that it represents, if not the most elegant solution, then the best, commensurate with feasible realisation.

One of the main drives for data compression has naturally come from the telecommunications industry, with particular reference to the services expected to have to be provided under the ISDN (Integrated Services Digital Network). These services are likely to include videophone teleconferencing, and photographic videotex.

Another prime mover for this technology is the coding of moving images on digital storage media. In both these areas standards are at the point of publication, as shown in Shah ^[2].

Of considerable importance is CCITT recommendation H261, which is aimed at the telecommunications objectives above. H261 segments the video problem into motion compensation and transform coding, specifying the DCT in the latter case. A review of the situation has been carried out by Yates and Ivey ^[3].

The digital storage media position is that ISO (International Standards Organisation) is presently considering the CD-I (Compact disc interactive) standard which is based upon the DCT, again described by Shah ^[2].

It is concluded that the study of transform coding is a current and valid area of interest, which retains some interesting problems in implementation.

The quadtree approach was stated in Section 7.0 as being a concept often returned to. An example of this is found in Ireton and Xydeas ^[4], which contains implicit thinking developed by Tanimoto ^[5] in an earlier decade.

This fact prompts the thought that during this research

numerous instances of "circular thinking" have been noted. [5]

Another case in point is where Selviah, Midwinter et al suggest that a more intuitive grasp of the action of neural networks may be obtained via a matched filter model, as described in Section 9.0. The circularity here concerns only classifiers.

The justification for the inclusion of quadtree methods is in the need for comprehensive cover, a wide understanding, and the ability to easily accomodate radical thinking such as proposed by Marshall [1].

Ideas of signal processing are an inherent part of image or data compression and since correlation is never far from mind in signal processing, it is held to be relevent to have studied applications of auto and cross correlation.

The conclusion that the areas studied are relevent and valid indicates that the underlying philosophy for this research was sound.

The research did allow a conclusion to be reached concerning the hypothesis that "devices optimised for signal processing brings benefit to real time" data compression, but it is additionally believed that in the course of study small contributions have been made to technique.

These are :-

- 1) Developing the VSP implementations
- 2) The adaptive use of Morse code principles in variable length transfer coding.
- 3) The coding developed using quadtrees and delta

modulation.

In experiments both 2) and 3) achieved close parity with the performance of existing schemes despite the fact that neither were further developed.

The subject of data compression has been reviewed and topics showing the greatest potential for the application of array and signal processors were looked at in detail. Interactions between data compression techniques and image processing were examined and implementations explored for appropriate cases. The use of a Vector Signal Processor has been demonstrated. The groundwork for further research has been laid and is expected to be of immediate use in a Medical electronics imaging project and a trans-national project for maritime data communications.

11.3 Signal processing devices and data compression

The results of experiments in pattern matching and substitution, transform coding and filtering using the Vector Signal Processor as a co-processor reveal that it is significantly useful, but has certain deficiencies.

The hope was that very fast real time processing could be achieved but performance was limited by the fact that most techniques involve either array processing together with bit orientated operations or secondary encoding involved methods for which the VSP is not optimised.

The consequence was that the host processor had to be used for proportionately more tasks than was hoped. Notwithstanding this the VSP produces a significant increase in performance than is available from the host type general purpose processor.

Certain functionality is best supplied by means of dedicated hardware. Included in this category is error control coding.

It is concluded that the best role of the VSP is that of acting as an "accelerator" for difficult or time consuming operations such as frequency domain convolution, FFTs, DCTs, and fast finite impulse response filters flexibly implemented in software.

Experimental measurements indicate 128 point FCT was calculated in 39.7 microseconds. This is slower than the manufacturers data but this may be due to the use of an XT (Norton benchmark 1.0) host instead of an AT. With the means at my disposal it was not possible to split host and VSP action times as they were interacting and bus sharing at a high rate.

Using the CIF (common intermediate format) of CCITT H261, which has a luminance spatial resolution of 352 pels by 288 lines and a frame rate of 30 Hz calculations were made to evaluate the transform coding performance.

Experiments suggest that segmentation into 16X16 sub blocks is optimal, in terms of the trade off between host interaction and array size maximisation. It was estimated that a 16X16 DCT would take 1000 microseconds using a single VSP. H261 resolution requires 22X18 sub-blocks of 16X16. The 396 blocks require around 396 milliseconds for processing. This is an order of magnitude too slow for H261, even if sufficient compression is achieved to fall within the 64kbs⁻¹ bandwidth maximum.

What is too slow for real time video-phone is not too slow for many applications. If picture storage on backing store

media, such as CD or Winchester discs, were to be considered for desktop publishing then the VSP can provide good performance.

Calculations based on the VSP measured speeds were carried out for a FAX application using pattern matching and substitution. CCITT document No.1 resulted in the VSP needing to calculate 42 ACFs. Zoran^[6] source data expects the VSP to execute 128X128 fast convolutions in 570 milliseconds resulting in a 24 second "generating" period which will be added to by screening and statistical matching. The pre-transmission processing time is thus further stretched. Since a FAX need not be "on line" during this delay it is likely to be absorbed by concurrent processing within the scanning period expected by users.

This kind of application, where it is possible to process data before storage or transmission, is suited to the VSP at its present clock rate and performance. This enables use in electronic publishing, CD-ROM databases, medical imaging and machine vision - where the characteristics of the human eye do not force no-flicker frame rates and relatively high resolution is not involved.

The VSP is capable of multiprocessor configuration, but the performance increase is approximately proportional so that the H261 standard would require an uneconomical number of processors.

Yates and Ivey^[3] contains an analysis which shows that up to 20 parallel "single architecture" units would be required and this confirms the conclusions above.

Present performance is adequate for compression of

Table 11.1 VSP general applicability

	Symbol matching	Transform coding	Quadtrees
Text	yes [*]	n/a	n/a
Line Dwg	yes	n/a	no
Grey scale	n/a	yes ⁺	no

* for fast correlation and FIR matched filters

+ With the exception of the Hotelling transform

n/a = not applicable

Table 11.2 Applicability of techniques to VSP

	Suitable	Speed	Comment
Shannon-Fano	no	-	-
Huffman	no	-	-
Run length	no	-	-
BNO	no	-	-
Morse	no	-	-
Convolution	yes	4mS	16X16 2-D
FIR filter	yes	no data	VSP-325
FFT	yes	0.4mS	16X16 2-D
FCT	yes	0.95mS	16X16 2-D
Fourier Desc	yes	0.04mS	16,1-D,rotate
Hotelling	no	-	-

ble 11.2 contd

	Suitable [*]	Speed	Comment
Neural net	yes	20mS	120 node Hopfield net, thresholding by host. (estimate, based on 10MHz clock)
Segmentation	no	-	-
Screening ⁺	yes	0.03mS	per 16X16 character cell.
Statistical ⁺ Matching	no	-	-

+ as described in Section 5.0

* using matched filter modelling.

-1

"still" images. Any slow scan e.g. up to 5 frames second would be feasible.

11.4 Summaries

Tables 11.1 and 11.2 respectively show the applicability of the VSP to major areas of work and to individual techniques.

11.5 Other considerations

Several other aspects of a device must be considered in appraising its value. Such an aspect is the development system environment. The VSP is provided with an editor/assembler and a debugger which are basic. Interaction with the VSP is possible via the host running "C" programs utilising a library of procedures, but the style of operation involves the "C" modules downloading assembler modules and invoking them. This leads to a "to and fro" action resulting in the host processor operating in a less than optimal way. Comparing the VSP with a DSP product such as the TMS32020 shows that in pure assembly level terms that the VSP is significantly faster and that program lengths are much shorter leading to reduced development periods. It is concluded that the VSP concept is hampered in reaching its full potential by the VSP/host mechanism and by the lack of a full cross compiler type facility. These disadvantages are outweighed when the VSP is used for those tasks for which its architecture is sympathetic or optimised but reference to figures 11.1 and 11.2 reveal the limitations.

References

- [1] Marshall, S., "Contour based image coding", IEE Electronics

Colloquium digest 1990/075 , pp4/1-4/3, May 1990.

[2] Shah,Y., "State of the art in low bit rate coding", IEE Electronics Colloquium digest 1990/075, pp1/1-1/3, May 1990.

[3] Yates R.B. and Ivey P.A., "Approaches to image data compression for video coding" IEE Electronics Colloquium digest 1990/075, pp10/1-10/5, May 1990.

[4] Ireton, M.A., and Xydeas, C.S., "A Progressive coding technique for binary images", IEE Electronics Colloquium digest 1990/075, pp11/1-11/4, May 1990.

[5] Tanomot, S.L., "Image transmission with gross information first", Computer Graphics and Image Processing, 9, pp72-76, 1979.

[6] Zoran, Technical notes, TN 92040-0187, pp3-5, 1987.

Appendix A

Published Papers

Listing 1.1 Details of published papers

Copies of the following are included as Appendix "A".

(1) "Vector Signal and Digital Signal Processors in Character Recognition". Colloquium on Pattern Recognition and Applications

IEE groups C5/E4 digest No.1989/109 October 1989.

This paper explains the pattern matching and substitution which is the subject of Section 5.0

(2) "Applying the Vector Signal Processor". Colloquium on Practical applications of digital signal processing devices.

IEE groups C2/E5 digest No.1990/100 June 1990.

The paper defines the nature of Vector Signal Processors and discusses architectural and operational details together with experiences of implementing Fast Convolution/FFTs/FCTs/Fourier descriptors. The material arises out of Section 3.0

(3) "On Hardware for development of Signal Processing Hardware".

Polytechnics and Colleges Software Engineering Journal No.3 Winter 1989.

Surveys the options for platforms appropriate to real time software development. Relates to Section 3.0

(4) "Vector Signal Processors and Digital Filters in Data Compression for Electronic Publishing".

In the refereed journal "Microprocessors and Microsystems", November 1990 edition.

Expands paper 1, providing experimental results and discussing Fast Cosine transform implementation. Covers material from Sections 5.0 and 6.1.4

VECTOR SIGNAL AND DIGITAL FILTER PROCESSORS IN CHARACTER RECOGNITION

*

G.A.King and P.Picton

Introduction

The techniques discussed arise from work on Data Compression for facsimile or for document databases and processing that will be found in new generations of photocopiers. Background details of each application may be found in Johnson, Segen and Cash [1] together with Wilcox and Spitz [2].

It is assumed that suitable segmentation methods will have isolated textual characters, line drawings, and possibly photographs. In our system characters are dealt with by symbol pattern matching, whilst adaptive Huffman style or Relative Address Designate encoding is used for the line drawings. The concern of this paper is symbol pattern matching.

Pattern Matching Method

Most workers in this field have used template matching, constructing an error pel map using weighted XOR to create a match figure of merit. See Holt and Xydeas [3]. Reported problems with template matching are due to mis-registration, anomalies due to thin entities, or orientation of symbols. See Boyle and Thomas[4]. Moving a template across a symbol and checking for best fit amounts to spatial domain convolution. The position of least difference creating the lowest figure of merit and a threshold may be defined for a "match".

The deficiencies of the weighted XOR map mentioned previously prompted the investigation of alternatives. In early optical work the notion of using Matched Filters for pattern recognition was suggested by Shelton and Horowitz[5], and correspondence by Kain[6] supports the idea by proposing statistical measures of matches. Computational overheads appear to have halted this line of development at the time but advances in specialised architectures suggested that a review of these original ideas would be worthwhile.

To achieve the necessary speed for real time applications a fully configurable matched filter is required. Product "A" is a digital filter processor capable of having FIR filter coefficients downloaded to it. It would perform the matched filter function, but our implementation involves cascadable EPLDs.

When a signal representing the symbol for which the filter is configured is applied, the output will be the Autocorrelation function for that symbol. If the Autocorrelation functions (ACFs) of the symbols in a given document are known, successive configuration of the filter with coefficients of candidate

*

G.A.King is with the Informatics Division, Southampton Institute of Higher Education. Dr.P.Picton is with the Open University.

symbols will produce at the filter output correlation functions that may be compared with the original ACFs. Incorrect symbols will produce Cross Correlation functions that may be rejected because of statistical characteristics.

Creating ACFs

Vector signal processors are optimised to operate on arrays in terms of vector add, vector multiply, re-ordered vector store, with an ALU structured for fast execution of FFT butterflies. This type of processor, a Peripheral Vector Processor has a number of examples explained in a survey by Karplus[7]. Convolution and Correlation are related processes, and the well known advantage of performing convolution in the frequency domain (usually termed "fast convolution"), is obtained when performing fast correlation.

The key point is that fast convolution can be used for fast correlation if one term is stored in bit reversed order. Typically array store instructions allow for this. The process of frequency domain correlation is illustrated in fig.1 and listing 1. Proofs are available in Gonzalez and Wintz[8] and also Zoran[9]. A short business letter such as CCITT facsimile document 1 can have a library of ACFs generated in 2-4 milliseconds. The Vector Signal Processor known as product "B" can achieve frequency domain correlation of arrays of 128 points in around 500 microseconds.

Screening

To reduce the time taken to recognise a symbol a rough comparison or screening stage is used. Three or four horizontal slices are taken from the segmented character and a majority vote system identifies likely candidates if, say, three out of four slices compare within a set tolerance. For the library size used in tests around 15% of the total were selected. Each was checked using the main statistical matching technique. There is a trade off between the screening and main techniques. If the screening is rigorous the main technique need only be invoked for a very few candidates. The faster the processing available for the main technique the less demanding the screening needs to be. The VSP is well suited to the array partial comparisons required for screening. The work being undertaken suggests that a more cursory screening than methods such as the height/width/internal black run method suggested by Holt and Xydeas[3] is appropriate.

Statistical Matching

Comparisons are made between the correlation function that is output from the matched filter, and the ACFs of match candidates arising from the screening stage. The essence is to detect significant differences rather than those due to scanner noise or other random effects. Kain[6] proposed several measures of similarity, the most useful of which involves dividing the standard deviation of each entity by its mean. It was thought that this method was insufficiently consistent and an alternative was derived from Harrison[10]. Each discrete correlation function is treated as a series of samples. The standard deviation and the mean is calculated. From this the standard error which would apply to a group of similar sets is predicted. The standard error of the difference is then compared with the difference of the means. Put another way, the difference of the means is compared with the spread of the means

theoretically predicted. A resulting match figure of merit is obtained giving a simple criterion based on the normal distribution. A figure of 5.0 indicates a 1 in 1000 probability of a match, 2.5 shows a 99% confidence of mis-match. Similarly, 2.0 yields 68% mis-match level of confidence. Results achieved typically have figures of merit under 1.0 for recognition. Just what value is used to accept a match may be defined to accommodate permitted levels of distortion.

Overall Strategy

The overall system is illustrated by flowchart 1, whilst the main matching technique is covered by flowchart 2.

Conclusions

Work so far has shown that implementation is feasible and specialised architecture co-processors offer sufficiently fast operation for acceptable real time performance.

Future work will try to capitalise on the inherent advantages of nth order correlations carried out in the frequency domain in that they can remove the effects of translation and scale variation. McLaughlin and Raviv[11] describe a procedure that is consistent with the use of a Vector Signal Processor. Fourier Descriptors will be re-examined and addressed to changes of scale and rotations.

References

1. JOHNSEN, SEGEN, CASH Coding of two level pictures by pattern matching and substitution, Bell System Technical Journal vol 62 No 8 1983.
2. WILCOX, SPITZ Automatic recognition and representation of documents, Proc. Int Conf on Electronic Publishing Nice 1988, Ed. J.C. Van Vliet.
3. HOLT, XYDEAS Recent developments in image data compression for digital facsimile, ICL Technical Journal May 1986.
4. BOYLE, THOMAS Computer Vision, Blackwell Scientific Publications 1988 ISBN 0-632-01577-2.
5. HORWITZ, SHELTON Pattern recognition using Autocorrelation Proc. IRE Jan 1961.
6. KAIN Autocorrelation Pattern Recognition, Correspondence Proc. IRE June 1961 pp 1085-1086.
7. KARPLUS Architectural and software issues in the design and application of peripheral array processors, Computer vol 14 No. 9 pp 11-17.
8. GONZALEZ, WINTZ Digital image processing, Wesley 1982 ISBN 0-201-02596-5.
9. ZORAN Fast Convolution, Technical Note TN92045-0187.

10. HARRISON Statistics and reliability, Open University Press 1976 SBN 335 02509 9.

11. MCLAUGHLIN, RAVIV Nth order Autocorrelations in pattern recognition, Information and Control 12, 121-142 1968.

gak 8/89

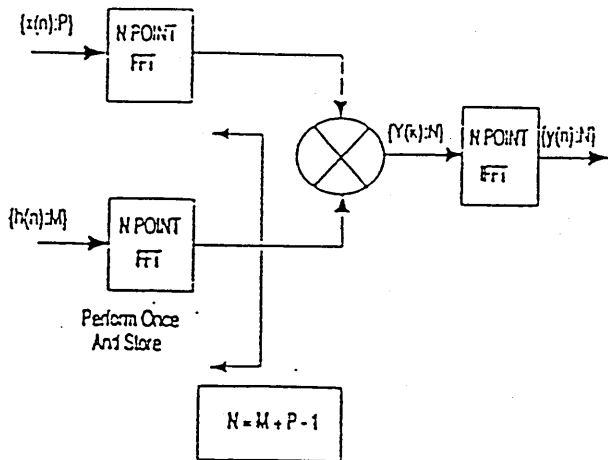


Figure 1.

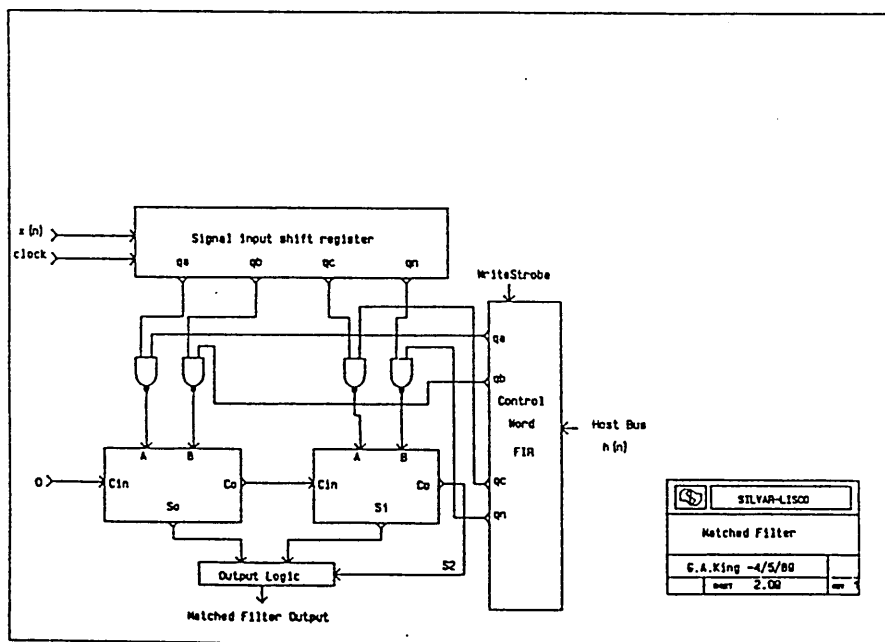
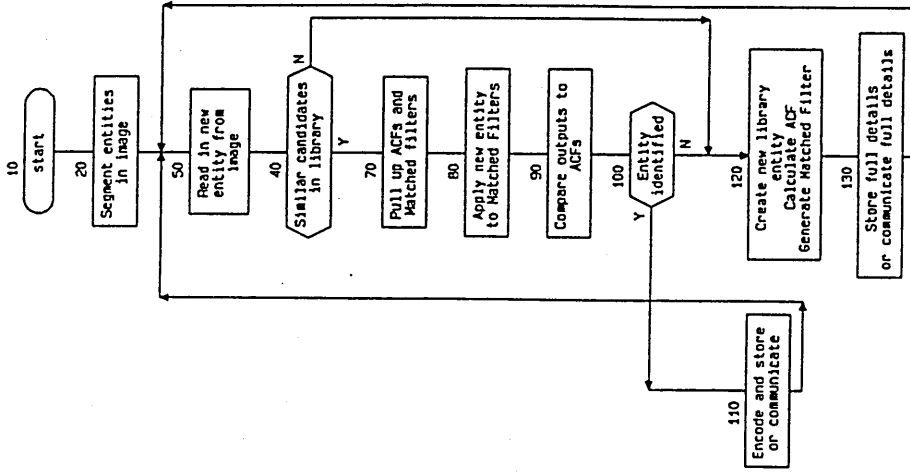


Figure 2.

1303833



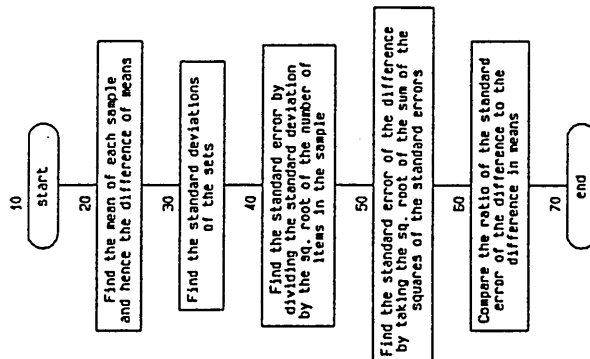
Process for Data Compression
Using Autocorrelation and
Matched Filters

G.A.King

Kindra VI.A
Research
VERSION 7
28-Sep-88 14:18

Flowchart 1.

31313



Method for testing the
significance of differences
between sample sets

G.A.King ver 1.01

Kindra VI.B
stats
VERSION 4
02-May-89 09:18

Flowchart 2.

APPLYING THE VECTOR SIGNAL PROCESSOR

G.A.King and P.D.Picton *

1. Introduction

The Vector Signal Processor (VSP) is designed for use as a co-processor and has a bus structure and timing regime suitable for operation in a manner similar to microprocessor peripherals with a DMA interface.

The architectural philosophy for the microword organisation has been approximately categorised as "horizontal" by Flynn[1] and is comprehensively described by Dasgupta[2]. The main features of this structure are :-

1. It enables different resources (e.g. functional units, data paths) in the micromachine to be controlled independently. i.e. A single microinstruction may specify concurrent operations.
2. It leads to relatively large microword lengths.
3. Programmers can exercise control of parallelism at machine operation level.

The characteristics of the VSP instruction set are :-

1. It operates at algorithm level - it is programmed at functional level.
2. Instructions are Vector orientated.
3. Instructions are slanted towards DSP problems (e.g. FFT butterflies, Magnitude Squares)

Applications and choices

We have used the VSP in experiments concerned with pattern matching and substitution, data compression of grey scale images, image processing and manipulation with digital filters and Fourier descriptors. These applications require the implementation of FFT, Fast Cosine Transforms, Fast correlation, and Real or Complex vector multiplication.

The VSP is capable of tightly coupled co-processor action, which infers shared memory and a DMA bus structure. Two running modes are possible, Master or Slave. The latter demands that the host is responsible for delivering instructions to the VSP's FIFO instruction buffer. Up to 12 instructions can be queued. The VSP FIFO and its other internal registers are memory mapped. As a Master device the VSP fetches and executes its own instructions once it has been "started" by the host.

Action is invoked by the host writing a start address into the VSP's instruction base/start register.

Alternatively, a start can be initiated by a JUMP indirect instruction being written to the instruction FIFO. The format of

* G.A.King is with the Information Systems Division, Southampton Institute of Higher Education. Dr. P.Picton is with the Open University, Faculty of Technology, Electronics Discipline.

the JUMP instruction may be :-

```
JMPI RS:0,EI:0,MBA:START_ADDRESS;
```

This presumes the host has pre-loaded START_ADDRESS. In our implementations the slave mode was rejected as too slow since it relies on the host speed. In dedicated systems the host may be a simple microcontroller. The master choice allows VSP programs to run to completion, or until an interrupt is generated by an instruction (EI:1).

The nature of VSP programs is best illustrated by example.

VSP program style

The programming effort for this processor may be compared by considering the calculation of a Magnitude Squared Spectrum of 128 real valued time samples.

```
LD NMPT:128,RS:0,MDF:2,ZR:1,MBA:0;
FFT NMBT:128,RS:0,FPS:64,LPS:1;
MGSQ NMPT:128,RS:0,ADF:2;
ST NMPT:128,RS:0,RV:1,MDF:2,MBA:256;
```

The characteristics of the horizontal architecture is evident. NMPT defines the number of points in the vector, MBA is the memory base address of the array, MDF defines the data format-2 is REAL data. For the FFT instructions NMBT is the number of butterflies to be performed whilst FPS=first pass separation and LPS=last pass separation of points for the butterflies which specifies the Cooley Tukey algorithm in this case. AS:1 selects fixed point arithmetic with automatic scaling by 0.5 at each stage. A useful feature is that the Assembler allows global or default values to be allocated to the fields.

Implementing the forward FCT

Assuming a normally ordered sample array of size N residing at memory address MBA:SUB_PICTURE, the first task is to comply with Makhoul's algorithm by loading the array into the internal memory of the VSP - even points first.

```
LD NMPT:N/2,MBS:1,MSS:2,RV:0,MBA:SUB_PICTURE+1;
```

This selects every other point beginning with the second point. Next, the odd points must be loaded in address reverse order. The only load parameter capable of reversal arranges bit reversal. This gives correct results for FFT algorithms, but not for Makhoul's FCT[3]. The remedy is to load the data, store it in reverse order and load it again leading to the sequence :-

```
LD NMPT:N/2,MBS:1,MSS:2,RV:0,MBA:SUB_PICTURE;
STB NMPT:N/2,MBS:N/2,MSS:N/2,MBAB:SCRATCH;
LD NMPT:N/2,MBS:N/2,MSS:N/2,MBA:SCRATCH;
```

It is necessary to perform the outstanding bit reversal for the load operation concerned with the Sande-Tukey algorithm. An FFT is followed by rotating the FFT by an exponential vector.

```
FFT NMBT:N,R:1,FPS:1,LPS:N/2,FSIZ:N,I:0,RS:1;
DEMO NMPT:N,ADF:3,RBA:0,RDA:496,VSIZ:32,RS:1;
ST NMPT:N,MBS:N,MSS:N,RS:0,MBA:RESULT;
```

The DEMO instruction effectively multiplies by :-
 $2^{\exp(-j(\pi)k/2N)}$ as specified in Makhouls paper.

Flow Control

There is a paucity of flow control instructions, JMPI,HLT and NOP being all that is available. Since all flow control must be achieved with these instructions, there is a need for interaction with the host. Assume a memory location SWITCH_ADD. Depending on certain tests, this address is loaded with the address of a previous VSP instruction in order to loop. Alternatively it could be loaded with the address of another routine, such as a scaling module.

The host is invoked by means of an interrupt, which, via the control bus halts further instruction fetches by the VSP. The interrupt is caused by a VSP instruction having its EI field set. The instruction FIFO pipelining principle means that if the instruction following that which caused the interrupt is a JMPI then, because the MBA field is a variable, the loading must be delayed. Assume that this is the case with SWITCH_ADD. After the interrupt, the host, as part of the interrupt service routine, makes a decision and loads SWITCH_ADD appropriately. So as to allow this, bearing in mind the FIFO pipelining, a number (4) of NOP instructions must be interposed. Consider the program fragment :-

```
STI NMPT:1,STR:5,EI:1,MBA:SCALE;
```

This stores a VSP internal register (the scale register) in a location denoted SCALE, whilst interrupting the host EI:1. The host loads SWITCH_ADD according to circumstances and subsequent lines are :-

```
NOP NMPT:1;  
NOP NMPT:1;  
NOP NMPT:1;  
NOP NMPT:1;  
JMPI MBA:SWITCH_ADD;
```

Implementing 2-D FFTs

The separability characteristics of the 2-D FFT allow an NXN image to be reduced to 2N N-point FFTs. The usual strategy is to take an FFT of the rows, and then an FFT of the columns. The data for the columnar case are separated by 8 locations. The job can be achieved by the first three passes of a 64 point FFT. This demands FSIZ:64,FPS:32,LPS:8. If the image is > 8X8 the VSP has to use more than one FFT instruction. Different scale factors can apply for different FFT operations. All image points must be aligned through scaling.

The Scaling System

A 16 bit register is divided into 4 nybbles and each nybble carries a 4 bit number that is the scaling factor for an FFT or IFFT operation. A Scale Register pointer fills the register from the LS end. The pointer is reset after the MS nybble is used, or after an LDSM instruction with MD:0,UP:1.

The Scale Register can be read by the STI STR:5, or by reading its memory mapped address. The largest scale factor generated in a series of FFTs (since the last reset) is stored in

the Maximum Scale Register (MSR). The MSR is reset by LDSM MD:0,UP:1. The replaced value of the MSR is retained in the Old MSR register by LDSM MD:0,UP:1.

Support Environment

The usual Assembler and Debugger are available but they are only adequate by modern standards. There are no trace windows. A library of interactive functions is supplied with the development system which is IBM resident. These are written in "C". Whereas the Microsoft compiler is recommended, our experience shows that Turbo C performs entirely satisfactorily.

A simulator program is available but cannot be commented upon as it has not been seen.

Conclusions

The signal processing performance of the VSP is very high and have allowed real time solutions to some of our applications. A distinct learning curve was necessary in order to cope with the horizontal architecture and the slightly idiosyncratic floating point and scaling requirements.

References

1. Very High Speed computing systems. M.J.Flynn. Proc.IEEE 54, 1901-9, 1966.
2. Computer Architecture- a modern synthesis.Subrata Dasgupta, Wiley,1989,vol 1.
3. A Fast Cosine Transform in one and two dimensions. J.Makhoul. IEEE Transactions on acoustics speech and signal processing. Vol ASSP-33, No.6 December 1985 pp1532-1539.
4. Zoran Technical Notes
TN92040-0187
TN92045-0187
TN92043-1087
TN94028-0187.

On Hardware for development of Signal Processing Software

G.A.King, Information Systems Division, Southampton Institute
of Higher Education

1.0 Introduction

Software for electronic signal processing can be created with high level languages such as "C", and whilst adequate, the problems associated with the speed necessary for real time operations on, for example, video signals, cannot be solved this way. An example might be a 512X512 pel FFT required in the inter field period for TV. This performance needs parallelism, and the software developer needs a sympathetic architecture. Courses having to consider this kind of software must select a suitable hardware platform for practical work. At S.I.H.E. the advent of BSc courses in Industrial Systems Technology, and Engineering with Business prompted some research into feasible platforms to support computer vision initiatives that are part of advanced robotics.

2.0 A Survey of the options

Such topics always generate the need for considerable computing power. It is a natural consequence of the 2-D spatial distribution of an image to map the pels into an array of processors, one processor pel⁻¹. This kind of thinking only requires simple processors of the bit serial type, and the array is classified according to Flynn⁽¹⁾ as SIMD (single instruction multiple data stream).

The idea that an array of simple processors execute

simultaneously a broadcast sequence of instructions allows segments of the image to be processed in parallel as described by Fountain⁽²⁾, and Pass⁽³⁾.

Whilst SIMD is an appropriate technology for simple processes performed by pel mapped bit processors, there are forms of processing, mainly concerned with higher level functions that are abstract in terms of the image data. Examples of the former simple processes are image filtering and feature extraction. An important type of higher level task is pattern recognition. The SIMD structure is not well suited to this and the requirements of higher level image tasks is MIMD as described by Flynn⁽¹⁾.

Multiple instruction multiple data stream systems can be organised in a number of different ways. A common memory can be shared by a number of sophisticated processors, alternatively each processor can have local memory and the image can be distributed among them. In the latter case it becomes most important that high speed communications, and a connectivity scheme be implemented. Where memory is common high speed communication is likely to be achieved by means of a parallel bus, and the system is classified as "tightly coupled". The distributed option is known as a "loosely coupled" system. The advantages and problems associated with these choices are discussed in Edwards⁽⁴⁾.

The conclusions reached by Edwards⁽⁴⁾ indicate that the communications bottlenecks in MIMD systems have not produced solutions where throughput is extremely high. Another problem concerns the sharing of tasks by a master processor and the need for a satisfactory synchronisation of task completion. Again

this proves a throughput constraint.

Conversely, SIMD architectures have become widespread and have good performance at the level of working discussed above.

Use in data compression for databases and for transmission schemes does infer the higher level of operation.

SIMD is not particularly suited to the application and MIMD may not reach the performance criterion even though costs would be high. This applies particularly when dedicated systems are considered.

SIMD arrays are largely designed to act as co-processors for mainframes and minicomputers. This is evidenced by the survey carried out by Kittler and Duff⁽⁵⁾. This survey also identifies MIMD difficulties having been solved by machines able to switch classification. The complexity inherent in both these options does not assist in developing systems for microprocessor hosts working magnetic or optical disc technology at the PC workstation level. Notwithstanding this discussion, SIMD and MIMD platforms would prove an expensive choice.

3.0 Affordable architectures

3.1 Peripheral Vector Processors

Rather than arrays of processors, with all their attendant costs, the most promising alternative is a processor that is optimised to operate on arrays. This could provide a cost effective co-processor within a PC type host. The question posed is can significant and satisfactory speeds be obtained?

This option, a Peripheral Vector Processor, is a notion that existed for some time and a number have been designed. A survey

(6)
by Karplus lists several examples. In general, these processors are classified as Horizontal architectures since they use uncommonly long microcode words with many fields. Each field controls adders and multipliers in a way that makes resources available in parallel. Another unusual feature of this type is the fact that the Assembler instructions specify the microcode level directly. Pipelining is used extensively during vector operations and layouts specialised to certain operations are common.

3.2 Digital Filter Processors

Convolution and Correlation are never far from mind in image processing environments. The interesting thing is that an FIR digital filter may easily perform spatial domain convolution by virtue of its configuration. Digital Filter systems are optimised to perform sum of products operations and some designs are reconfigurable and flexible. These Digital Filter Processors (DFPs) are based on the parallel operation of a number of cells. Extra parallelism in terms of recursive operation is described by Brumfitt (8).

4.0 Conclusions

Both VSPs (Vector Signal Processors) and DFPs (Digital Filter Processors) are commonly available as co-processor boards for PC hosts and their performances are remarkably good. Each option is supported by development software including assemblers, simulators, debuggers and application program libraries. Performance is adequate for applications in Machine Vision, Telecommunications, Medical Imaging, Digital Video, Electronic cameras and Military reconnaissance. Costs are around

fifteen hundred to two thousand pounds per host.

References

1. "Very High Speed computing systems". M.J.Flynn. Proc.IEEE 54, 1901-9 1966.
2. "A review of SIMD architectures" T.J.Fountain. Image Processing System Architecture. Research Studies Press 1986.
3. "The GRID parallel computing system". S.Pass. Image Processing System Architecture. Research Studies Press. 1986.
4. "A Review of MIMD Architecture for Image Processing". M.D.Edwards. Image Processing System Architecture. Research Studies Press. 1986.
5. "Image Processing System Architectures" Ed. J.Kittler, M.J.B. Duff. Research Studies Press. 1986.
6. "Architectural and Software Issues in the Design and Application of Peripheral Array Processors". W.J.Karplus. Computer vol 14 No.9 pp 11-17.
7. "A Review of other Architectural Concepts of Image Processing". P.J.Brumfitt. Image Processing System Architectures. Research Studies Press. 1986.

GAK/1/90

Vector Signal Processors and Digital Filters
in data compression for electronic publishing

G.A.King, Information Systems Division, Southampton Institute of Higher Education.

P.D.Picton, Electronics Discipline, Technology Faculty, The Open University.

Abstract

A pattern matching and substitution method is described for textual symbol entities. The method uses a matched filter classifier and compares its output with autocorrelation functions derived by fast convolution using a vector signal processor. A statistical method for match detection is suggested and experimental results are given. A review of Fast Cosine Transform algorithms for the transform coding of grey scale images concludes that Makhoul's algorithm is appropriate and implementation details for the vector signal processor are explained.

Introduction

The ideas developed in this paper find application in data compression for facsimile transmission or for document databases and processing which will be appropriate to new generations of photocopiers or desktop publishing. The approach is to use pattern matching and substitution for documents containing textual symbols and the Fast Cosine Transform for grey scale images. In each case the Vector Signal Processor is used for its

high performance fast convolution and for its applicability to transform coding.

The basic algorithm for pattern matching and substitution is after the fashion of Johnson, Segen and Cash^[1], and separately in a different context by Wilcox and Spitz^[2].

Textual data - Symbol Pattern Matching

The method assumes the use of document scanning, typewritten symbols and that individual entities have been previously segmented.

Once characters are isolated, their form is matched against those already recognised. If a match occurs the 2-D pixel image is not transmitted or stored. Instead a suitable short code is substituted, typically a variable length type in order to approach the source entropy and the result is data compression.

If a "no match" condition occurs the pixel image and the code to be associated with it in future is transmitted or stored for use in the data recovery phase. A small amount of negative compression is incurred for a "no match" condition but in practice this is more than compensated for by savings when matches do occur.

Flowchart 1 provides an overview of the strategy. The key process for successful application of these ideas is the pattern matching method.

Most workers in this field have used template matching, constructing an error pel map using weighted XOR to create a match figure of merit as did Holt and Xydeas^[3]. Many of the difficulties with template matching arise from registration of

segmented characters or anomalies due to very thin entities.

The bit maps for the two characters are combined by means of exclusively ORing them together. This creates an error pel map in which the error pels are weighted according to the number of adjacent error pels that exist.

The weighted error pel map is then totalled in order to generate a match figure of merit. A threshold for match is defined and decisions may then be made.

The difficulties mentioned above, concerning thin entities and registration can easily be seen to be exacerbated by the weighting procedure. This may be explained by considering a 1 pixel wide vertical element such as a thin "i" or "l". Slight mis-registration in an overlaid template could mean that no pixels line up. Thus every error pel would have adjacent error pels leading to a high total and "no match".

Holt and Xydeas have tried to compensate by further customising, or increasing the conditionality of their algorithms.

Other problems such as those reported by Boyle and Thomas^[4] concern the orientation of symbols.

Moving a template across a symbol and checking for best fit amounts to spatial domain convolution. The computational overhead has led to convolution being performed with very small kernels. Specialised microprocessor architectures such as the Vector Signal Processor give the potential of larger and much faster convolution.

The discrepancies of the weighted XOR map prompted an

investigation of an alternative scheme. Research into early optical work revealed the notion of using matched filters for pattern recognition as suggested by Horowitz and Shelton^[5], and correspondence by Kain^[6] supports the idea by proposing statistical measures of matches. The combination of a matched filter classifier and a statistical procedure involving correlation functions improves performance by creating insensitivity to misregistration.

Matched Filters

A matched filter is characterised by having an impulse response equal to the time reversed version of the signal waveform to which it is matched. In the context of this application, the filter is configured to an array generated from the segmented symbol.

When the symbol for which the filter is configured is applied, the output of the matched filter is the autocorrelation function (ACF) for that symbol. Incorrect symbols will produce a cross correlation function (CCF). The principle is illustrated by figure 1. The main matching algorithm recognises which by means of statistical methods.

This application calls for the need to match large numbers of symbols and consequently the matched filter required must be programmable. The impulse response must be capable of being varied by writing control words to it.

Implementing the Matched Filter

Lynn^[7] discusses matched filters and demonstrates a

particularly appropriate technique which applies to the sampled data system case. The assumption is that the signal is modelled as two-variable with binary states. This will generate an expression for the impulse response where the coefficients of the terms will either be 0 or 1. Clearly, where the coefficient is 0 then that term may be neglected.

In order to arrive at a configuration for the filter the following procedure was derived from Lynn^[7], and Blandford^[8]. Although for real segmented characters a much longer string is involved, consider a fragment of the single dimensional string representing a "slice" of the two dimensional character array.

101110000101.....

The time reversed version is :-

.....101000011101

Assuming a digital filter, the z plane transfer function will be :-

$$H(z) = 1 + z^{-2} + z^{-7} + z^{-8} + z^{-9} + z^{-11}$$

since $H(z) = Y(z)/X(z)$ then :-

$$Y(z) = X(z) + X(z).z^{-2} + X(z).z^{-7} + X(z).z^{-8} + X(z).z^{-9} ..$$

This allows the recurrence formula to be written as :-

$$y(n) = x(n) + x(n-2) + x(n-7) + x(n-8) + x(n-9) + x(n-11)$$

which may be implemented as a Finite Impulse Response (FIR) filter of the non-recursive type. For the beginning of the example sequence fragment the allocation of coefficients is :-

$$h(0)=1, h(1)=0, h(2)=1, \dots, h(11)=1$$

This system was committed to an electrically programmable logic device (EPLD). The delay elements are implemented as a shift register of D type flip flops and the multipliers could simply take the form of two input AND functions because of the basic two valued approach. The coefficient input to the multipliers comes from a D-type PIP0 (parallel in-parallel out) register. This register connects via address decoding and glue logic to the host processor synchronous bus. In this way the host can store a "configuration" word in order to set up the matched filter. The summation requirement is satisfied by cascaded full adders and output logic. The concept is shown in figure 2.

The problem with the two valued approach is that each character cell of, say, 16X8 pels results in long strings e.g. 128 pels. Each pel needs a signal shift register bit and consequently the logic units would need to be cascaded.

n-valued Model

An alternative approach involves treating each horizontal slice of the character cell as a single sample value, and thus each character would comprise of 16 samples. This will reduce the string length but there will be a compensating increase in the complexity of the multiplier circuits and the summation unit.

Using arbitrary coefficient values to illustrate, the recurrence formula might appear :-

$$y(n)=252+79.x(n)+22.x(n-1)+121.x(n-2) \dots$$

The requirement is for at least a double precision (16 bit) result, together with array multiply hardware similar to that described by Hayes ^[9].

The Vector Signal Processor

The Zoran VSP is optimised to solve digital signal processing problems and is an array processor having single command execution of Fast Fourier Transforms, Digital filters, matrix multiplies, polynomial expansion, peak detection and thresholding. The ALU handles complex floating point variables and can reach 37.5 Mflops. Other single instructions are; vector add, vector subtract and load/store of arrays. Looping constructs and subroutines are also supported.

Taylor ^[10] describes a variant VSP designated ZR34325 which features an FIR instruction. This instruction has a field specifying the number of taps in the desired filter and expects an array in external memory to define the coefficients necessary. This solution will not be as fast as a pure hardware implementation but Taylor reveals results showing the ZR34325 capable of 32 tap filter action with an array of 128 real samples in 382 microseconds. If this is a viable solution both the Matched filter requirements and the Correlation needs can be provided by the VSP.

Any application which can be fulfilled in the frequency domain is very efficiently implemented and the processor has a bus control regime which is compatible with operation as a co-processor in a PC type host. The VSP is a class of processor described as a peripheral vector processor in a survey by

Generating Autocorrelation Functions

Flowchart 1 shows the overall pattern matching and substitution process. It is implied that for each character a set of full details will be created when it is encountered for the first time. This set has the following constituents :-

1. A bit reversed version of the character cell to be used as configuration input to the matched filter.
2. An Autocorrelation function of the character cell sample values.
3. A short code allocated to this symbol.

Since a large number of new symbols will have to be dealt with in the early stages of textual processing it is advantageous if the Autocorrelation functions can be generated quickly. The need is for fast correlation. Fast correlation involves translations into the frequency domain.

Time domain

convolution is :-
$$y(n) = \sum_m x(m) \cdot h(n-m)$$

and correlation is :-
$$R(m) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_m x(n) \cdot y(n+m)$$

The applicability to the task in hand is evident if $h(n)$ is the impulse response of the matched filter, $m=0$ to n , $x(n)$ is

the input sample sequence and $y(n)$ is the output. N is a power of 2 greater than the length of the shorter sequence. If DFTs (discrete Fourier Transforms) are used then the expressions become :-

for convolution,

$$Y(k) = H(k) \cdot X(k)$$

and for correlation, $S(k) = H(n-k) \cdot X(k)$

$$\begin{aligned} \text{Where :- } \text{DFT } [x(n)] &\rightarrow [X(k)] \\ \text{DFT } [h(n)] &\rightarrow [H(k)] \\ \text{DFT } [y(n)] &\rightarrow [Y(k)] \\ \text{DFT } [R(m)] &\rightarrow [S(k)] \end{aligned}$$

The key point is that fast convolution can be used for correlation if $H(k)$ is stored in bit reversed order. The array store instructions for the VSP allow for this.

The process of frequency domain convolution is used to create the library ACFs. This job is invoked when a segmented character is not recognised as having been seen before at reference 120 in flowchart 2.

Details of the programming are illustrated by figure 3 and listing 1. Proof of the method is available in Gonzalez [12] and Wintz [13] and Zoran [13]. The listing shows the characteristics of the horizontal architecture of the device, since each instruction has uncommon length with many fields.

The Screening Stage

Before the main matching process proceeds, a rough

comparison or screening stage takes three horizontal slices (top, centre and bottom) and one vertical slice in the centre of the segmented symbol cell. A bitwise XOR of is performed with the same slices of the set of library characters.

A majority vote algorithm notes that if three out of four slices compare within the set tolerance to a library symbol, then the item is regarded as a possible candidate and is tested with the main matching technique.

Starting with a library of characters, each character was screened against the complete set including itself. The results are shown in table 1. The average number of characters selected as "possibles" needing to be checked by the main technique was 3.73, representing just under 15% of the library size. In practice the library will contain a number of differing "versions" of the same character and this is expected to increase the number of "possibles". The screening stage did not fail to select the correct character amongst the "possible" subset.

There is a trade off between the screening and main techniques. If the screening is rigorous the main technique need only be invoked for a few candidates. The faster the processing for the main technique, the less demanding the screening need be. Results support the view that screening may be cursory when compared to more traditional pattern matching. Certainly less exhaustive than the height/width/internal black run suggested by Holt and Xydeas^[3].

The Main Statistical Technique.

The essence of matching is to detect significant differences. Kain^[6] proposed several measures of similarity, the most useful is a figure derived by comparing the standard deviation of the filter output and the library ACF and then dividing by the mean for each sequence.

It was felt that a better measure could be devised. The problem is the need to be able to determine whether the difference between sample sequences is significant or whether it arises from noise or other random effects. A method for doing this is described by Harrison^[14] and originates in statistical estimation techniques, such as those described in Chatfield⁽¹⁵⁾. The mechanism is explained as follows :-

Two statistical measures are Standard Deviation, and the Mean.

A set of samples will have values for these two parameters. If there are several sets of samples then the Standard Error of the Means is the Standard Deviation for the set of means.

If a series of sample values is not significantly different then its mean will be within the Standard Error predicted. If two sets of samples are possessed, one selected set should have its mean and standard deviation calculated. Knowing the number of samples it is possible to predict the Standard Error which would apply to a group of similar sets. The same procedure may be repeated for the other set and then the Standard Error of the difference of the means must be calculated.

The actual difference between the two means is then

compared with the Standard Error of the difference.

The results are interpreted as follows :-

If two sets of data differ only by random effects, then their means have a 68% chance of falling within a range defined by the standard error of the difference of those means. Other probabilities follow the normal distribution, so that there will be a 99.73% probability of being within three times the standard error of the difference. When viewed in the inverse sense this means that a symbol differing by three standard deviations has less than 1% probability of being the symbol compared for.

The pre-requisite for the method to be viable is the ability to calculate the standard error of the difference, starting with two sample sequences. Flowchart 3 defines the necessary steps. The final match figure of merit is obtained by dividing the difference of the two sample means by the standard error of the difference. Low values (<1) indicate a high level of confidence in the match. Figure 4 demonstrates the method for two sampled waveforms. The resulting figure of merit (modulus of r), being 2.544 indicates a very low match probability.

The nature of these calculations is such that the VSP is not well suited and the host 80386 was used.

Programming the VSP

Array processing instructions have fields denoted MBA, MBS and MSS. MBA is the memory base address for the array, MBS specifies the number of elements to be transferred to internal memory for processing, whilst MSS is the memory step size. The

assembler instruction fields :-

NMPT:32,MBS:4,MSS:16,MBA:2000H;

involve an array at external memory 2000 hex, consisting of 32 elements, being split into two groups of 16, and only the first 4 elements of each are to be accessed for processing.

This is obviously very powerful in the context of the problem and allows the screening algorithm to be implemented on the VSP. FFTs and IFFTs are achieved with a single instruction and examples can be seen in listing 1. When using the VSP for frequency domain correlation arrays of 128 points can be correlated in 500 microseconds. Although the full system has not yet been integrated, calculations using CCITT document number 1 indicate that the necessary ACFs can be generated in 2-4 milliseconds for a short business letter. The slowest operation is the statistical matching process.

Experiments and Results

Kains method and that developed from Harrison were the subject of a comparative study. Complying with the original experiments by Kain, only the alphabetic character set was used.

ACFs were generated for each symbol and then stored in a library. Each symbol to be recognised was screened against all the rest in order to determine which, according to the screening algorithm, were candidates for the main matching technique. The results are shown in table 1. CCFs were generated for each candidate using a matched filter set up for the original subject symbol.

If the original subject symbol was "a", then the screening

selected "a,g,q". The correlation functions derived were "ag", "aq", and "aa". This last ideally being the same as the ACF from the library. The correlation function "aa" should yield the highest match probability, i.e. the lowest numerical output from the statistical match routine. The statistical process would have, as inputs :-

- (1) The ACF of "a"
- (2) The correlation functions "ag", "aq", "aa" in turn.

The outputs for a representative series from the tests are shown in table 2. In the table an entry such as "bw" means that the ACF for character "b" was compared with the cross correlation of "b" and "w" (the output that would have resulted from "w" being applied to the matched filter configured for "b").

The results in the column relating to the method due to Harrison were derived as shown in figure 4. The table does not list the result of "bb", "ww", "ff" etc since in all cases this would yield 0. Zero means a "perfect" match. The results column entitled method due to Kain is derived by comparing the standard deviation of the matched filter output to that of the library ACF and then dividing by the mean. This was the best of the techniques suggested in Kains original research and the figures are provided to allow a comparison.

Table entries with numerical elements in their names such as "i2" or "e5" refer to symbol variations due to corruption, noise or other distortions. In the experimental work symbol patterns such variations were added to the ACF library. Table entries in this class were a1,i1,b1,e1. Symbols containing greater

distortion were matched against these in order to evaluate the relative tolerance of the two statistical match methods in cases of "same character but corrupted".

Although Kain reported 87% correct recognition in his original work there was a serious problem in differentiating between "i" and "j". This is confirmed by table 2 entry "ij". Note that the new match procedure has no difficulty and clearly rejects this case, although in an integrated system as proposed the problem would not arise because the screening process only selects "i" and "l" for further matching.

The figures for "ij" infer that the probability that they match is that associated with being 6.76 standard deviations from the mean in a Gaussian distribution. Kains match procedure generates 0.74, an indecisive value.

For the purposes of experiment it was decided to set the match figure of merit at 1.6. Outputs from the match procedure having values less than this would be declared the "same" symbol. This threshold is adjustable but 1.6 represents 3 or 4 uncompensated error pels. The threshold would normally be selected to define the acceptable distortion provided the match procedure is sufficiently discriminating and consistent. The method due to Kain produced poor results in this respect. Where "b" was the subject, screening suggested w,f,k,m,p,b and Kains method gives probable matches with "k" and "p" as well as "b". The new method has no such problem. Other examples are provided, but it is clear that the new method is much more consistent and discriminating. Table entries I1I2,B1B2,and E1E5 were for symbols differing only by small amounts of random noise. Entries

A1A2, E1E2, and E1E4 all involved seriously distorted characters.

With the font used only "x" and "c" have yielded possible false match decisions and this has now been eliminated by the addition of vertical slicing at the screening stage.

Grey scale images - The Fast Cosine Transform

The performance of the Principal Component Transform is approached by the Cosine Transform when the original data is strongly correlated as is usually the case with image data.

The purpose of transform coding is to map the signal into a transform space in order to make the samples more independent or uncorrelated. This allows a data compression to be achieved by the quantising stage. The assumption is that the image has been digitised and split into sub-pictures.

The Vector Signal Processor is designed for array operations and for the Fast Fourier Transform, but the decision to use the Cosine Transform prompted the investigation into which of several fast discrete options are most appropriate.

Alaul Haque⁽¹⁶⁾ proposes a 2-D algorithm working directly on 2-D data sets. The method involves the partitioning of matrices and the subsequent regrouping of submatrices. For large arrays the computational and floating point overheads are great.

Ghanbari and Pearson⁽¹⁷⁾ suggest a Fast Cosine Transform (FCT) algorithm based on Hadamard sparse matrices. The bonus being that coefficients assume only 1,0,-1 states, allowing relatively simple hardware implementations. Since the VSP is quite happy with floating point operations this is an unnecessary

limitation.

Byeong Gi Lee⁽¹⁸⁾ and Makhoul⁽¹⁹⁾ describe FCT implementation based on the Fast Fourier Transform (FFT). Both of these methods are use on a procedure which re-orders the input data.

The development choice was Makhouls algorithm, which is defined by figure 5.

If the input data sequence is $x(n)$, then the re-ordered data $v(n)$ is derived as follows :-

$$v(n) := \begin{cases} x(2n) & 0 \leq n \leq \left[\frac{N-1}{2} \right] \\ x(2N-2n-1) & \left[\frac{N}{2} \right] \leq n \leq N-1 \end{cases}$$

Where $[]$ indicates " the integer part of". This means that the re-ordered sequence is obtained by taking the even points in $x(n)$ in order, followed by the odd points in reverse order. The method then calls for the DFT of $v(n)$ to yield $v(k)$. This may be executed as an FFT. The final step is to multiply the array $v(k)$ by $2 \exp[-j \pi k/2N]$.

VSP implementation of the FCT

Assuming a normally ordered array of samples of size N residing at memory base address (MBA), $MBA:SUBPICTURE$, the first task is to load the array into the internal memory of the VSP system - even points first.

```
LD NMPT:N/2,MBS:1,MSS:2,RV:0,MBA:SUBPICTURE+1;
```

This instruction loads half the number of points selecting

every other point beginning with point 2, normal order. Next the odd points must be loaded in reverse address order. The RV field is capable of prescribing bit reverse operations on addresses or data, but what is required here is a re-ordering which can be achieved by loading in normal order, storing backward, and then loading again in normal order. The bit reversal operations are provided for certain FFT operations and is not appropriate to Makhoul's algorithm. The instruction sequence needed is :-

```
LD NMPT:N/2,MBS:1,MSS:2,RV:0,MBA:SUBPICTURE;  
STB NMPT:N/2,MBS:N/2,MSS:N/2,MBAB:SCRATCH+N/2;  
LD NMPT:N/2,MBS:N/2,MSS:N/2,MBA:SCRATCH;
```

The STB instruction stores data arrays whilst decrementing the memory base address. MBAB defines the base address when storing backward. The results are that data is loaded into VSP memory as follows: $x(2), x(4), x(6), \dots, x(n/2), \dots, x(7), x(5), x(3), x(1)$.

It is necessary to have precalculated $2\exp(-j k/2N)$ for $0 \leq k \leq (N/2)$. This task is best done by the VSP and to this end the instruction set provides a special instruction.

The DEMO instruction multiplies a complex vector in internal memory by a series of complex coefficients generated from a look up table which contains 256 Cosine values ranging from 0 to 90 degrees. After multiplying each element of the specified Cosine vector with corresponding array in internal RAM, the products are placed in internal RAM also.

There are a number of microword control fields unique to the DEMO instruction. These are :-

RBA - ROM base address

This parameter defines the starting angle of the Cosine vector, and is set to 10x the value in degrees. i.e. to start at 90 degrees RBA=900. The internal look up table contains 1024 discrete values and this makes only some values for RBA legal. The rule is :-

If "i" is the angle, $i \cdot 3600/1024$, where $0 \leq i < 1024$

RDA - ROM decrement address.

This address is used to define the incremental angles for successive Cosine coefficients. As with RBA the parameter is specified as 10x the required value in degrees. Legal values belong to the set ;

$(i+1) \cdot 1800/512$, $0 \leq i \leq 511$.

VSIZ - Vector size

Specifies the number of samples beginning with RBA to be addressed from the internal look up table. After this value the look up table address reverts to the RBA value. Legal values are 4,8,16,32,128 points.

The sequence of instructions required is as follows :-

FFT

DEMO (multiply by $2 \exp(-j k/2N)$)

ST (leave result back in external memory)

The inverse FCT developed by Makhoul requires that the input data be first complex conjugated, rotated by the MODLT instruction and then inverse FFTed. The MODLT instruction is exactly analogous to the DEMO instruction except that the frequency is

translated up in frequency instead of down. Complex conjugation is specifically provided by the CMCN VSP instruction. The CMCN is not required for the forward transform and so the inverse transform takes about 10% longer to execute.

Quantising

A spatial domain array and its transformation by the FCT are shown in figure 6(A) and 6(B)

Merely transforming into the frequency domain does not generate a compression of itself. As can be seen from figure 6(B) the transformation has produced strong compression potential with all the high valued coefficients bunched together followed by a long run of similarly valued ones. This long run consists of zero or near zero valued coefficients which may be allocated zero.

These contribute very little and may be ignored or compressed because of their redundancy. Zero valuing small but non-zero coefficients will create some small distortion when the data is reconstituted. The amount of distortion acceptable for a given system will determine what coefficients must be considered significant, and which may be zeroed. The allocation of bits to coefficients either zonally, i.e. only allocating significant values to a particular portion of the frequency spectrum - in this case equivalently low pass filtering, or by threshold i.e. zeroing all values below a given amplitude, is a function of quantising.

Whichever quantising equivalent scheme is chosen, the result is that a Run length or Huffman scheme can provide

FCT	yes	0.95mS	16X16 2-D
Fourier Descrip	yes	0.04mS	16, 1-D rotate
Hotelling	no	-	-
Screening	yes	0.03mS	per 16X16 cell
Statistical			
Matching	no	-	-

References

- [1] Johnson,O.,Segen,J.,Cash,G.L.,"Coding of two level pictures by pattern matching and substitution", Bell System Technical Journal , Vol 62 No.8 pp2513-2520, October 1983.
- [2] Wilcox,L.D.,Spitz,L., "Automatic recognition and representation of documents",Document manipulation and typography , Proc Int. Conf on electronic publishing, Nice (France) April 20-22 , Ed J.Van Vliet. 1988.
- [3] Holt,M.J.J.,Xydeas,C.S., "Recent developments in image data compression for digital facsimile", ICL Technical Journal May pp123-146,1986.
- [4] Boyle,R.D.,Thomas,R.C., "Computer Vision ", Blackwell Scientific 1988.
- [5] Horowitz,L.P.,Shelton,G.L., "Pattern recognition using autocorrelation", Shelton G.L., Proc IRE 49 pp175-185,1961.
- [6] Kain,R.Y., "Autocorrelation pattern recognition",Proc IRE 49 pp 1085-1086,1961.
- [7] Lynn,P.A.,"An introduction to the analysis and processing of signals", Methuen, 1983

- [8] Blandford,D.K., "The Digital Filter Analyser", Addison Wesley, 1988.
- [9] Hayes ^{J:P} "Computer Architecture and Organisation" , *McGraw Hill* 1979
- [10] Taylor,D.M., "Single commands for complex DSP functions", Electronic Product Design, pp31-37, November 1987.
- [11] Karplus,W.J., "Architectural software issues in the design and application of peripheral array processors", Computer, No.14, pp11-17, 1981.
- [12] Gonzalez,R.C., Wintz,P., "Digital Image Processing", Addison Wesley 1982.
- [13] Zoran, "Fast Convolution", Technical Note TN 92045-0187.
- [14] Harrison,R.D., "Statistics and Reliability", Open University, 1976.
- [15] Chatfield,C., "Statistics for Technology", Chapman and Hall, 1978.
- [16] Alaul Haque,M., "A two dimensional fast cosine transform", IEEE transactions on acoustics, speech and signal processing, Vol ASSP-33, No.6, pp1532-1539, 1985.
- [17] Ghanbari,M., Pearson,D.E., "Fast cosine transform implementation fir television signals", IEE Proc. Vol129 Pt F No.1 pp59-68, 1982.
- [18] Byeong Gi Lee, "FCT- a fast cosine transform", IEEE Proc, pp28A.3.1-28A.3.4, 1984.
- [19] Makhoul,J., "A fast cosine transform in one and two dimensions", IEE transactions on acoustics, speech, and signal processing, Vol ASSP-28, No.1, pp27-34, 1980.

Figure 1	"Matched filter context"	
Figure 2	"Matched filter system"	
Figure 3	"Fast Convolution data flow"	
Figure 4	"Main statistical matching technique"	
Figure 5	"Makhouls FCT algorithm"	
Figure 6a	"Spatial domain signal"	
Figure 6b	"Result of FCT on (a) above"	
Flowchart 1	"Pattern matching and substitution overview"	
Flowchart 2	"Full pattern matching and substitution algorithm"	
Flowchart 3	"Statistical matching algorithm"	
Listing 1	"VSP fast convolution program"	
Table 1	"Screening results"	
Table 2	"Matching figures of merit"	
Table 3	"VSP applicability conclusions"	<i>(in text)</i>

Figure 1

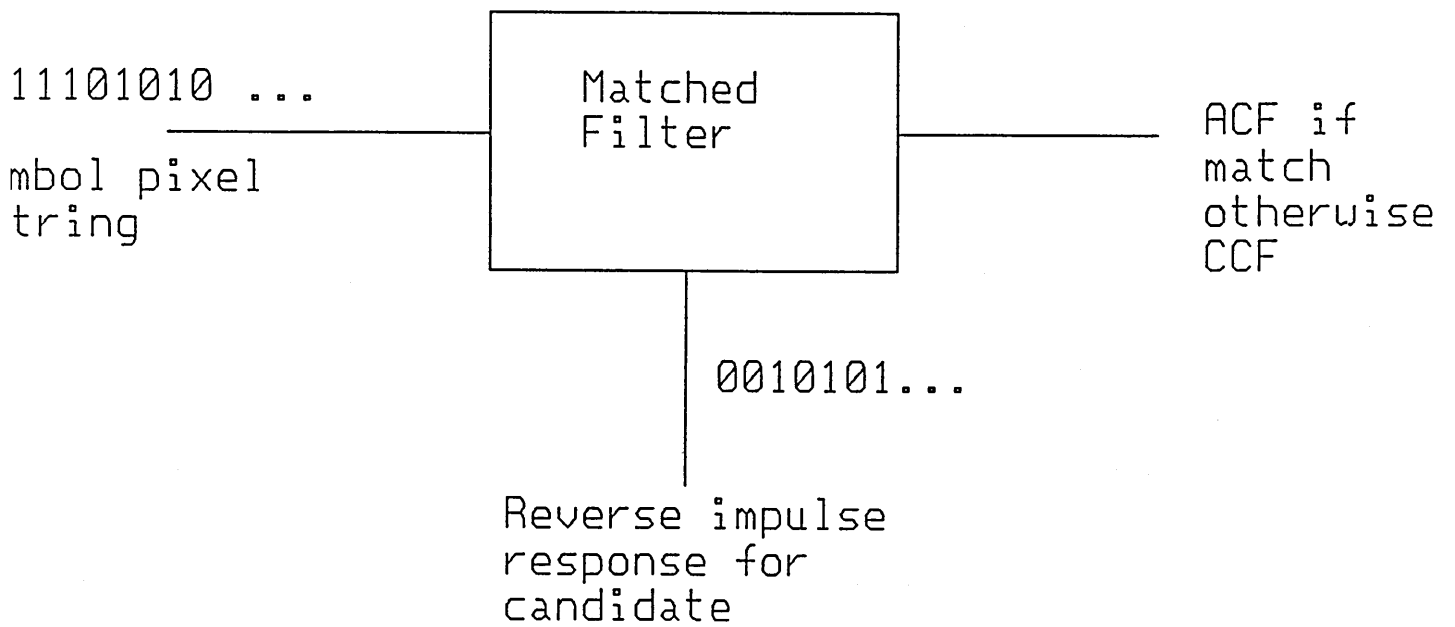
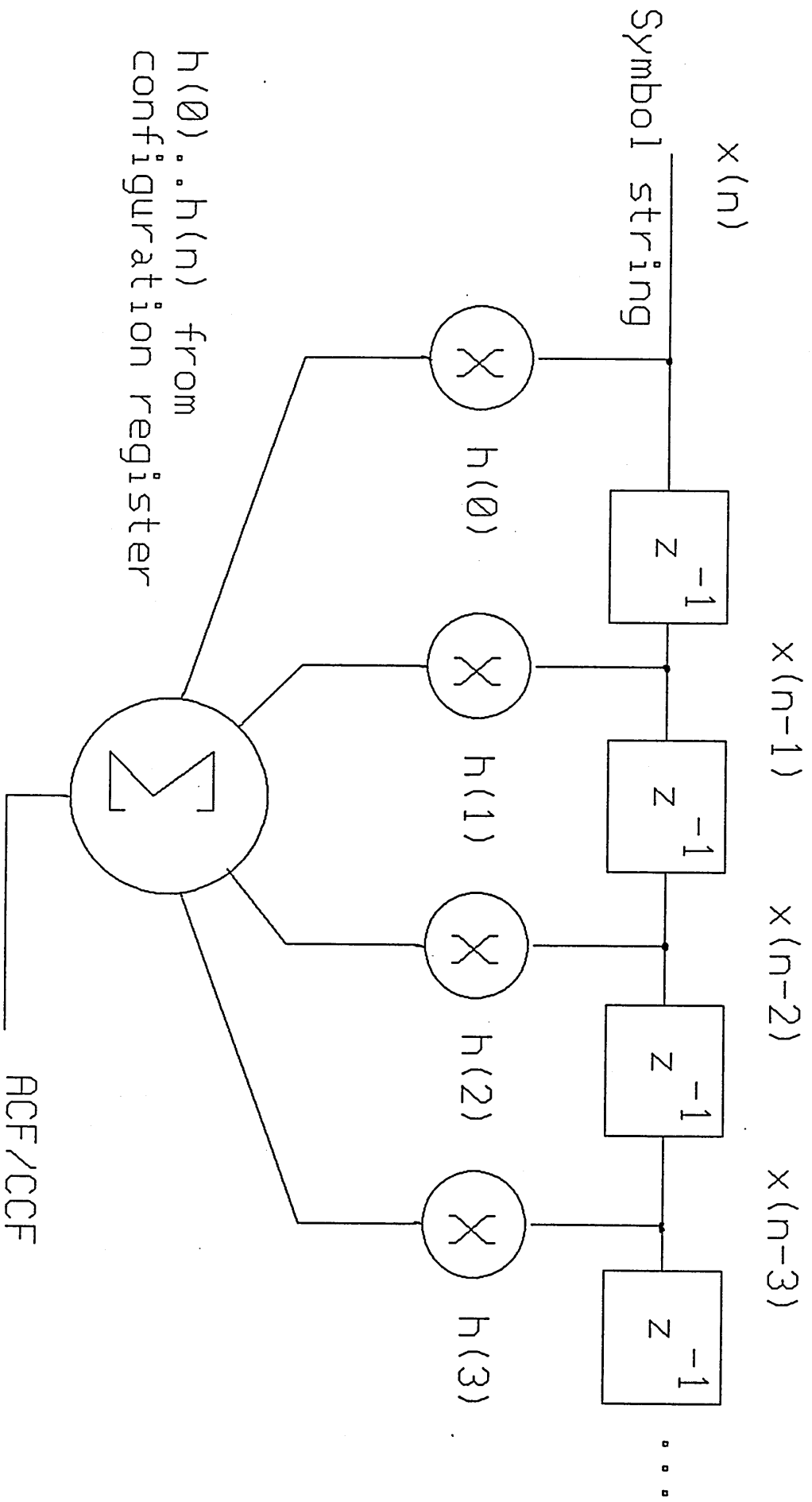
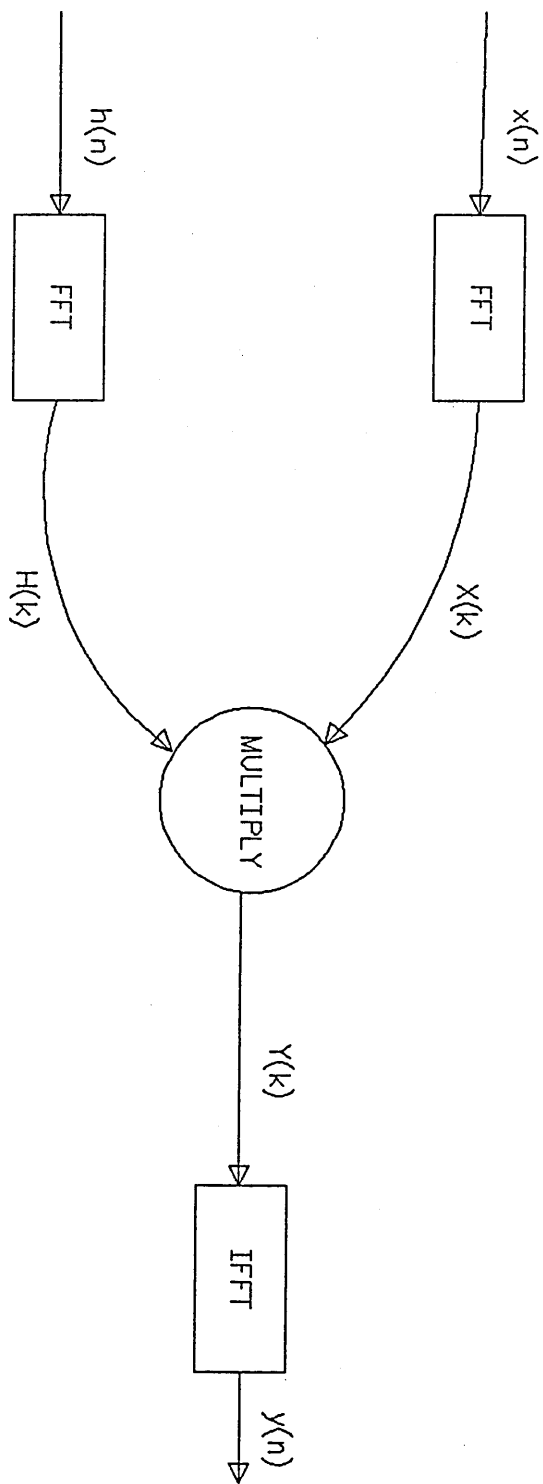


Figure 2

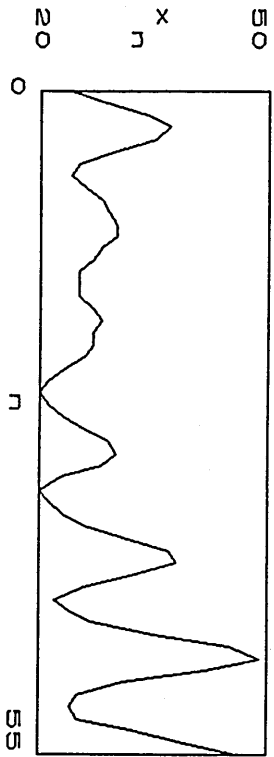


$h(0) \dots h(n)$ from
configuration register

Frequency Domain Convolution



Library ACF - candidate



```
mean(x) = 28.75
stdev(x) = 5.52
```

```
b := mean(x)
f := stdev(x)
```

Find the standard error in each case

$$j := \frac{f}{\sqrt{56}}$$

$$j := \frac{h}{\sqrt{56}}$$

Find the standard error of the difference

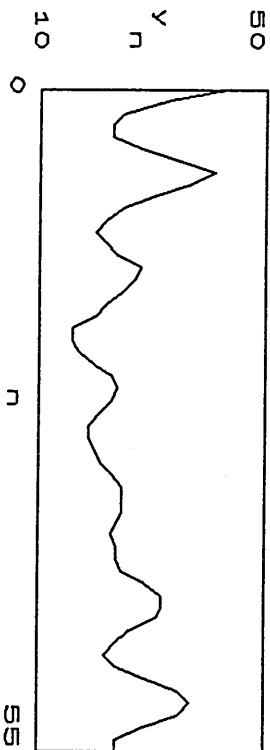
$$k := \left[\sqrt{2 + j^2} \right]$$

Calculate the ratio of the standard error of the difference to the difference of the means

$$r := \left[\frac{d - b}{k} \right]$$

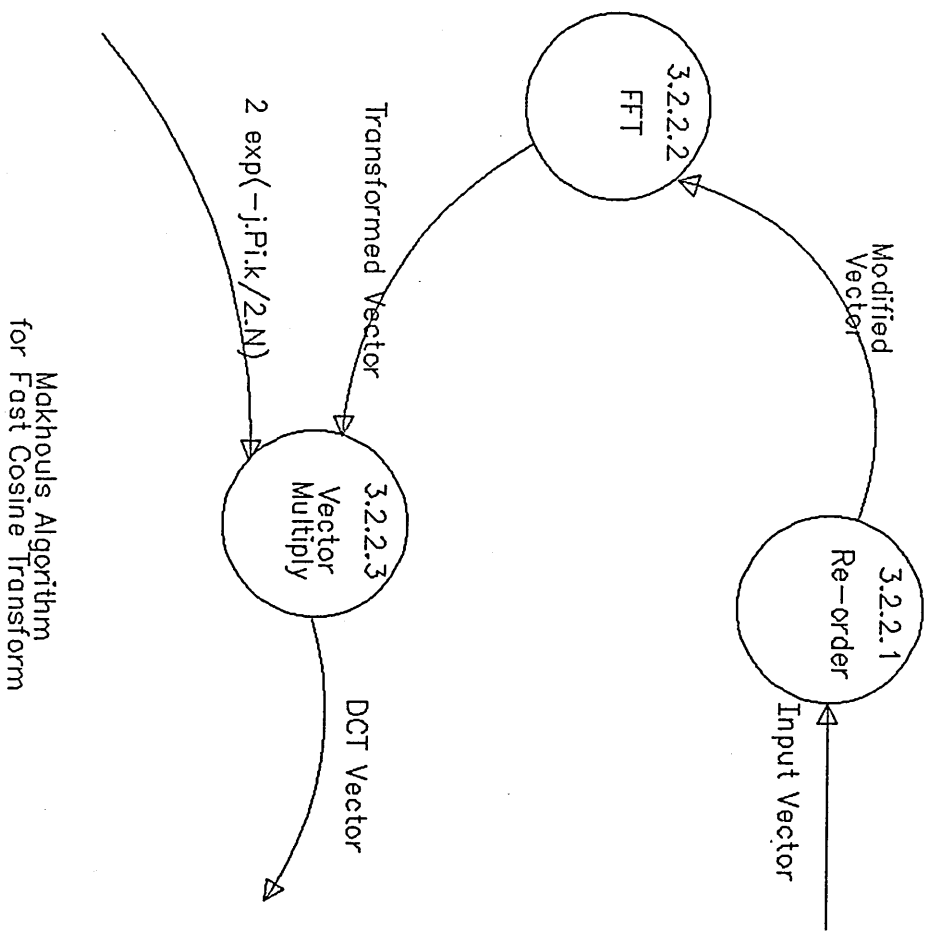
$$|r| = 2.544$$

Output from matched filter



```
mean(y) = 25.768
stdev(y) = 5.868
```

```
d := mean(y)
h := stdev(y)
```



Makhoul's Algorithm
for Fast Cosine Transform

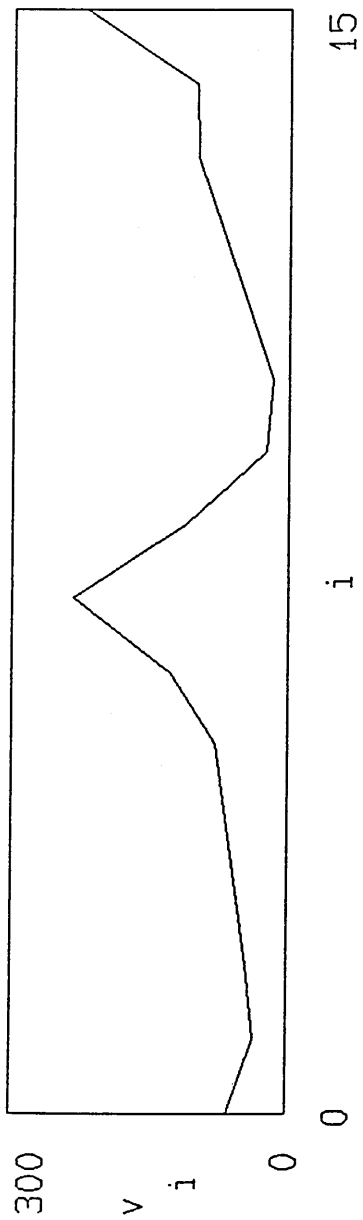


Fig. (6)

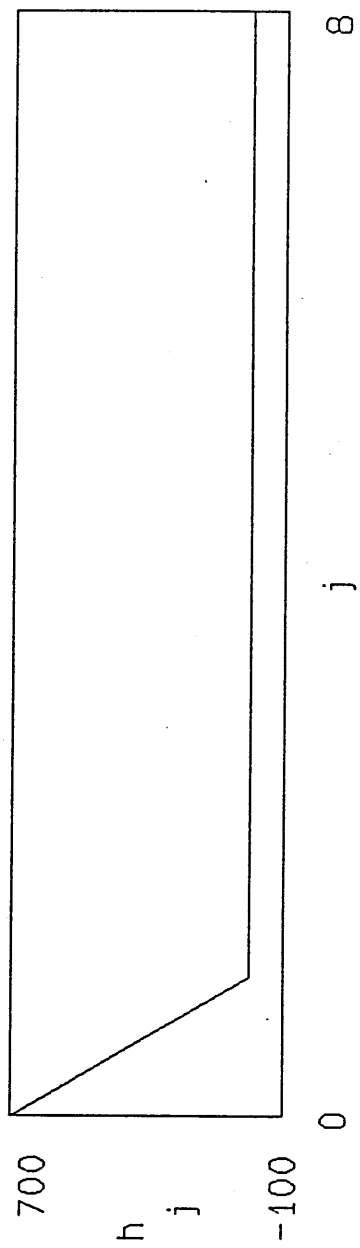
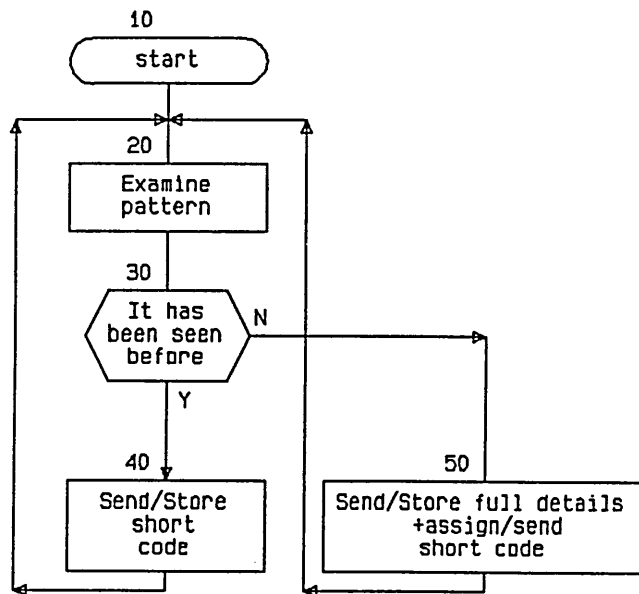


Fig. (7)

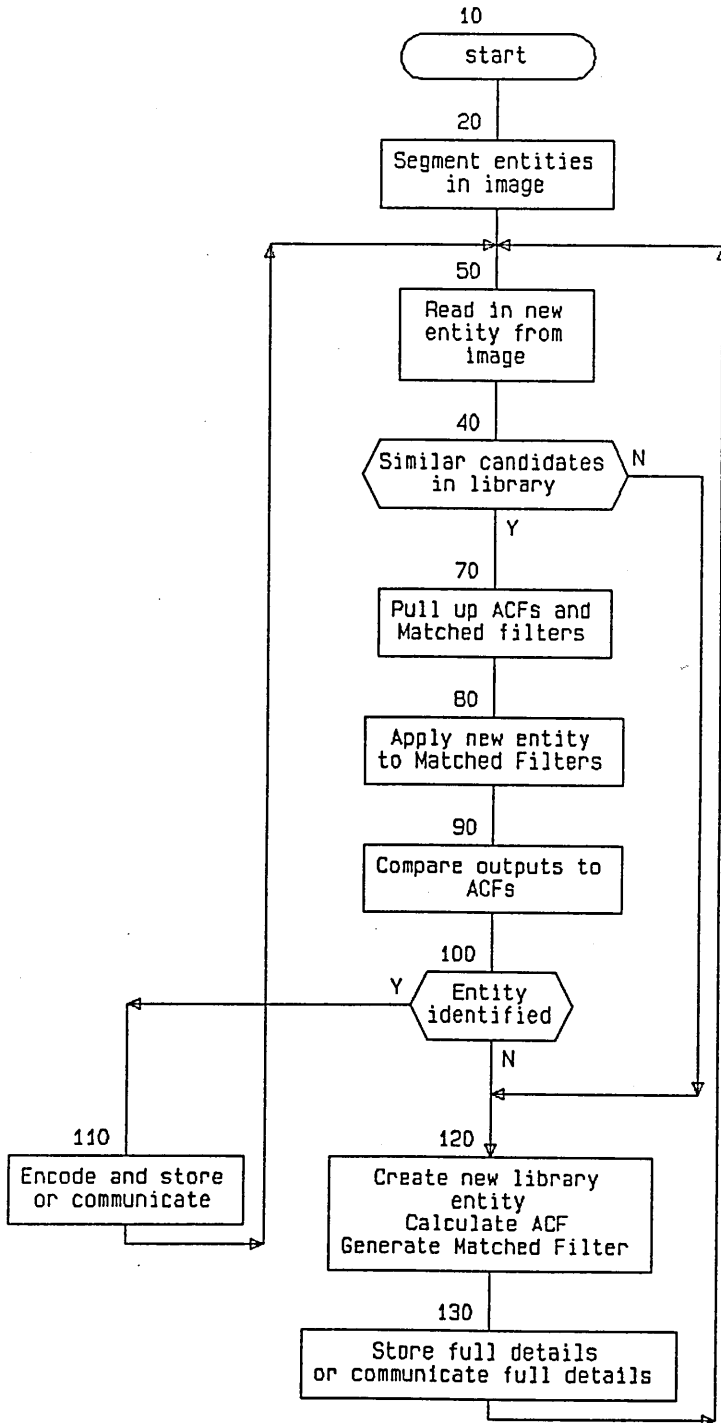
sbs



Pattern matching and substitution
basic idea
G.A.King

Kindra V1.B
sbs VERSION 6
15-Sep-89 10:17:

research



stats

10

start

20

Find the mean of each sample
and hence the difference of means

30

Find the standard deviations
of the sets

40

Find the standard error by
dividing the standard deviation
by the sq. root of the number of
items in the sample

50

Find the standard error of the difference
by taking the sq. root of the sum of the
squares of the standard errors

60

Compare the ratio of the standard
error of the difference to the
difference in means

70

end

Method for testing the
significance of differences
between sample sets

G.A.King ver 1.01

Kindra V1.B
stats VERSION 4
02-May-89 09:18:

Table 1

Character	Selected by screening
a	a,g,q
b	b,f,k,m,p,u
c	c,e,s,v,x
d	d,s
e	e,c,o,v,x
f	f,b,k,m,n,p,u,w
g	g,a
h	h,r
i	i,l
j	j,z
k	k,b,f,m,n,p,u,w
l	l,i
m	m,b,f,k,u
n	n,f,k,p,u
o	o,e
p	p,b,f,k,m,n,u,w
q	q,a
r	r,h
s	s,c,d,v,x
t	t
u	u,f,k,n,p
v	v,c,e,s,x
w	w,b,f,k,m,p
x	x,c,e,s,v
y	y
z	z,j

Table 2

Match figures of merit		
Subject symbol and comparisons	Method derived from Harrison	Method due to Kain
b		
bu	18.10	9.80
bf	16.33	1.90
bk	13.80	0.97
bm	10.61	4.74
bp	7.94	0.45
u		
uf	40.26	0.86
uk	34.03	1.86
um	6.15	0.52
up	28.10	0.10
f		
fk	1.89	0.52
fm	17.50	4.24
fn	5.91	0.41
fp	5.38	1.26
fu	6.52	0.32
fu	21.91	5.69
i		
ij	6.76	0.74
j		
jz	7.83	3.42
x		
xc	0.53	1.34
xs	1.71	2.24
xv	2.11	0.87
xe	4.38	2.57
a1		
a2	3.34	0.81
i1		
i2	0.34	1.20
b1		
b2	0.75	0.14
e1		
e5	0.68	0.97
e4	11.63	0.16
e2	6.97	2.21

LOC	OBJ	LINE	SOURCE
		1	/* Fast Correlation for VSP */
		2	
		3	/* Cooley-Tukey Algorithm for X(N) and H(N) R:0 *
		4	
		5	/* Sande-Tukey for IFFT R:1,FPS:1,LPS:64 */
		6	
		7	/* Fixed divide by 2 scaling-each pass AS:1 */
		8	
		9	/* G.A.King Data Compression version 1.00 11/10/
		10	
		11	/* SET UP INSTRUCTION FIELD DEFAULT VALUES */
		12	
		13	DEFAULT EI:0,INTRP:0,RV:0,ZR:0,ZP:0,MBS:128,MSS:
		14	
		15	DEFAULT OR:0,CN:0,MDF:3,RS:0;
		16	
		17	DEFAULT AS:0,FSIZ:128,FFT.RBA:0,ADF:3,SB:0,LN:1;
		18	
		19	
		20	/* OTHER DECLARATIONS */
		21	
		22	equ SCRATCH=0xFFF,XN=0x200;
		23	
		24	equ N=128,HZ=0x300,F=0x400,SCALE=0xFFC;
		25	
0000	0040	26	M: dw 64;
		27	
0001	0000	28	PROGSTART: dw 0;
		29	
		30	equ OUT=0x500;
		31	
		32	org 0
		33	
		34	/* SET UP SINGLE RAM SECTION MODE */
		35	
0000	0010	36	LDSM NMPT:1,MD:1,UP:0,MBA:SCRATCH;
0001	A170		
0002	0FFF		
		37	
		38	/* COMPUTATION OF H(Z) STORED AS H(N-K)
		39	
		40	AND STORED AS H(Z) AS THE TWO ELEMENTS FOR AU
		41	
0003	0000	42	LD NMPT:N,MDF:2,ZR:1,MBA:HZ;
0004	E050		
0005	0300		
		43	
0006	9406	44	FFT NMBT:N,R:0,FPS:64,LPS:1,I:0;

```

0007 0030
0008 0020
45
0009 0800 46 STB NMPT:N,MBAB:F;
000A E068
000B FBFF
47
000C 0800 48 ST NMPT:N,MBA:0x500;
000D E060
000E 0500
49
50 /* RESET SCALE REGISTER POINTER AND MAX SCALE RE
51
000F 0010 52 LDSM NMPT:1,MD:0,UP:1,MBA:SCALE;
0010 A270
0011 0FFC
53
54 /* PERFORM VECTOR MULTIPLICATION */
55
0012 0000 56 LD NMPT:N,MDF:2,MBA:F;
0013 E040
0014 0400
57
0015 1006 58 MLTC NMPT:N,SH:1,MBA:0x500;
0016 E068
0017 0500
59
60 /* NOW PERFORM INVERSE FFT */
61
0018 9406 62 FFT NMBT:N,R:1,FPS:1,LPS:64,I:1;
0019 8180
001A 0023
63
64 /* STORE THE SCALE REGISTER IN EXTERNAL MEMORY *
65
001B 0811 66 STI NMPT:1,STR:5,EI:1,MBA:SCALE;
001C 4050
001D 0FFC
67
68 /* STORE THE ACF IN EXTERNAL MEMORY */
69
70 /* REMEMBERING TO EXCLUDE INCORRECT RESULTS OF C
71
72 /* number of valid points is N-M-1 */
73
001E 0BF0 74 ST NMPT:63,MDF:1,MBA:OUT;
001F E020
0020 0500
75
0021 C000 76 HLT;
0022 0000
77
78 end;
79

```

S Y M B O L	TYPE	SEG	VALUE/SIZE	AT
F	CONSTANT	C	0x0400	
HZ	CONSTANT	C	0x0300	
M	LABEL	C	0x0000	
N	CONSTANT	C	0x0080	
OUT	CONSTANT	C	0x0500	
PROGSTART	LABEL	C	0x0001	
SCALE	CONSTANT	C	0x0FFC	
SCRATCH	CONSTANT	C	0x0FFF	
XN	CONSTANT	C	0x0200	

ASSEMBLY COMPLETE
 CODE SIZE: 25H, 37D
 79 LINES READ
 0 WARNINGS DETECTED
 0 ERRORS DETECTED

APPENDIX B

Vector Signal Processor Programs

LOC	OBJ	LINE	SOURCE
		1	
		2	
		3	/*
		4	256 point FFT
		5	G.A.King 7/89
		6	The importance of the scaling operation is
		7	emphasised
		8	*/
		9	
		10	org 0;
		11	
		12	DEFAULT EI:0, MDF:3, ZR:0, INTRP:0, ZP:0, AD:0,
		13	DEFAULT AS:0, R:0, FSIZ:128, FFT.RBA:0, ADF:3;
		14	
		15	DEFAULT FFT.I:0;
		16	
		17	
		18	/*Two RAM sections necessary for this size of FF
		19	*/
0000	0010	20	LDSM NMPT:1, RS:0, MD:1, UP:0, MBA:MODEADD;
0001	A170		
0002	0079		
		21	
		22	/*
		23	First stage of the 256 point FFT. Six passes
		24	eight pass FFT to be performed. Operations o
		25	sections alternately to take advantage of par
		26	
		27	*/
		28	
		29	DEFAULT MBS:1, MSS:4, FPS:32, LPS:1;
		30	
		31	/*
		32	With MBS:1 and MSS:4 every fourth complex loc
		33	is addressed.
		34	With FPS:32 and LPS:1 the data in each butter
		35	thirty two points apart for the first pass an
		36	for the last pass.
		37	
		38	*/
		39	
0003	0400	40	LD NMPT:64, RS:0, MBA:IN;
0004	0460		
0005	03E8		
0006	9206	41	FFT NMBT:64, RS:0;
0007	0070		
0008	0020		
0009	0408	42	LD NMPT:64, RS:1, MBA:IN+2;
000A	0460		
000B	03EA		
000C	920E	43	FFT NMBT:64, RS:1;
000D	0070		

```

000E 0020
000F 0C00      44 ST      NMPT:64, RS:0, MBA:IN;
0010 0460
0011 03E8
0012 0400      45 LD      NMPT:64, RS:0, MBA:IN+4;
0013 0460
0014 03EC
0015 9206      46 FFT     NMBT:64, RS:0;
0016 0070
0017 0020
0018 0C08      47 ST      NMPT:64, RS:1, MBA:IN+2;
0019 0460
001A 03EA
001B 0408      48 LD      NMPT:64, RS:1, MBA:IN+6;
001C 0460
001D 03EE
001E 920E      49 FFT     NMBT:64, RS:1;
001F 0070
0020 0020
0021 0C00      50 ST      NMPT:64, RS:0, MBA:IN+4;
0022 0460
0023 03EC
0024 0C08      51 ST      NMPT:64, RS:1, MBA:IN+6;
0025 0460
0026 03EE

52
53 /*
54   After each FFT a four bit scale factor is sto
55   the sixteen bit scale register. After each st
56   register needs to be saved. These scale value
57   used to normalize the data before the second
58
59   Save the Scale Register.
60 */
61
0027 0818      62 STI     NMPT:1, RS:1, STR:5, MBA:SCALE;
0028 4150
0029 0077

63
64 /* Load Scale RAM with previously stored scale v
65
002A 0010      66 LDSM    NMPT:1, RS:0, MD:0, UP:1, MBA:SCALE;
002B A270
002C 0077

67
68 /*
69   Second stage of the 256 point FFT. The final
70   of the eight pass FFT will be performed. Befo
71   transformed the data will be normalized using
72   values from the first stage of the FFT.
73 */
74
75   DEFAULT MBS:64, MSS:64, FPS:2, LPS:1, SCLVL:4,
76   SCLBL:1, SB:1;
77
78 /*
79   With MBS:64 and MSS:64 a 64 point block of co

```



```

80      memory is addressed.
81      With FPS:2 and LPS:1 the data in each butterf
82      two points apart for the first pass, and adja
83      last.
84      */
85
002D  0400      86  LD      NMPT:64, RS:0, MBA:IN;
002E  D460
002F  03E8
0030  5C06      87  SCL     NMPT:64, RS:0;
0031  0281
0032  9206      88  FFT     NMBT:64, RS:0, RBA:0;
0033  0170
0034  0020
0035  0408      89  LD      NMPT:64, RS:1, MBA:IN+128;
0036  D460
0037  0468
0038  5C0E      90  SCL     NMPT:64, RS:1;
0039  0281
003A  0C00      91  ST      NMPT:64, RS:0, MBA:TEMP;
003B  D460
003C  07E9
003D  920E      92  FFT     NMBT:64, RS:1, RBA:56;
003E  0170
003F  0420
0040  0400      93  LD      NMPT:64, RS:0, MBA:IN+256;
0041  D460
0042  04E8
0043  5C06      94  SCL     NMPT:64, RS:0;
0044  0281
0045  0C08      95  ST      NMPT:64, RS:1, MBA:TEMP+128;
0046  D460
0047  0869
0048  9206      96  FFT     NMBT:64, RS:0, RBA:28;
0049  0170
004A  0220
004B  0408      97  LD      NMPT:64, RS:1, MBA:IN+384;
004C  D460
004D  0568
004E  5C0E      98  SCL     NMPT:64, RS:1;
004F  0281
0050  920E      99  FFT     NMBT:64, RS:1, RBA:84;
0051  0170
0052  0620

100
101  /* Save the Scale Register */
102
0053  0818      103  STI     NMPT:1, RS:1, STR:5, MBA:SCALE;
0054  4150
0055  0077

104
105  /* Load Scale RAM with previously stored scale v
106
0056  0010      107  LDSM    NMPT:1, RS:0, MD:0, UP:1, MBA:SCALE;
0057  A270
0058  0077

108

```

```

109  /*
110     The data must now be normalized
111     using the scale values from the second stage
112     and then bit-reversed.
113  */
114
115  DEFAULT LD.MBS:64, LD.MSS:64, ST.MBS:1, ST.MSS:4
116     ST.RV:1, SCLVL:1;
117
118  /*
119     For LD MBS:64 and MSS:64, so a 64 point block
120     contiguous memory is loaded.
121     For ST MBS:1 and MSS:4, so every fourth compl
122     stored.
123     With SCLVL:1 each point is scaled by the nibb
124     first word of Scale RAM which is pointed at b
125  */
126
0059  5C06 127  SCL  NMPT:64, RS:0, SCLBL:4;
005A  0211
005B  0C00 128  ST   NMPT:64, RS:0, MBA:IN+2;
005C  0560
005D  03EA
005E  5C0E 129  SCL  NMPT:64, RS:1, SCLBL:8;
005F  0219
0060  0400 130  LD   NMPT:64, RS:0, MBA:TEMP;
0061  D460
0062  07E9
0063  0C08 131  ST   NMPT:64, RS:1, MBA:IN+6;
0064  0560
0065  03EE
0066  5C06 132  SCL  NMPT:64, RS:0, SCLBL:1;
0067  0201
0068  0408 133  LD   NMPT:64, RS:1, MBA:TEMP+128;
0069  D460
006A  0869
006B  5C0E 134  SCL  NMPT:64, RS:1, SCLBL:2;
006C  0209
006D  0C00 135  ST   NMPT:64, RS:0, MBA:IN;
006E  0560
006F  03E8
0070  0C08 136  ST   NMPT:64, RS:1, MBA:IN+4;
0071  0560
0072  03EC

137
138
0073  6818 139  NOP  NMPT:1, RS:1;
0074  6811 140  NOP  NMPT:1, RS:0, EI:1;
141     /* The second NOP will interrupt when t
142     has gone idle. Two NOP's in differ
143     sections will insure that the second
144     not overlap with previous instructio
145     NOP's are required for the B-steppin
146     VSP161, because it cannot be guarant
147     the chip is completely idle when the
148     interrupt occurs. This will be corre
149     the C-stepping. */

```

```

150
151
152
0075 C000 153 HLT;
0076 0000
154
155
156 /*
157 The following assembler pseudo-ops define th
158 spaces for the program.
159
160 SCALE is the address at which the scale regi
161 be saved.
162 MODEADD is the address at which the value to
163 mode register
164 up for the operation of the program is store
165 IN is the address at which the input data wi
166 stored. So this address will be known the l
167 counter of the assembler is set to 1000.
168 TEMP is an address where some of the interme
169 results will be stored.
170 The output will be stored at IN.
171 */
172
0077 0000 173 SCALE: dw 0;
0078 0001 174 LENGTH: dw 1;
175
0079 4070 176 MODEADD: dw 0x4070;
177
178 org 1000;
179
03E8 0000 180 IN: dw 0;
181
182 org $ + 1024;
183
07E9 0000 184 TEMP: dw 0;
185
186 end;

```

S Y M B O L	TYPE	SEG	VALUE/SIZE	AT
IN	LABEL	C	0x03E8	
LENGTH	LABEL	C	0x0078	
MODEADD	LABEL	C	0x0079	
SCALE	LABEL	C	0x0077	
TEMP	LABEL	C	0x07E9	

```

ASSEMBLY COMPLETE
CODE SIZE: 7CH, 124D
186 LINES READ
0 WARNINGS DETECTED
0 ERRORS DETECTED

```

LOC	OBJ	LINE	SOURCE
		1	/* Forward FCT kernel 16 point Makhouls Algorithm
		2	/* Assumed image data at location subimage*/
		3	/* reorder is a data buffer used for the purpose
		4	/* G.A.King 28/2/90 */
		5	/* single row processing of a 16X16 subimage*/
		6	/* host to provide loop construct*/
		7	
		8	DEFAULT MDF:2;
		9	equ SUBIMAGE=1000,REORDER=1200,RESULT=1400;
		10	
		11	/* Load the 16 point array into VSP internal mem
		12	/* even points first*/
		13	
0000	0080	14	LD NMPT:8,MBS:1,MSS:2,MBA:SUBIMAGE;
0001	0040		
0002	03E8		
		15	
		16	/* Store consecutively*/
		17	
0003	0880	18	ST NMPT:8,MBS:8,MSS:8,MBA:REORDER;
0004	6840		
0005	04B0		
		19	
		20	/* Read in the odd points and reverse order*/
		21	
0006	0080	22	LD NMPT:8,MBS:1,MSS:2,MBA:SUBIMAGE+1;
0007	0040		
0008	03E9		
0009	0880	23	STB NMPT:8,MBS:8,MSS:8,MBAB:REORDER+15;
000A	6848		
000B	FB40		
		24	
		25	/* Perform outstanding bit reversal to comply*/
		26	/* Sande-Tukey FFT algorithm*/
		27	
000C	0100	28	LD NMPT:16,MBS:16,MSS:16,RV:1,ZR:1,MBA:REORDER;
000D	8D50		
000E	04B0		
		29	
		30	/* Take FFT and apply exponential vector*/
		31	
000F	908E	32	FFT NMBT:16,R:1,FPS:1,LPS:8,FSIZ:16,I:0,RS:1;
0010	8198		
0011	0009		
0012	510E	33	DEMO NMPT:16,ADF:3,RBA:0,RDA:496,VSIZ:32,RS:1;
0013	DCC8		
0014	001B		
		34	
		35	/* Send output to results array*/
		36	
0015	0900	37	ST NMPT:16,MBS:16,MSS:16,RS:0,MBA:RESULT;
0016	8C40		

```

0017 0578
          38
0018 C000 39 HLT;
0019 0000
          40

```

S Y M B O L	TYPE	SEG	VALUE/SIZE	AT
REORDER	CONSTANT	C	0x04B0	
RESULT	CONSTANT	C	0x0578	
SUBIMAGE	CONSTANT	C	0x03E8	

```

ASSEMBLY COMPLETE
CODE SIZE: 1AH, 26D
40 LINES READ
0 WARNINGS DETECTED
0 ERRORS DETECTED

```

LOC	OBJ	LINE	SOURCE
		1	/* Vector multiply-Assumes spatial domain refere
		2	
		3	/* describing a graphics entity are complex numb
		4	
		5	/* transformed into the frequency domain by FFT-
		6	
		7	/* constant will produce a scaling so that when
		8	
		9	/* is increased or decreased in size-Multiplying
		10	
		11	/* in the frequency domain will cause rotation i
		12	
		13	/* after IFFT */
		14	
		15	/* G.A.King VSP Research version 1.00 1/89 */
		16	
		17	/* SET UP INSTRUCTION FIELD DEFAULT VALUES */
		18	
		19	DEFAULT EI:0,INTRP:0,RV:0,ZR:0,ZP:0,MBS:8,MSS:8,
		20	
		21	/* OTHER DECLARATIONS */
		22	
		23	equ IN_ARRAY=0X200,OUT_ARRAY=0X1000,CON_VAL=0X40
		24	
0000	4870	25	SCRATCH:dw 0x4870;
		26	
		27	org 0
		28	
		29	/* SET UP RAM AS A SINGLE BLOCK */
		30	
		31	/* NUMBER OF POINTS 1, LOAD ONLY SCALE REG,NO UP
		32	
0000	0010	33	LDSM NMPT:1,MD:1,UP:0,MBA:SCRATCH;
0001	A170		
0002	0000		
		34	
		35	/* TRANSFER ARRAY FROM EXTERNAL TO INTERNAL RAM
		36	
		37	/* NUMBER OF POINTS 8,MEMORY DATA FORMAT COMPLEX
		38	
0003	0080	39	LD NMPT:8,MDF:3,MBA:IN_ARRAY;
0004	6860		
0005	0200		
		40	
		41	/* DO VECTOR MULTIPLY -COMPLEX VECTOR BY A CONST
		42	
		43	/* NUMBER OF POINTS 8,ARITH DATA FORMAT- ALL SAV
		44	
		45	/* OPERAND IN EXTERNAL MEMORY */
		46	
0006	1886	47	MLTR NMPT:8,ADF:3,CN:1,MBA:CON_VAL;
0007	6850		

```

0008 0400
48
49 /* STORE RESULTS BACK INTO EXTERNAL MEMORY */
50
51 /*NUMBER OF POINTS 8, MEM DATA FORMAT COMPLEX */
52
0009 0880
000A 6860
000B 1000
53 ST NMPT:8,MDF:3,MBA:OUT_ARRAY;

54
000C C000
000D 0000
55 HLT;

56
57 end;
58
59
60

```

S Y M B O L	T Y P E	S E G	V A L U E / S I Z E	A T
CON_VAL	CONSTANT	C	0x0400	
IN_ARRAY	CONSTANT	C	0x0200	
OUT_ARRAY	CONSTANT	C	0x1000	
SCRATCH	LABEL	C	0x0000	

```

ASSEMBLY COMPLETE
CODE SIZE: FH, 15D
60 LINES READ
0 WARNINGS DETECTED
0 ERRORS DETECTED

```

LOC	OBJ	LINE	SOURCE
		1	
		2	
		3	/*
		4	128 Point inverse FFT
		5	G.A.King 6/89
		6	
		7	Set value of Location Counter to 0. FFT progr
		8	loaded at this location.
		9	*/
		10	
		11	org 0;
		12	
		13	DEFAULT RS:0, EI:0, INTRP:0, ZP:0, AD:0, AS:0, F
		14	FSIZ:128, RV:0, R:0, MDF:3, ZR:0;
		15	
		16	/*
		17	inverse FFT set FFT.I:1.
		18	Set up the Mode Register so that the chip has
		19	section
		20	*/
0000	0010	21	
		22	LDSM NMPT:1, RS:0, MD:1, UP:0, MBA:MODEADD;
0001	A170		
0002	000F		
		23	
		24	/*
		25	128 point IFFT.
		26	
		27	The input data for this FFT should be stored
		28	1000 in the memory space of the VSP. The outp
		29	stored at location 1256.
		30	*/
		31	
		32	DEFAULT MBS:128, MSS:128, FPS:64, LPS:1;
		33	
		34	
0003	0000	35	LD NMPT:128, MBA:IN;
0004	F860		
0005	04E8		
0006	9406	36	FFT NMBT:128, R:0, I:1;
0007	0030		
0008	0022		
0009	0801	37	ST NMPT:128, RV:1, MBA:OUT, EI:1;
000A	F960		
000B	03EA		
000C	6811	38	NOP NMPT:1, RS:0, EI:1;
000D	C000	39	HLT;
000E	0000		
		40	
		41	
000F	4870	42	MODEADD: dw 0x4870;
		43	


```

      44  org 1256;
      45
04E8  0000  46  IN:  dw 0;
      47
      48  org $ - 255;
      49
03EA  0000  50  OUT: dw 0;
      51

```

S Y M B O L	TYPE	SEG	VALUE/SIZE	AT
IN	LABEL	C	0x04E8	
MODEADD	LABEL	C	0x000F	
OUT	LABEL	C	0x03EA	

```

ASSEMBLY COMPLETE
  CODE SIZE:  12H,   18D
51 LINES READ
0 WARNINGS DETECTED
0 ERRORS DETECTED

```

APPENDIX C

Vector Signal Processor Instruction Set

THE ZR34161 VSP INSTRUCTION SET

This chapter briefly describes the VSP instruction set. Each instruction is covered individually in one of the sections of this chapter. In the instruction descriptions, fixed fields are shown as 0 or 1, variable fields are labeled with the appropriate instruction field name, and DON'T CARE bits are left blank. Each word of multiple-word instructions is numbered from 0 to 2 when all three words are used. The bits in each word are numbered from 0 (rightmost/least significant bit) to 15 (leftmost/most significant bit).

Fields that are common to a majority of the instructions are listed in Section 3.1. Fields that are unique to a specific instruction are described in the same section as the instruction.

Note: All instructions which affect the accumulators clear the old accumulate values at start of instruction execution.

For more detailed information about the VSP instruction set, refer to the publications listed in the preface.

COMMON INSTRUCTION FIELDS

This section lists instruction fields that are common to a large number of instructions. The fields are listed alphabetically for quick reference. You should refer to this section whenever these fields are included in instruction descriptions throughout this manual.

Arithmetic Data Format

This field specifies which part of the operation result is stored in internal RAM. Valid values are:

- 0 No change; the result goes only to the accumulators.
- 1 Imaginary part only stored.
- 2 Real part only stored.
- 3 Complete result is stored.

Constant

This field specifies whether a constant or a vector operand from external memory is to be combined with the complex vector in internal RAM. Valid values are:

- 0 External memory contains a vector operand.
- 1 External memory contains a constant operand.

Enable Interrupt

EI specifies whether an interrupt is to be generated. Valid values are:

- 0 No interrupt is generated; only the status bit will be set.
- 1 An interrupt is generated at the end of instruction execution.

Reverse data ordering

RV specifies whether data is to be bit-reversed after being loaded into VSP internal RAM. Valid values are:

- 0 Data in normal order.
- 1 Bit-reverse order one level.
- 2 Data within blocks of size MBS in normal order, blocks in bit-reverse order.
- 3 Data within blocks of size MBS in bit-reverse order, blocks in normal order.

Note: If RV = 01 or 10, NMPT must be a power of two. If RV = 10 or 11, MBS is used for both memory segmentation and reversing – use caution. If RV is used for reversing only, set MSS = MBS - 1 or MBS >= NMPT.

Shift

Valid values are:

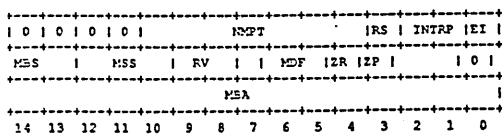
- 0 Result is not right-shifted.
- 1 Result is right-shifted one bit (scaled down by 2) to avoid overflow.

MEMORY INSTRUCTIONS

Memory instructions move data between external VSP memory and internal VSP memory or registers. All memory instructions are three words in length. The VSP memory instructions are:

- LD (Load)
- LDSM (Load Scale/Mode Register)
- ST (Store)
- STI (Store Information)
- STB (Store Backward)

LD (Load) moves data existing in external memory to internal VSP RAM. LD is a three-word instruction.



Following fields are defined in Section 3.1: NMPT, RS, EI, MBS, MSS, RV, MDF,

Following fields are unique to the LD instruction:

RP Interpolation.

MBAMemory Base Address

This field defines the starting address of the data in external memory. Valid values are:

Any 16-bit address between 0 and 0xFFFF (0 and 65535).

MBSMemory Block Size

The number of real, imaginary or complex data points to be loaded before a skip occurs. Also see RV and MSS below. MBS must be a power of two. Valid values are:

1, 2, 4, 8, 16, 32, 64 and 128.

MDFMemory Data Format for accessing VSP external memory

Valid values are:

- 0 Not used.
- 1 Imaginary only.
- 2 Real only.
- 3 Complex; first part real, second part imaginary.

MSS Memory Step Size

MSS defines the number of points specified in MBS plus the number of points to be skipped. MSS must be a power of two. See MBS above and RV below. Valid values are:

2, 4, 8, 16, 32, 64, 128 and 256.

NMPT Number of Points

NMPT defines the number of points (samples) of real, imaginary or complex data. Valid NMPT values are:

Any integer between 1 and 128.

RS RAM Section number

This parameter is used with the NMS parameter in the mode register to execute arithmetic and IO instructions concurrently. Valid values are:

- 0 Section 0; internal memory addresses 0 to 63 when NMS = 0.
- 0 Section 0; internal memory addresses 0 to 127 when NMS = 1.
- 1 Section 1; internal memory addresses 64 to 127 when NMS = 0.

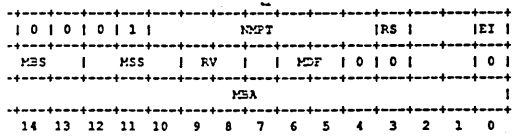
Note: RS can be 1 only if NMS=0 (specifying two RAM sections). If an ALU instruction operates with RS=0, a memory instruction with RS=1 can operate in parallel with the ALU instruction (or vice versa).

following fields are defined in Section 3.1: NMPT, RS, EI and MBA.

following fields are unique to the LDSM instruction:

- Update.**
Active only if MD = 0.
- 0 No update, scale register pointer not reset.
 - 1 Old maximum scale register updated from the current scale register, scale register pointer reset.
- Mode.**
- 0 Only the scale registers and scale RAM are loaded.
 - 1 Only the mode register is loaded.

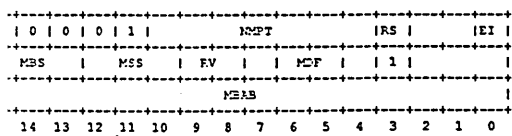
3 ST (Store)
moves data from the VSP internal RAM to external memory. ST is a three-word instruction.



fields in the ST instruction have the same definitions and use as in the LD instruction. If descriptions are contained in Section 3.1.

4 STB (Store Backward)
moves data from the internal VSP RAM to external memory in a manner similar to ST instruction. However, with STB the memory base address is decremented, not incremented as with ST. In other respects STB is similar to ST.

is a three-word instruction.



following fields are defined in Section 3.1: NMPT, RS, EI, MBS, MSS, RV and F.

following fields are unique to the STB instruction:

411-M-1.3-0687 3-5

The number of zeros to be added after each data point read from external memory. Valid values are:

- 0 No zeros added.
- 1 One zero added (NMPT must be even).
- 2 Two zeros added (NMPT must be divisible by three).
- 3 Three zeros added (NMPT must be divisible by four).

When using the INTRP parameter, NMPT includes the zeros to be added by INTRP. If zero padding is specified (ZP=1), the constraints on NMPT above do not apply. If the data is complex, a real zero and an imaginary zero are added for each zero added by INTRP.

ZR Zero Fill.
This field is used to zero that portion of internal memory masked off the MDF field. One of the two MDF bits must be zero for ZR to have any effect.

- 0 Internal memory unaffected.
- 1 Real part filled with zeros (with MDF=01).
- 1 Imaginary part filled with zeros (with MDF=10).

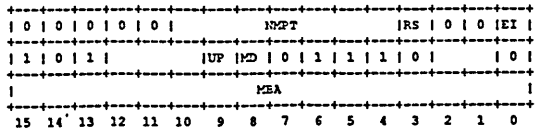
If MDF = 11, ZR must be zero.

ZP Zero Padding.
This field allows the VSP to read a vector from external memory and pad zeros onto the end of the vector. When ZP=1, NMPT must be equal to the length of the external vector plus the number of zeros to pad. Valid values are:

- 0 No zero padding to end of vector.
- 1 NMPT/2 or (NMPT + 1)/2 points from memory, the rest zeros.

3.2.2 LDSM (Load Scale/Mode Registers)
LDSM moves data from external memory to the VSP's 64-nibble scale RAM or to the mode register, as determined by the MD bit in the instruction. It resets the maximum scale register, resets the pointer to the scale register, and updates the old maximum scale register if MD=0 and UP=1. This ensures that the next scale factor will be written to the least significant nibble and causes maximum scale accumulation to be restarted.

LDSM is a three-word instruction.



3-4 ZR63411-M-1.3-0687

Arrangement of the list of registers above. Valid values are:

- 0 Registers 2 and 3 are interchanged.
- 1 The order above stands unchanged.

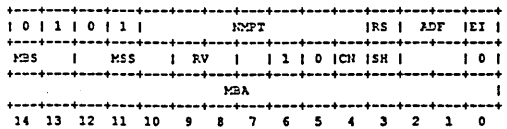
ALU/EXTERNAL MEMORY INSTRUCTIONS

section covers the four ALU instructions that operate on two data vectors. One of vectors must already exist in the internal VSP memory, and the other must exist in external memory. When these instructions are used, it is not possible for concurrent and I/O operations to be performed because these instructions require the use of the BTU and the EU. All instructions in this section are three words in length:

- ADDR (Vector Add Real)
- ADDC (Vector Add Complex)
- MLTR (Vector Multiply Real Accumulate)
- MLTC (Vector Multiply Complex Accumulate)

ADDR (Vector Add Real)
R adds a real vector in external memory to both the real and imaginary parts of a complex vector in internal RAM and stores the result in internal RAM. External memory remains unchanged.

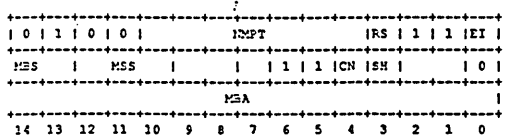
R is a three-word instruction.



fields contained in the ADDR instruction are defined in Section 3.1.

ADDC (Vector Add Complex)
C adds a complex vector in external memory to a complex vector in internal RAM adding real parts to real parts and imaginary parts to imaginary parts. The sum is stored in internal RAM. External memory remains unchanged.

C is a three-word instruction.



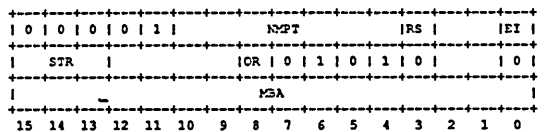
fields contained in the ADC instruction are defined in Section 3.1.

411-M-1.3-0687 3-7

MBAB Memory Base Address Backward.
The base address to use when storing backward. This address will store the imaginary part of the first data point.

3.2.5 STI (Store Information Registers)
STI moves the contents of specified information registers within the VSP to external memory.

STI is a three-word instruction.



The following fields are defined in Section 3.1: RS, EI and MBA.

The following fields are unique to the STI instruction:

NMPT Number of information registers to be read.
Note that the range is different from the one used in earlier instructions. Values can be 1 through 8 decimal.

STR Starting Register.
The place in the following list where the count of NMPT registers starts.

Register Number	Maximum Value of NMPT
1 Real Accumulator, LSB	8
2 Real Accumulator, MSB	7
3 Imaginary Accumulator, LSB	6
4 Imaginary Accumulator, MSB	5
5 Scale Register	4
6 Maximum Scale Register	3
7 Status Register	2
8 Next Fetch Address	1

There is an implied relation between NMPT (which defines the number of registers to store) and STR (which defines the register with which to begin the storage). This relationship is shown in the table above. For instance, if storage begins with register number five (scale register), the maximum number of registers that can be stored is four.

OR Order.

3-6 ZR63411-M-1.3-0687

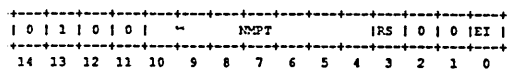
instructions covered in this section are:

- ACCR (Accumulate Real)
- ACCI (Accumulate Imaginary)
- ABS (Absolute Value)
- CMLT (Cross Multiply)
- CMCN (Complex Conjugate)
- MGSQ (Magnitude Square)
- DEMO (Demodulate)
- MODLT (Modulate)
- SCL (Scale)
- SCLT (Scale Literal)
- FFT (Fast Fourier Transform)

1 ACCR (Accumulate Real)

R accumulates the real part of the internal vector and stores the result in the real accumulator. Internal memory is not changed.

R is a one-word instruction.

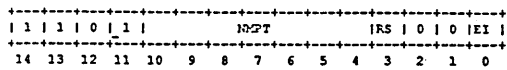


Fields contained in the ACCR instruction are defined in Section 3.1.

2 ACCI (Accumulate Imaginary)

I accumulates the imaginary part of the internal vector and stores the result in the imaginary accumulator. Internal memory is not changed.

I is a one-word instruction.

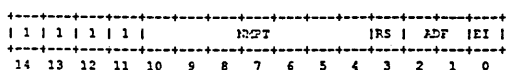


Fields contained in the ACCI instruction are defined in Section 3.1.

3 ABS (Absolute Value)

causes selected parts of the internal vector to be replaced by their absolute values. Specifies whether only the real part, only the imaginary part, or both parts will be affected.

ABS is a one-word instruction.

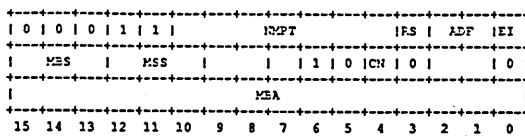


Fields contained in the ABS instruction are defined in Section 3.1.

3.3.3 MLTR (Vector Multiply Real Accumulate)

MLTR multiplies a complex vector in internal RAM by a real vector in external memory. External memory remains unchanged. The product is stored in internal RAM and added to the values in the real and imaginary accumulators.

MLTR is a three-word instruction.

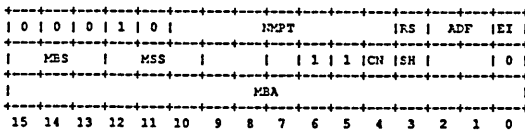


All fields contained in the MLTR instruction are defined in Section 3.1.

3.3.4 MLTC (Vector Multiply Complex Accumulate)

MLTC multiplies a complex vector in internal RAM by a complex vector in external memory. External memory remains unchanged. The product is stored in internal RAM and the sum of the products is stored in the real and imaginary accumulators.

MLTC is a three-word instruction.



All fields contained in the MLTC instruction are defined in Section 3.1.

3.4 INTERNAL ALU INSTRUCTIONS

There are eleven instructions that carry out arithmetic operations within the VSP using its internal registers and memory. Because the BIU is not used when these instructions are executed, they may be executed concurrently with memory LD/ST instructions if different RAM sections are being used by the ALU and memory instructions. Instructions in this section vary in length from one to three words.

O is a three-word instruction.



Following fields are unique to the DEMO instruction:

ROM Base Address

This parameter defines the starting angle of the cosine vector. It is specified as 10 times the desired value in degrees; for example, to get a starting angle of 45°, set RBA=450. Allowed values of RBA range between 0 and 3600, corresponding to 0° - 360°. Since the internal lookup table includes only 1024 distinct angles over the range 0° - 360°, not all values of RBA between 0 and 3600 are allowed. To ensure that the specified value corresponds exactly to one of the allowed angles, choose RBA to be one of the values in the set

$i * 3600 / 1024$, where $0 \leq i < 1024$.

ROM Decrement Address

This address is used to define the incremental angles for successive cosine coefficients, thereby determining the amount of frequency translation. The DEMO instruction will automatically generate all quadrants of the sinusoid, even though the ROM contains only the first quadrant values. Like RBA, this parameter is specified as 10 times the desired value in degrees. To make the DEMO perform a downconversion, RDA should be in the range $0 \leq RDA \leq 1800$. Again, not all angles in this range are allowed due to the finite resolution of the cosine table. Legal values belong to the set

$(i + 1) * 1800 / 512$, $0 \leq i \leq 511$.

Vector Size

Specifies the number of samples beginning with RBA to be addressed from the internal sine/cosine LUT, after which the LUT address rolls back to the RBA value.

Literal	Logical
000	-> 4 points
001	-> 8 points
010	-> 16 points
011	-> 32 points
100	-> 128 points

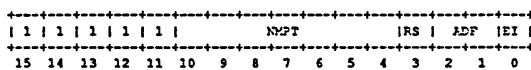
Following fields are defined in Section 3.1: NMPT, RS, ADF, EI and SH.

250

3.4.4 CMLT (Cross Multiply Accumulate)

CMLT multiplies the real part of the internal vector by the imaginary part. The result is stored in the real part. The sum of these products also goes to the real accumulator. ADF must be 0 or 2. If ADF is 0, only the real accumulator is changed.

CMLT is a one-word instruction.

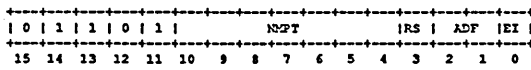


All fields contained in the CMLT instruction are defined in Section 3.1.

3.4.5 CMCN (Complex Conjugate)

CMCN replaces the complex internal vector with its complex conjugate. ADF must be 3.

CMCN is a one-word instruction. CMCN may be used as a NOP instruction by setting ADF=0 and NMPT=1. See the description of the NOP instruction in Section 3.5.3 for more details.

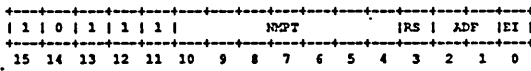


All fields contained in the CMCN instruction are defined in Section 3.1.

3.4.6 MGSQ (Magnitude Square Accumulate)

MGSQ calculates the square of the magnitude of the internal vector. The result is scaled down by two to prevent overflow and is written into the real part of the internal memory. The sum of the magnitude square elements is stored in the real accumulator. ADF must be 0 or 2. If ADF is 0, only the real accumulator is updated.

MGSQ is a one-word instruction.

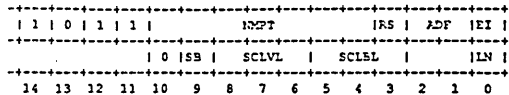


All fields contained in the MGSQ instruction are defined in Section 3.1.

3.4.7 DEMO (Demodulate)

DEMO multiplies a complex vector in internal memory by a series of complex coefficients generated from a lookup table which contains 256 cosine values ranging from 0° to 90°. Thus, this instruction corresponds to a heterodyning operation in a product demodulator. After multiplying each element of the specified cosine vector with the corresponding element in internal RAM, the products are placed in the internal RAM and the products are summed into the real and imaginary accumulators.

is a two-word instruction.



following fields are defined in Section 3.1: NMPT, RS, ADF and EI.

following fields are unique to the SCL instruction:

- Subtract.
 - Source of the content of the scale factor.
 - 0 Use the factor in scale RAM.
 - 1 Use the old maximum scale register value minus the scale RAM value.

BL Scale Block Length
 SCLBL is the number of points in VSP RAM to have the same scale factor.
 Valid values are:

- 1, 2, 4, 8, 16 and 32.

VL Scale Vector Length.

- Valid values are:

- 1, 2, 4, 8, 16, 32 and 64.

If the scale vector length is 1, SCLBL defines which of the first four Scale RAM nibbles to use.

Word Length of Instruction.

- 0 Three-word instruction.
- 1 Two-word instruction.

10 SCLT (Scale Literal)
 T scales the internal vector by a constant defined by the SHF parameter. SCLT is a word instruction.



following fields are defined in Section 3.1: NMPT, RS, ADF and EI.

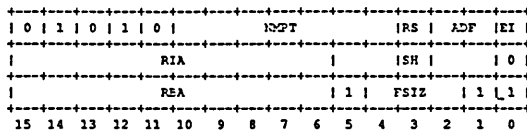
3.4.8 MODLT (Modulate)

This instruction is exactly analogous to DEMO, but the signal is translated up in frequency rather than down.

The parameters are the same as DEMO, except RDA has been replaced by RIA. RIA is the ROM Increment Address, and should be in the range $0 \leq RIA < 1800$. Legal values for RIA belong to the set

$$i * 1800 / 512, 0 \leq i \leq 511.$$

MODLT is a three-word instruction.



The following fields are defined in Section 3.1: NMPT, RS, ADF, EI and SH.

3.4.9 SCL (Scale)

SCL scales an internal complex vector down in magnitude by performing an integral number of right-shifts on the data samples of the vector operand. The number of bits of right-shifting performed is determined by elements of a scale vector (in VSP scale RAM).

For a full description of the VSP Scale RAM, refer to the VSP User's Manual.

Each nibble in the scale vector is a scaling factor from 0 to 15 representing the number of right-shifts -- divide-by-twos -- to apply to the elements of the internal vector.

The length of the scale vector is specified in the instruction setup. Also specified is the number of successive points in the internal vector to be scaled by the same scaling factor. When the scale vector length is shorter than the operand length, the scale vector starts over at its beginning to process the remainder of the operand. If the scale vector is only one nibble in length, the instruction allows specification of which nibble out of the first four in the Scale RAM is used in the instruction execution.

SCL also sums the scaled results into both the real and imaginary accumulators.

The content of each nibble of the scale vector is the number of right-shifts (divide-by-twos) performed on the operand vector:

- 0 No effect.
- 1 Divide by 2.
- 15 Divide by 32,768 (2^{**15}).

Note that R occurs a second time in the instruction parameters. Its second appearance is as the last parameter in the third word.

First Pass Separation.

The separation of the two sample points in the first butterfly pass. Valid values are:

- 1, 2, 4, 8, 16, 32 and 64.

Last Pass Separation.

The separation of the two sample points in the last butterfly pass. Valid values are:

- 1, 2, 4, 8, 16, 32 and 64.

ROM Base Address.

The offset address (representing angles from 0 up to, but not including, 180°) of the first coefficient to be used in the FFT in each pass. In each successive pass, RBA is right-shifted one bit. RBA is specified as 10 times the desired angle in degrees. For example, to specify an initial coefficient of 45°, set RBA=450.

FFT Size.

The number of points used in each FFT when the instruction is used to calculate more than one FFT. The total number of points is the product of the number of FFTs and FSIZ. Valid values are:

- 8, 16, 32, 64 and 128.

Automatic Scale.

Chooses the type of scaling to perform in conjunction with the FFT calculations.

- 0 Block floating operation. Scaling will be performed manually with the scale instruction.
- 1 Fixed divide-by-two each pass.

Note that it is possible to experience an overflow when AS is set to 1.

Inverse.

- 0 Forward FFT.
- 1 Inverse FFT.

CONTROL INSTRUCTIONS

ection describes the three instructions that control program flow in the VSP. They 1 length from one to three words. The three instructions are:

- IMPI (Jump Indirect)
- HLT (Halt)
- NOP (No Operation)

251

The following fields are unique to the SCLT instruction:

SHF Shift.

The number of right-shifts to apply to each vector element. Valid values are:

Any value from 0 to 15.

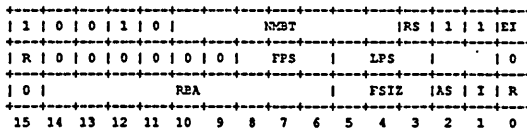
SB - Subtract.

Source of the content of the scale factor.

- 0 Use the SHF parameter as defined in the instruction for the number of right-shifts.
- 1 Use the old scale RAM value minus the SHF value as the number of right-shifts to perform.

3.4.11 FFT - The Fast Fourier Transform Instruction

FFT executes a Fast Fourier Transform or an Inverse Fast Fourier Transform on data stored internally in the VSP RAM. For a full description of this powerful and flexible instruction, refer to the VSP User Manual.



The following fields are defined in Section 3.1: RS and EI.

The following fields are unique to the FFT instruction:

NMBT Number of Butterflies Per Pass.

The number of butterflies (NMBT) is a value describing the number of data points to be operated on (twice the number of butterfly operations). Valid values are:

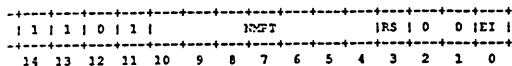
Any integer between 2 and 128.

R Reverse.

Order of the data in internal memory.

- 0 Normal order.
- 1 Bit-reversed order.

When R is set to 1, FPS should be 1, LPS should be greater than 1, and RBA should be 0.



fields in the NOP instruction are defined in Section 3.1.

3.5.1 JMPI (Jump Indirect)

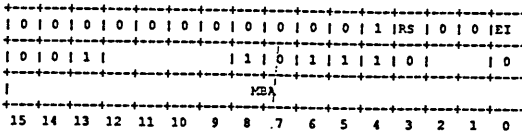
JMPI is the main program flow control instruction within the VSP. JMPI causes the VSP to load a new instruction fetch address located at the memory base address defined by MBA in the instruction word. JMPI is executed by the BIU.

Notice that MBA is not the address where the VSP begins fetching subsequent instructions. Rather, the data in location MBA is the address of the next instruction. For example, if location 15 holds the data 72, then the instruction

```
JMPI MBA:15;
```

would cause the next instruction to be fetched from location 72.

JMPI is a three-word instruction.

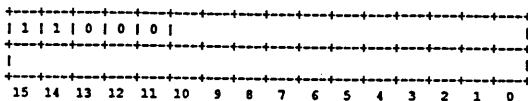


All the fields in the JMPI instruction are defined in Section 3.1.

3.5.2 HLT (Halt)

HLT stops the VSP bus-interface unit from fetching any more instructions. It is used as the last instruction in a program or when it is desired to halt instruction fetch. ALU instructions executing or queued in the instruction FIFO when a HLT instruction is executed are not affected. ALU instructions will complete and provide status to the host as defined in the particular ALU instruction. HLT has no meaning in the slave mode.

HLT is a two-word instruction, where all but the first five bits are DON'T CARE.

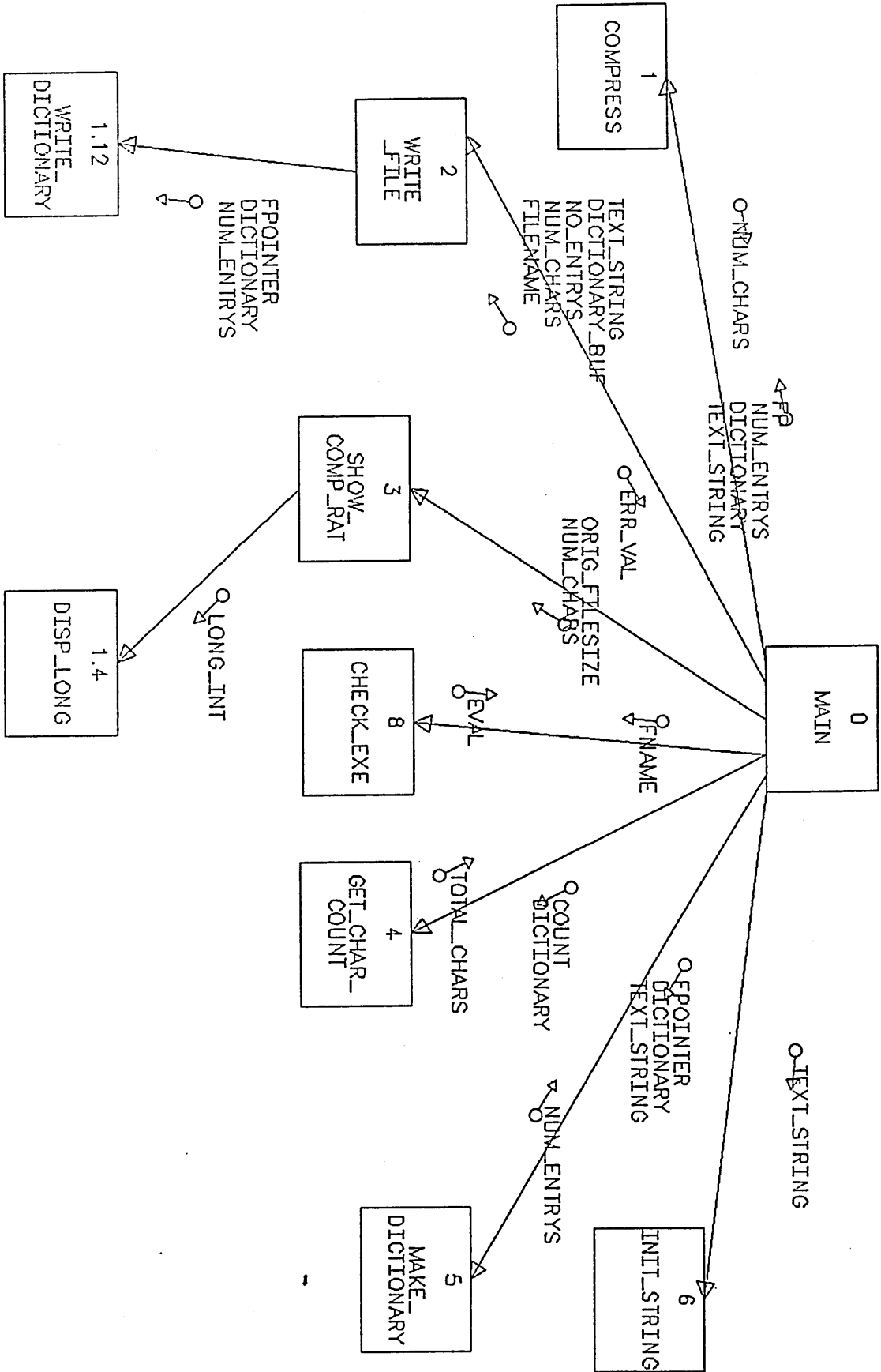


3.5.3 NOP (No Operation)

NOP is a one-word null instruction that has no effect on the execution of other instructions, nor on registers (except the status registers, which are updated) or memory. NOP is implemented as a CMCN instruction with ADF:0. Note that the execution time of the NOP instruction is a function of the NMPT parameter defined in the instruction. This allows variable-length NOP instructions for applications requiring predictable delays. It is sometimes useful to insert NOPs in a program to reserve space for later use or to time operations for real-time applications.

APPENDIX D

Morse Compression Program Documentation



```

INT ARGV
WORD_CHAR_COUNT
NUM_WORDS
CHAR *ARGV[]
LONG CHAR_COUNT
FILESIZE
UNSIGNED CHAR FAR *STRING
FILE *FSTR

STRUCT CHAR_DETAILS *
UNSIGNED CHAR SIZE
UNSIGNED CHAR CODE

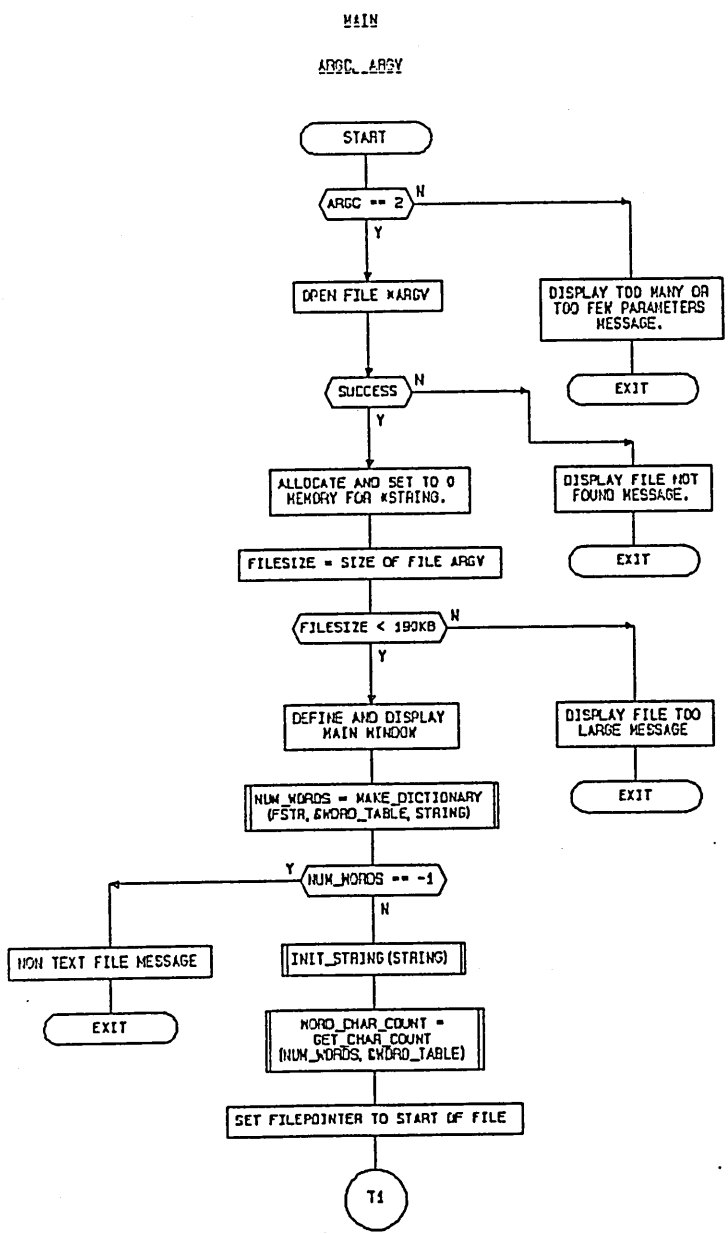
STRUCT WH *
INT LEFT, TOP, RIGHT, BOTTOM
INT FEAR, FURE, ALTREAR, ALTFURE

STRUCT HLEN *
CHAR WORD[9]
UNSIGNED CHAR CODE
INT NUM_SAVED
CHAR *HLEN

STRUCT DICT *
CHAR WORD[9]
UNSIGNED CHAR CODE
INT LENGTH

STRUCT DICT WORD_TABLE

```

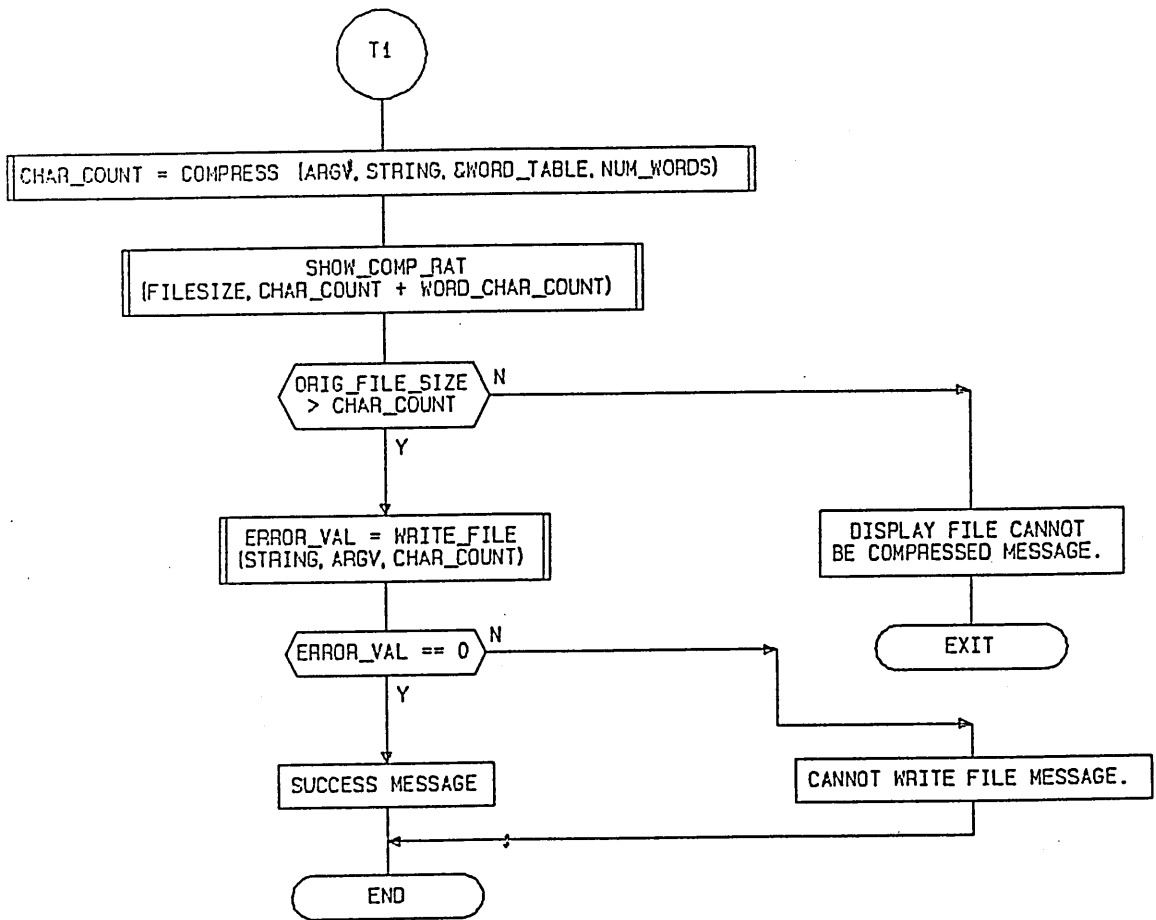


MAIN 0
 COMPRESSES THE FILE, WHOSE FILENAME IS PASSED TO THE PROGRAMME AS A COMMAND LINE PARAMETER. WRITES THE COMPRESSED FILE BACK TO THE ORIGINAL FILE IF ABLE TO COMPRESS THE TEXT IN THE ORIGINAL FILE.

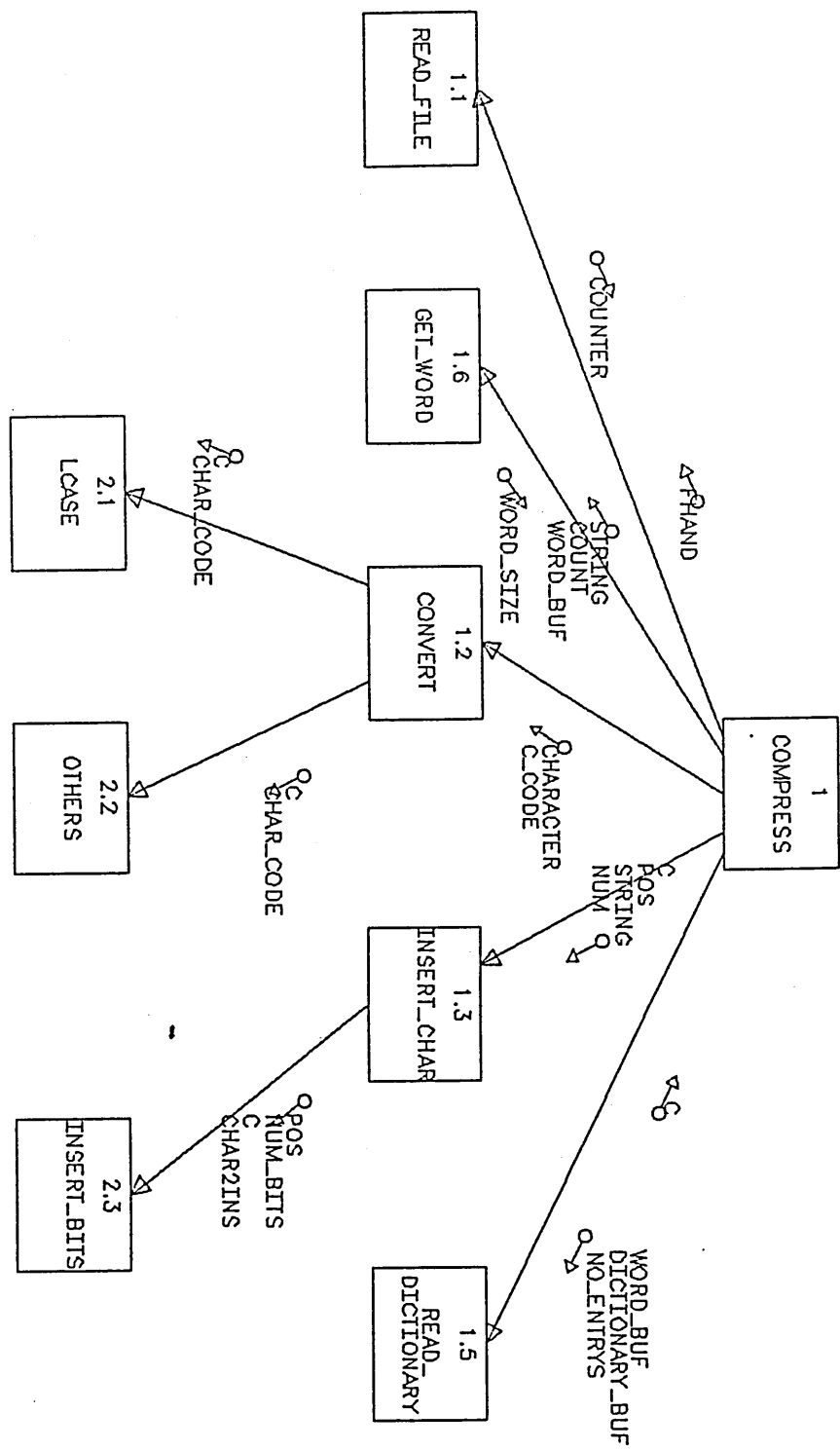
Kinds V1.B
 COMP5 VERSION 13
 05-Jun-90 08:30

MAIN

ARGC, ARGV



Kindra V1.B
Comp16 VERSION 13
06-Jun-90 08:31:



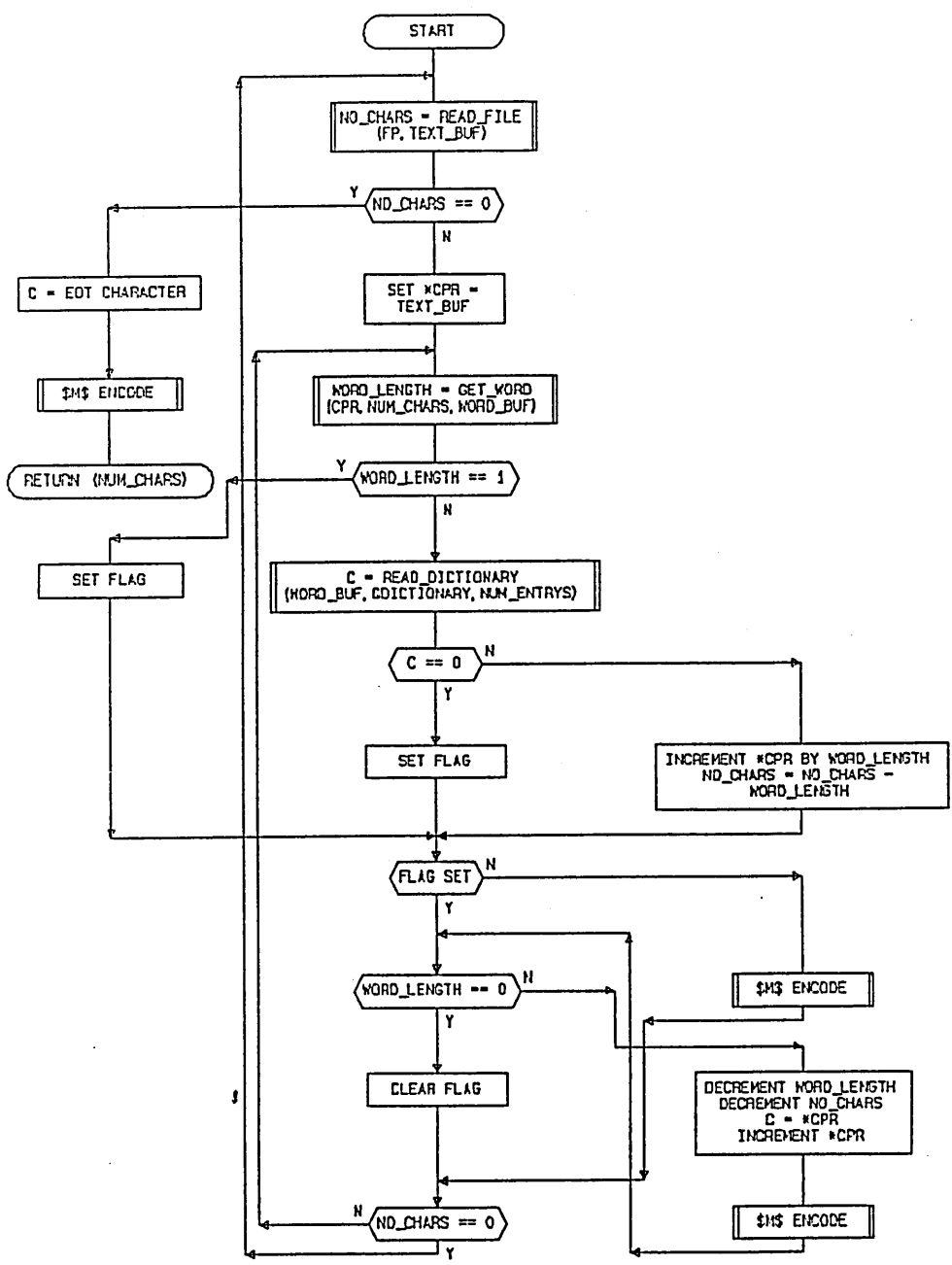
```

FILE FP
UNSIGNED CHAR FAR *TEXT_STRING
STRUCT DICT *DICTIONARY
INT NUM_ENTRIES
NO_CHARS
WORD_LENGTH
FLAG
UNSIGNED CHAR C
LONG NUM_CHARS = 1
CHAR *CPR
CHAR TEXT_BUF [256000]
CHAR WORD [9]

```

COMPRESS

FP, TEXT_STRING, DICTIONARY, NUM_ENTRIES



STRUCT DICT DECLARED ON MAIN	MODULE COMPRESS 1 COMPRESSES THE TEXT FILE GIVEN BY THE FILE POINTER FP. THE COMPRESSED TEXT IS STORED TO TEXT_STRING. RETURNS THE NUMBER OF CHARACTERS THAT HAVE BEEN STORED TO TEXT STRINGS.	Kindra V1.B comp17 VERSION 4 05-Jun-90 08:32
------------------------------	---	--

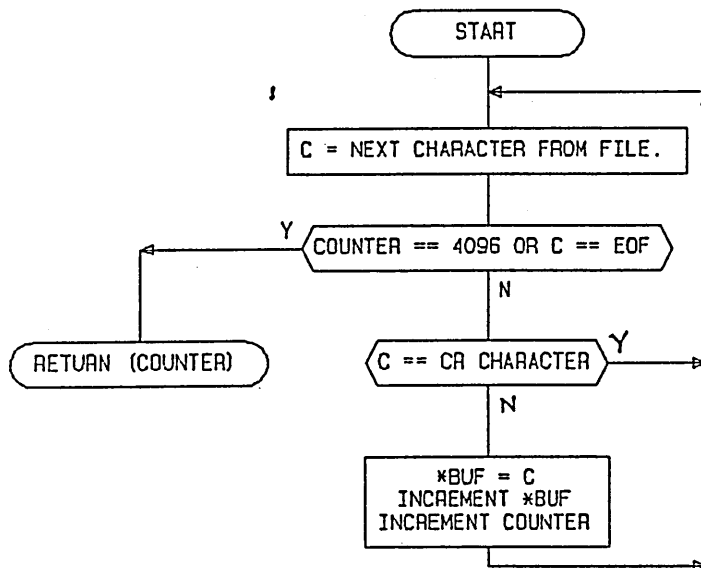
```

FILE *FHAND
UNSIGNED CHAR *BUF
INT COUNTER
UNSIGNED CHAR C

```

READ_FILE

FHAND_BUF



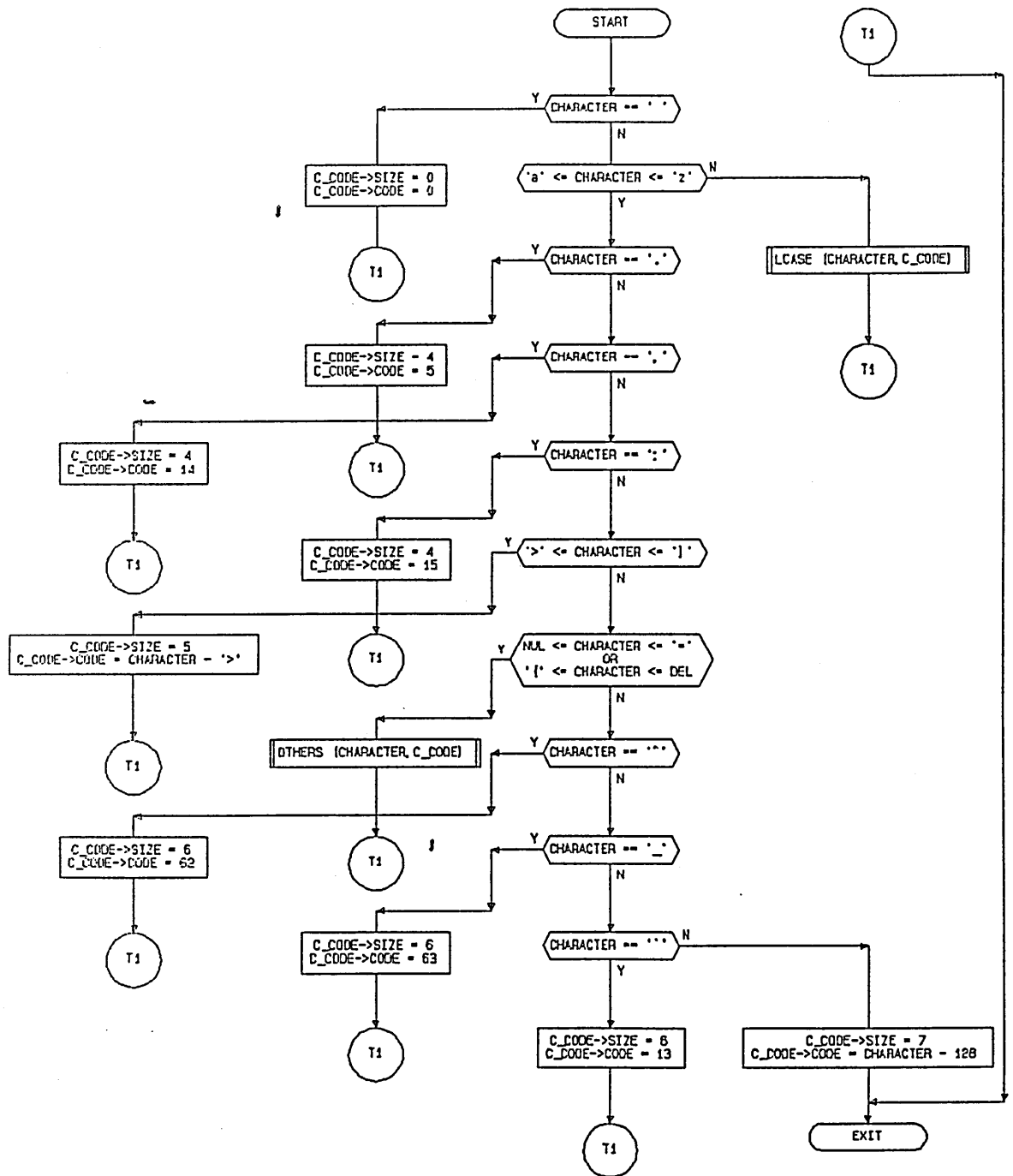
MODULE READ_FILE 1.1

READS CHARACTERS INTO A BUFFER. RETURNS THE NUMBER OF CHARACTERS THAT HAVE BEEN READ INTO THE BUFFER.

Kindra V1.B
 COMP8 VERSION 2
 09-May-90 11:13:

UNDEFINED CHAR CHARACTER
STRUCT CHAR_DETAILS C_CODE

CONVERT
CHARACTER_C_CODE



STRUCT CHAR_DETAILS DECLARED ON MAIN	MODULE CONVERT 1.2 CONVERTS THE CHARACTER'S ASCII VALUE INTO IT'S CODES. THE CODES ARE STORED TO THE STRUCTURE C_CODE. THESE CODES ARE :- THE NUMBER OF BITS IN THE CHARACTER'S CONVERTED CODE AND THE CHARACTER'S CONVERTED CODE.	Kindra V1.B CD#99 VERSION 2 09-May-90 12:29:
---	--	--

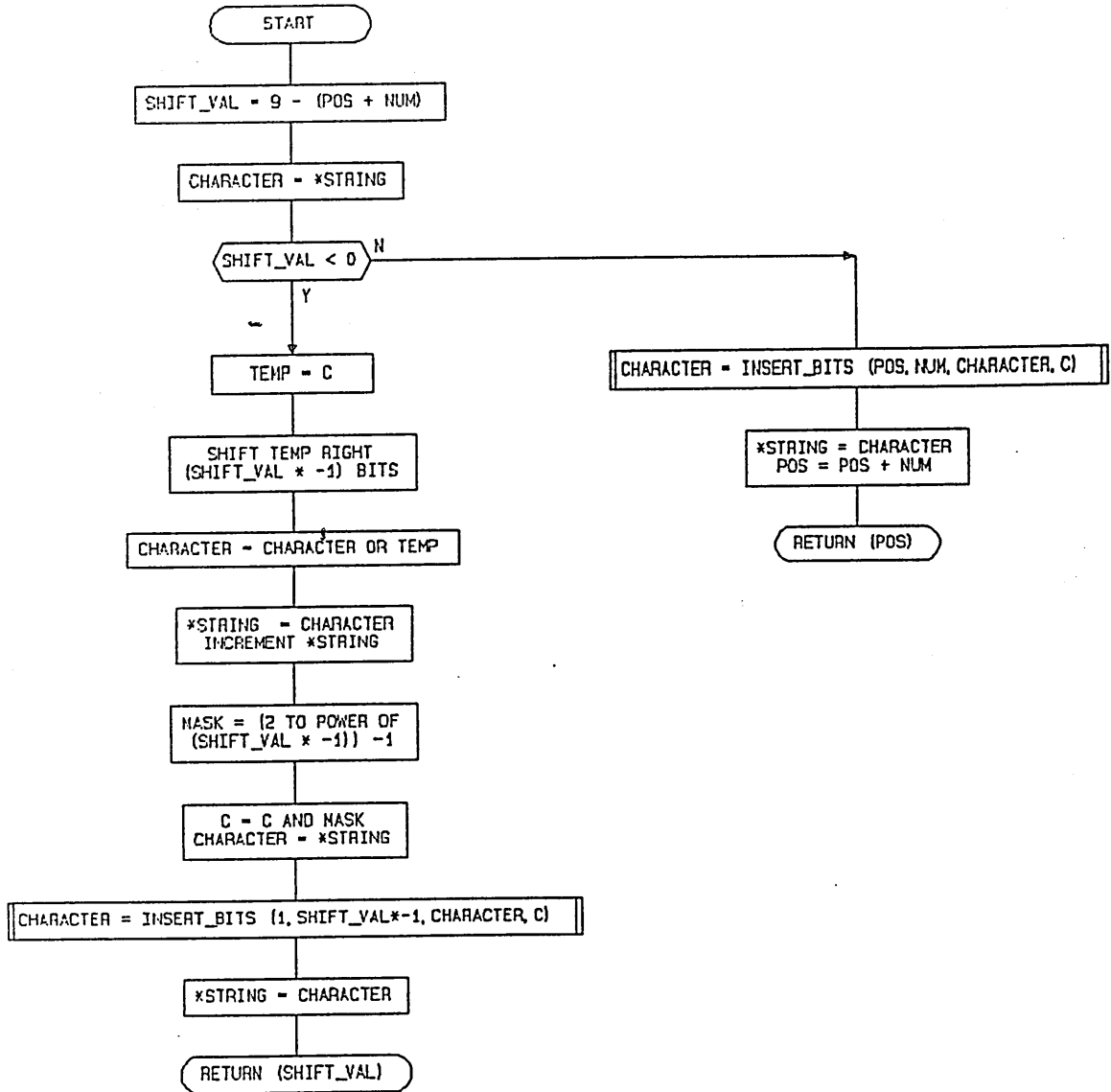
```

UNSIGNED CHAR C
NUM
TEMP
MASK
CHARACTER
UNSIGNED CHAR FAR *STRING
INT POS

```

INSERT_CHAR

C, POS, STRING, NUM



MODULE INSERT_CHAR

1.3

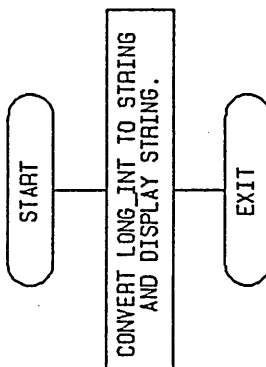
INSERTS THE CHARACTER C, INTO THE LOCATION POINTED TO BY STRING AT A BIT POSITION GIVEN BY POS. THE NUMBER OF BITS TO INSERT IS GIVEN BY NUM. RETURNS THE BIT POSITION THAT THE NEXT CHARACTER IS TO BE INSERTED AT. IF THE NUMBER OF BITS IS TOO GREAT TO FIT IN THE ONE LOCATION POINTED TO BY STRING, WILL INSERT THE SURPLUS INTO THE NEXT LOCATION POINTED TO BY STRING AND RETURN A NEGATIVE INT VALUE THAT REPRESENTS THE POSITION OF THE LAST BIT INSERTED.

Kindra V1.8
comp2 VERSION 2
08-May-90 12:42

LONG LONG_INT

DISP_LONG

LONG_INT



MODULE DISP_LONG 1.4

DISPLAYS A LONG INTEGER VALUE AT THE CURRENT CURSER POSITION AND IN THE CURRENT WINDOW.

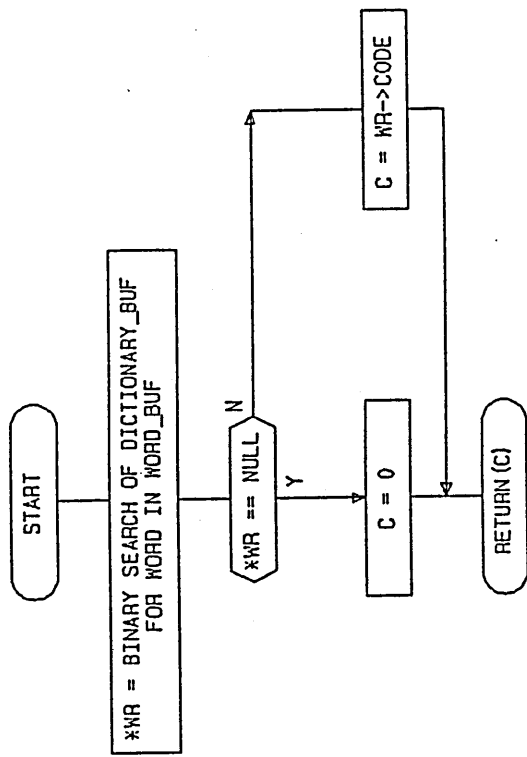
Kindra V1.B
COMP7 VERSION 2
09-May-90 10:43:

```

CHAR *WORD_BUF
STRUCT DICT *DICTIONARY_BUF
INT NO_ENTRIES
UNSIGNED CHAR C

```

READ_DICTIONARY
WORD_BUF, DICTIONARY_BUF, NO_ENTRIES



STRUCT DICT DECLARED ON MAIN	MODULE READ_DICTIONARY 1.5 LOOKS THROUGH DICTIONARY TO SEE IF WORD IN *WORD_BUF EXISTS IN THE DICTIONARY. IF IT DOES EXIST RETURNS THE WORD'S SINGLE CHARACTER CODE. ELSE RETURNS 0.	Kindra V1.B COMP17 25-May-90 10:53:
---------------------------------	---	---

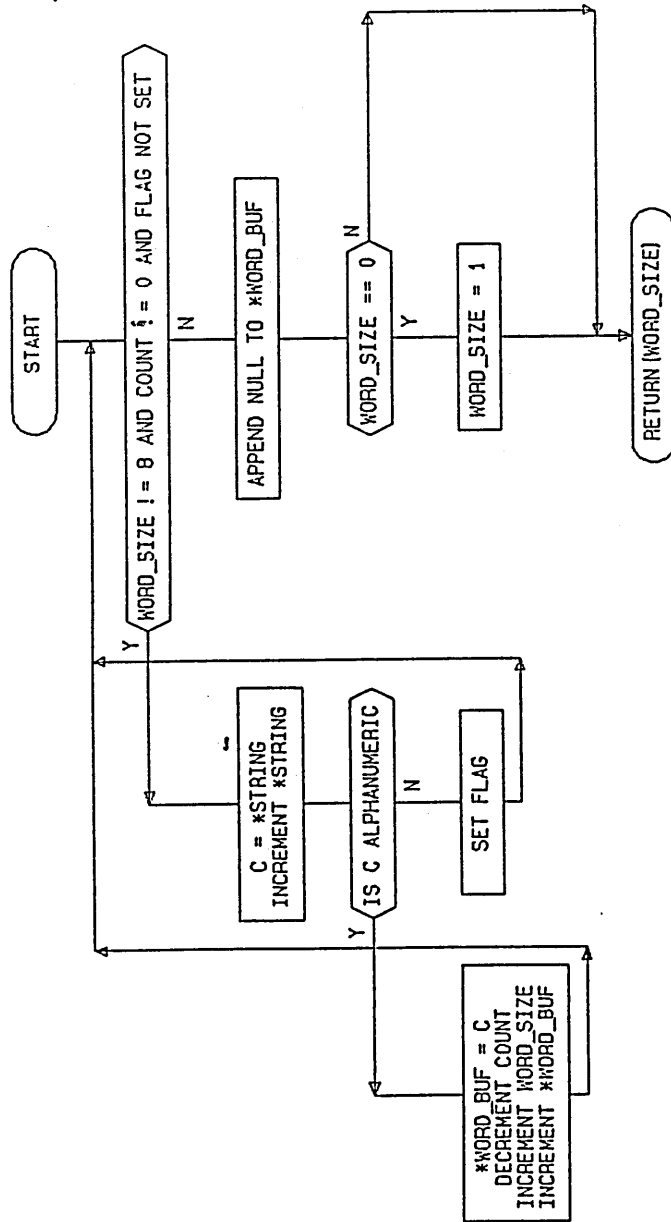
```

CHAR *STRING
INT COUNT
CHAR *WORD_BUF
CHAR C
INT FLAG
INT WORD_SIZE = 0

```

GET_WORD

SIBING_COUNT_WORD_BUF



MODULE GET_WORD 1.6

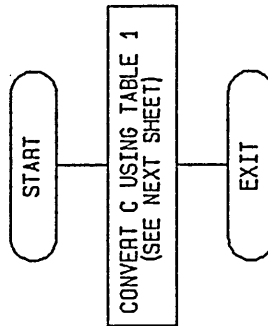
EXTRACTS A WORD FROM THE *STRING. A WORD IS A SEQUENCE OF 2 TO 8 ALPHANUMERIC CHARACTERS. KEEPS A COUNT OF HOW MANY CHARACTERS ARE LEFT IN STRING. IF NONE LEFT EXITS LOOP. RETURNS THE NUMBER OF CHARACTERS IN THE EXTRACTED WORD OR 1 IF *STRING DOES NOT POINT TO A WORD.

Kindra V1.8
comp23 VERSION 6
06-Jun-90 08:35:

UNSIGNED CHAR C
STRUCT CHAR_DETAILS CHAR_CODE

LCASE

C..CHAR_CODE



STRUCT CHAR_DETAILS
DECLARED ON MAIN

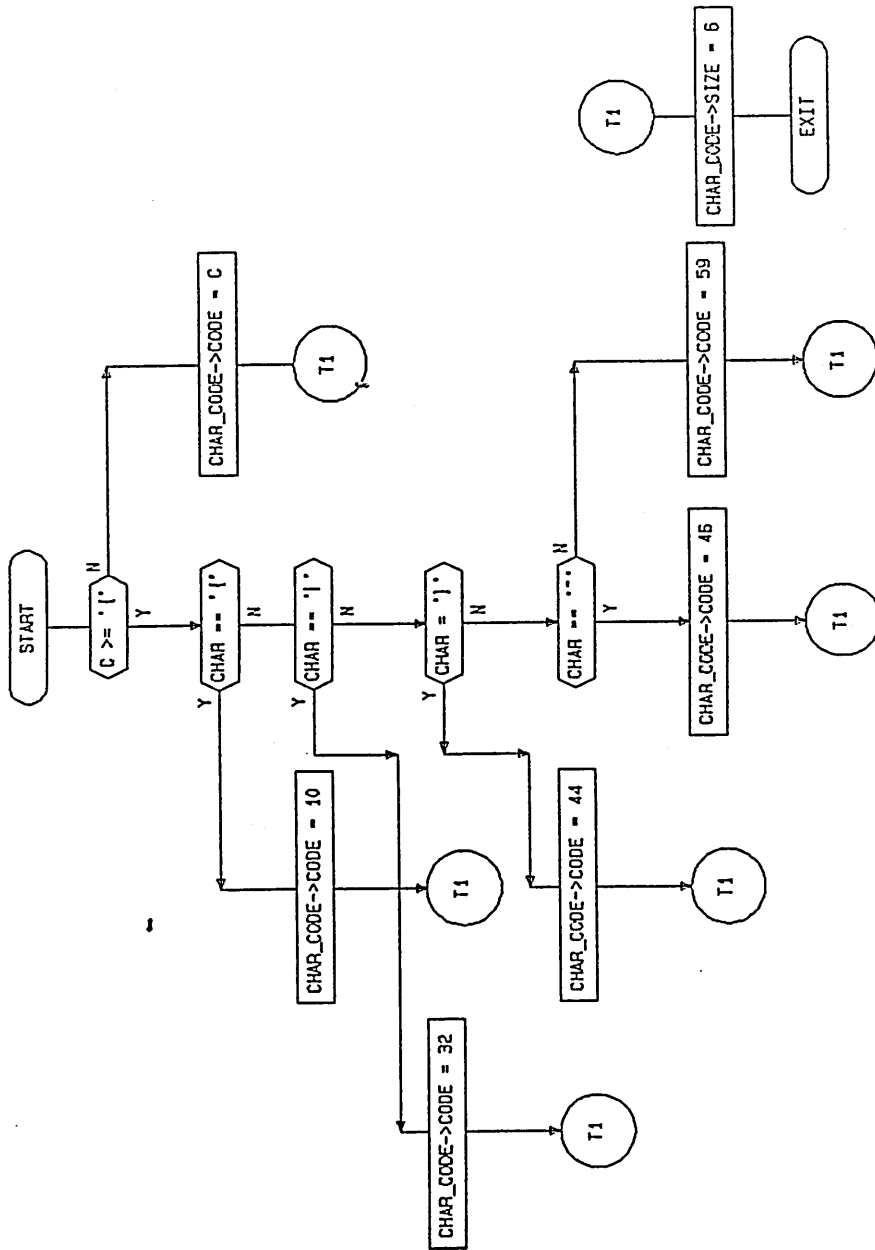
MODULE LCASE 2.1

RETURNS THE CHARACTER'S CONVERTED CODE AND SIZE FOR
LOWER CASE A TO Z.

Kindra V1.B
COMPI0 VERSION 2
09-May-90 12:45.

CHAR C
STRUCT CHAR_DETAILS CHAR_CODE

OTHERS
C.CHAR_CODE



STRUCT CHAR_DETAILS
DECLARED ON MAIN

MODULE OTHERS 2.2

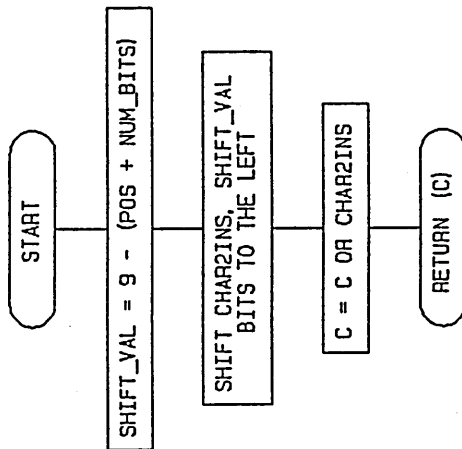
RETURNS THE CHARACTER'S COMPRESSED CODE AND COMPRESSED CODE
SIZE FOR CHARACTERS WITH A COMPRESSED CODE SIZE OF 6 BITS.

Kindra V1.8
COCOLL VERSION 6
02-JUN-99 05:29

INT POS
UNSIGNED
CHAR C
NUM_BITS
CHAR2INS

INSERT_BITS

POS..NUM_BITS..C..CHAR2INS



MODULE INSERT_BITS 2.3

INSERTS NUM_BITS (THIS IS THE NUMBER OF BITS IN CHAR2INS) INTO C AT A POSITION. POS BITS IN FROM MSBIT.

Kindra V1.B
COMP3 VERSION 2
03-MAY-90 09:24

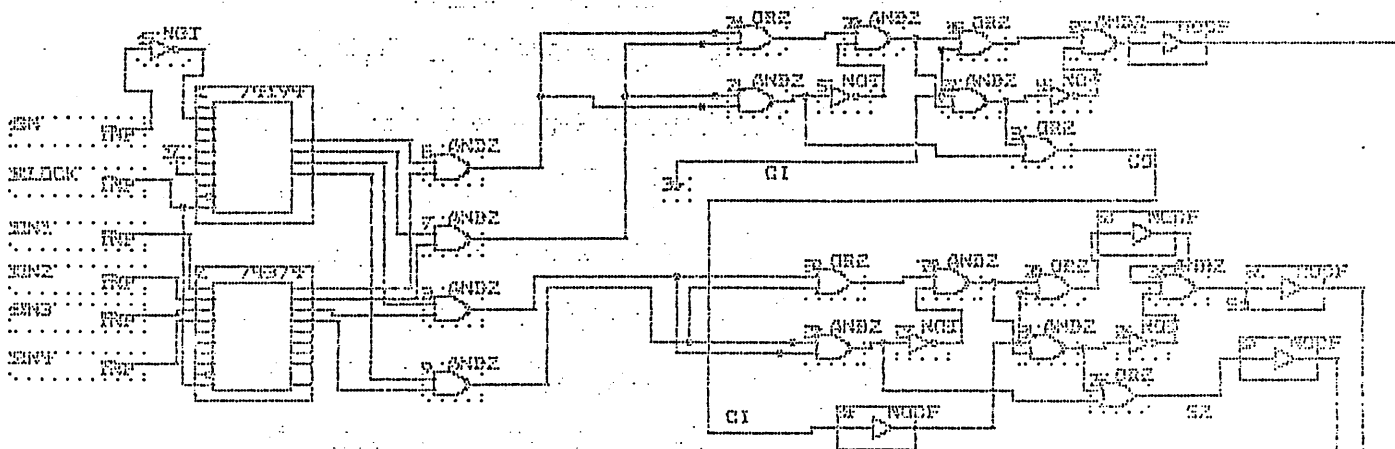
APPENDIX E

Design details

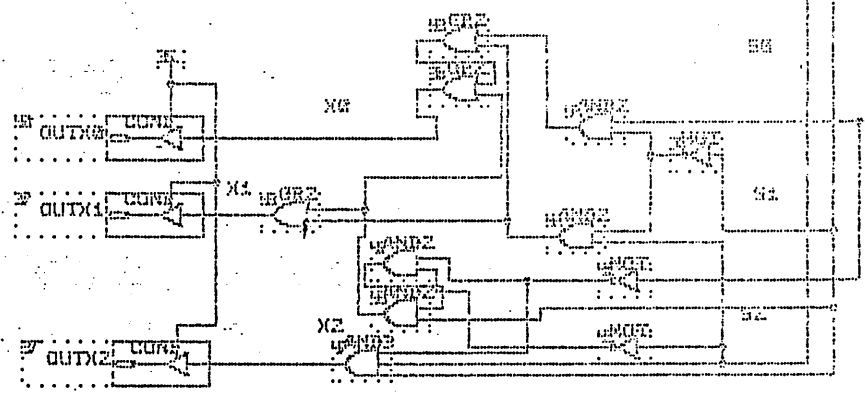
Matched Filter implementation

as

Altera EPLD 1200 series device



S.I.H.E. S.C.E.	
PRNG. MATCHED FILTER	
S.A.KINS	
EP1216	1 0/2 A
EP1217	1
UP	OFF



G.A.King
S.I.H.E. S.C.E.
May 1989
1.00

EP1210
Prog. Matched Filter
LogiCaps Schematic Capture Ver 1.6

OPTIONS: TURBO = ON, SECURITY = OFF

PART: EP1210

INPUTS:

hn4, hn3, hn2, hn1, clock, xn

OUTPUTS:

outx2, outx1, outx0

NETWORK:

74194(.....032029,VCC,....032045,..047039,..047037,..047035,..047033) % SYM 1 %
74374(..032060,..032062,..032064,..032066,.....032045,.....047066,..047064,..047062,..047060) % SYM 2 %
..165035 = OR(..158034,..158036) % SYM 3 %
..168025 = NOT(..158034) % SYM 4 %
..134025 = NOT(..158036) % SYM 5 %
..074038 = AND(..047033,..047060) % SYM 6 %
..074051 = AND(..047035,..047062) % SYM 7 %
..074064 = AND(..047037,..047064) % SYM 8 %
..074078 = AND(..047039,..047066) % SYM 9 %
..022031 = INP(xn) % SYM 10 %
..032045 = INP(clock) % SYM 11 %
..032060 = INP(hn1) % SYM 12 %
..032062 = INP(hn2) % SYM 13 %
..032064 = INP(hn3) % SYM 14 %
..032029 = NOT(..022031) % SYM 15 %
..155015 = OR(..148014,GND) % SYM 16 %
..032066 = INP(hn4) % SYM 18 %
..148014 = AND(..132013,..134025) % SYM 20 %
..158036 = AND(..074051,..074038) % SYM 21 %
..158034 = AND(GND,..148014) % SYM 22 %
..174015 = AND(..155015,..168025) % SYM 23 %
..132013 = OR(..074038,..074051) % SYM 24 %
..177080 = OR(..170079,..170081) % SYM 25 %
..180070 = NOT(..170079) % SYM 26 %
..146070 = NOT(..170081) % SYM 27 %
..167060 = OR(..160059,..160061) % SYM 28 %
..160059 = AND(..144058,..146070) % SYM 29 %
..170081 = AND(..074078,..074064) % SYM 30 %
..170079 = AND(..160061,..160059) % SYM 31 %
..186060 = AND(..179059,..180070) % SYM 32 %
..144058 = OR(..074064,..074078) % SYM 33 %
..187127 = NOT(s2) % SYM 36 %
outx2 = CONF(..114165,VCC) % SYM 37 %
..150115 = OR(..157114,..157116) % SYM 38 %
outx1 = CONF(..113138,VCC) % SYM 39 %
outx0 = CONF(..150115,VCC) % SYM 40 %
..176150 = NOT(..182150) % SYM 41 %
..176162 = NOT(..182162) % SYM 42 %
..113138 = OR(..157116,..131139) % SYM 43 %
..131139 = AND(..187127,..182162) % SYM 44 %
..171122 = AND(..182150,..187127) % SYM 45 %
..157116 = AND(..148155,s2) % SYM 46 %
..114165 = AND(..176150,..182162,s2) % SYM 47 %
..157114 = OR(..171122,..131139) % SYM 48 %
..148155 = AND(..176150,..176162) % SYM 49 %
..182162 = NOCF(..177080) % SYM 50 %
s2 = NOCF(..186060) % SYM 51 %
..182150 = NOCF(..174015) % SYM 52 %
..179059 = NOCF(..167060) % SYM 53 %
..160061 = NOCF(..165035) % SYM 54 %

END\$

OUTPUTS

Name	Pin	Resource	MCell	PTerms	FdBck Group	Clear	OE Group
outx0	10	CONF	26	3/ 8	1	-	VCC
outx1	11	CONF	25	2/ 6	1	-	VCC
outx2	8	CONF	28	1/ 4	1	-	VCC

BURIED REGISTERS

Name	Pin	Resource	MCell	PTerms	FdBck Group	Clear	OE Group
s2 (9)		COIF!	27	5/10	1	-	-
.0047033 (16)		NORF	20	1/12	7G	.0M001N1	-
.0047035 (19)		NORF	17	1/ 8	7G	.0M001N1	-
.0047037 (18)		NORF	18	1/ 8	7G	.0M001N1	-
.0047039 (17)		NORF	19	1/ 4	7G	.0M001N1	-
.0047060 (22)		NORF	11	1/ 8	8G	GND	-
.0047062 (23)		NORF	10	1/ 4	8G	GND	-
.0047064 -		NORF	16	1/ 8	5G	GND	-
.0047066 -		NORF	15	1/ 8	5G	GND	-
.0160061 (15)		COIF!	21	1/ 4	3	-	-
.0179059 (14)		COIF!	22	5/10	3	-	-
.0182150 (12)		COIF!	24	4/ 6	3	-	-
.0182162 (13)		COIF!	23	3/ 8	3	-	-

INPUTS

Name	Pin	Resource	MCell	PTerms	FdBck Group	Clear	OE Group
clock	1	CKR	-	-	-	-	-
hn1	5	INF	-	-	-	-	-
hn2	4	INF	-	-	-	-	-
hn3	3	INF	-	-	-	-	-
hn4	2	INF	-	-	-	-	-
xn	6	INF	-	-	-	-	-

UNUSED RESOURCES

Name	Pin	Resource	MCell	PTerms	FdBck Group	Clear	OE Group
-	21	MCELL	12	8	8G	GND	-
-	24	MCELL	9	12	8G	GND	-
-	25	MCELL	8	4	2	-	-
-	26	MCELL	7	10	2	-	-
-	27	MCELL	6	8	2	-	-
-	28	MCELL	5	6	2	-	-
-	29	MCELL	4	6	4	-	-
-	30	MCELL	3	8	4	-	-
-	31	MCELL	2	10	4	-	-
-	32	MCELL	1	4	4	-	-
-	-	MCELL	13	8	6G	-	-
-	-	MCELL	14	8	6G	-	-

PART UTILIZATION

/28 MacroCells (57%)
 / 5 Input Pins (100%)
 PTerms Used 27%

Macrocell Interconnection Cross Reference

MATCH.rpt

FEEDBACKS:

			M M	M M	M M M M	M M M M	M M M M	M M M M	
			1 1	1 1	1 1 1 2	2 2 2 2	2 2 2 2	2 2 2 2	
			0 1	5 6	7 8 9 0	1 2 3 4	5 6 7 8		
.0047862	. NORF	@M10->	* . . *		(23)
.0047860	. NORF	@M11->	* . . *		(22)
.0047866	. NORF	@M15->	* * * .		
.0047864	. NORF	@M16->	* * * .		
.0047835	. NORF	@M17-> # . .	* . . *		(19)
.0047837	. NORF	@M18-> # .	. * * .	. . * .		(18)
.0047839	. NORF	@M19-> * * .	. . * .		(17)
.0047833	. NORF	@M20-> # . .	* . . *		(16)
.0180601	. COIF	@M21->	x x	* * * .		(15)
.0179259	. COIF	@M22->	x x * .		(14)
.0182162	. COIF	@M23->	x x	* * . *		(13)
.0182150	. COIF	@M24->	x x	* * . *		(12)
outx1	CONF @M25->	x x		@11
outx0	CONF @M26->	x x		@10
2	COIF @M27->	x x	* * . *		(9)
outx2	CONF @M28->	x x		@8

INPUTS:

04	INP	@2 ->	. .	*
03	INP	@3 ->	. .	. *
02	INP	@4 ->	*
01	INP	@5 ->	. *
0	INP	@6 ->	*
				0 0 0 0	0 0 0 0
				0 0	0 0	0 0 0 0	0 0 0 0	u u 2 u	t t t
				4 4	4 4	4 4 4 4	6 7 8 8	x x	x
				7 7	7 7	7 7 7 7	0 9 2 2	1 0	2
				0 0	0 0	0 0 0 0	0 0 1 1		
				6 6	6 6	3 3 3 3	6 5 6 5		
				2 0	6 4	5 7 9 3	1 9 2 0		