VLSI ARCHITECTURES

FOR HIGH SPEED

FOURIER TRANSFORM PROCESSING

BY

IAIN ROSS MACTAGGART

A Thesis submitted to the Faculty of Science
of the University of Edinburgh, for the
degree of Doctor of Philosophy.

Department of Electrical Engineering

November 1984

Acknowledgements

I would like to thank Dr. Mervyn Jack, my supervisor for the encouragement and guidance he has offered throughout the three years of this research and also Dr. James Dripps for his interest and support in this work as my second supervisor. Thanks are also due to the staff of the Edinburgh Microfabrication Facility who fabricated the MOS devices described in this thesis, in particular Mr. Alan Gundlach. Miss Maureen Gray who typed the captions for the figures in this thesis also deserves my grateful thanks.

Lastly, I would like to thank my close friends, in particular my own family for their invaluable support.

DECLARATION OF ORIGINALITY

This Thesis, composed entirely by myself, reports on work conducted by myself in the Department of Electrical Engineering at the University of Edinburgh.

Signed:

I. R. Mactaggart

Definition of Pen Colours in Silicon Layout Plots

N-channel MOS Plots

Colour	Mask
Green	Diffusion
Blue (dotted)	Buried Contact
Red	Poly-Silicon
Black	Contact
Black (dotted)	Implant
Blue	Metal

Silicon on Sapphire Plots

Colour	Mask
Green	Islands
Green (dotted)	PMOS device Implant
Red	Poly-Silicon
Black (dotted)	NMOS device Implant
Black	Contact
Blue	Metal

Contents

CHAPTER 1 Introduction	Page
1.1 General	1
1.1.1 FFT Memory	4
1.1.2 FFT Control	4
1.1.3 FFT Fast Arithmetic	6
1.2 Layout of Thesis	7
CHAPTER 2 Fourier Transform Processing	12
2.1 Introduction	12
2.2 The Discrete Fourier Transform	13
2.3 The Fast Fourier Transform	22
2.3.1 The Radix-2 Decimation-in-Time Fast Fourier Transform	25
2.4 The Prime Radix Fourier Transform	39
2.4.1 The Prime DFT	40
2.4.2 Analogue Prime DFT Computation	44
2.5 FFT System Considerations	47
2.5.1 Arithmetic Concurrency in the FFT	47
2.5.1.1 Single Arithmetic Unit FFT System	47
2.5.1.2 Pipelined FFT System using One Arithmetic Unit per Pass	49
2.5.1.3 Highly Concurrent FFT System using N/2 Arithmetic Units	50
2.5.2 Control Distribution	51
2.5.3 The use of Associative Memory	53
2.5.4 Input Conditioning	56
2.5.5 Output Conditioning	58
2.5.6 Signal Growth and Data Scaling	59
2.5.7 Noise Considerations in a Practical FFT System	60
2.5.7.1 Analogue to Digital Conversion Noise	60
2.5.7.2 Coefficient Quantisation Noise (W)	61
2.5.7.3 Roundoff Noise due to Data Scaling	62
2.6 Review of VLSI FFT Devices	63
2.6.1 Single Chip FFT Processors	63
2.6.2 FFT Arithmetic Processors	66
2.6.3 Parallel Digital Multipliers	68
2.7 FFT Control Devices	70
2.7.1 General Purpose Control Units	71
2.7.2 Silicon Compilation for FFT Control Units	74
2.8 Special Memories for the FFT	76
2.9 Summary	77
CHAPTER 3 High Bandwidth Vector Arithmetic	83
3.1 Introduction	83
3.2 CORDIC Arithmetic Approaches	83
3.2.1 Vector Rotation in Cartesian Coordinates	89
3.3 Distributed Arithmetic Methodology	93

3.3.1 Introduction 93
3.3.2 Consequences of using Distributed Arithmetic 97
3.4 Distributed Arithmetic for Computing Small DFT's 99
3.4.1 General 99
3.4.2 Distributed Arithmetic and the Prime DFT 100
3.5 Distributed Arithmetic and the FFT 100

CHAPTER 4 VLSI Datapath Architectures for Complex Number Arithmetic 112
4.1 Introduction 112
4.2 Conventional Arithmetic Datapaths 112
4.3 Distributed Arithmetic Datapaths 118
4.3.1 Pipelining 119
4.4 Summary 131

CHAPTER 5 MOS-LSI/VLSI Distributed Arithmetic Processors 134

5.1 Introduction 134
5.2 EU201 - A Totally Parallel 6-Bit Radix-2 FFT Butterfly 136
5.2.1 General 136
5.2.2 Fast Adder for Data Assimilation at Array Output 141
5.2.3 Clocking Scheme 145
5.2.4 Performance of EU201 Device 145
5.2.5 Summary of EU201 Butterfly Processor 146
5.3 EU219 - A Configurable 8-Bit Version of EU201 148
5.3.1 General 148
5.3.2 Single Input and Single Output Port 148
5.3.3 High Speed Multiplexer 150
5.3.4 Global Four Phase Clock 151
5.3.5 Tri-State Logic and Timing Considerations 154
5.3.6 Formation of Butterfly Outputs 156
5.4 Digital and Analog Testing of EU219 156
5.4.1 General 159
5.4.2 Probe Testing 159
5.4.3 Testing EU219 on a Logic Analyser 160
5.4.4 Test1 for EU219 161
5.4.5 Test1 Results 163
5.4.6 Test2 Results 163
5.4.7 Test3 171
5.4.8 Test4 171
5.4.9 Test5 171
5.4.10 Analogue Performance of EU219 177
5.4.11 Summary of the EU219 Butterfly Processor 180
5.5 A 16-Bit CMOS-SOS Arithmetic Processor 185
5.5.1 General 185
5.6 CMOS SOS Processor Architecture 186

5.6.1 Systolic Array Complex Multiplier using D.A. 188
5.6.1.1 High Data Throughput 188
5.6.1.2 High Arithmetic Precision 191
5.6.1.3 Low Rounding and Arithmetic Noise 191
5.6.1.4 Low Power Consumption 194
5.6.2 Data Sorter 194
5.6.3 Final Adder/Subtractor 196
5.6.4 CMOS Design Considerations 196
5.6.5 Clocking Scheme 202
5.6.6 Timing Requirements 206
5.6.7 Latching of Input Data and Coefficients 208
5.6.8 Control Signals Required by Pipeline 210
5.7 Logic Simulation 212
5.8 Additional Cell-Level Details 212
5.9 Summary 223

CHAPTER 6 Conclusions 225
6.1 Future Research Work 228
Appendix 1 - PLA's in Silicon Compilers 229
Appendix 2 - Authors Publications 235

Chapter 1 - Introduction

1.1. General

The central aim of this thesis is to study the application of Very Large Scale Integration (VLSI) to high speed computation of the Fourier Transform, and in particular, the Fast Fourier Transform (FFT) algorithm.

The FFT system will be discussed with a view towards the construction of VLSI architectures, however, much of the original work in this connection will be centred on VLSI architectures for computing the arithmetic requirements of the discrete and fast Fourier transform which involves the use of complex numbers.

The use of distributed arithmetic is shown to be highly applicable [1,2,3,4] to parallel arithmetic datapaths for operation on complex numbers, and the datapaths that will be described are very efficient at performing vector rotation (as complex multiplication) and addition.

Some specific VLSI implementations of these datapath architectures are described and their performance is compared with commercial devices.

The Fourier Transform [5] named after the French mathematician, Jean Baptiste Joseph de Fourier (1768 - 1830) allows a continuous or discontinuous function defined over a finite interval to be represented as the integral (infinite summation) of an infinite number of complex exponentials, each with a potentially unique amplitude and phase. The record of these amplitudes and phases is

commonly referred to as the frequency domain if the input is derived from the time domain. This can be expressed mathematically as shown in equation (1.1.1).

$$X(f) = \int_{-\infty}^{+\infty} x(t)\, e^{(-2\pi ft)}\, dt \qquad (1.1.1)$$

Where x(t) represents the input time domain waveform which is transformed to a frequency domain output waveform X(f).

If the time domain signal is periodic, bandwidth limited and sampled, then it is possible to represent the Fourier integral as a finite summation of complex exponentials [5] to a good approximation. In the Discrete Fourier Transform (DFT), a periodic time domain waveform of N complex samples are transformed to another N complex samples, each sample representing the magnitude and frequency of a specific rotating vector which may be considered to be present in the input waveform as determined by the process of complex multiplication with fixed unity magnitude coefficient vectors rotating in the opposite direction ("beating" these frequency components in the data vector to DC), followed by summation. When all the resultant (equally spaced) integer frequencies of rotation described in the output record of the DFT are summed as is done in the inverse discrete Fourier transform (IDFT), the original time domain waveform may be reconstructed.

The DFT has applications in radars, [6,7] (such as in high bandwidth Doppler beam sharpening systems), vocoders, [8,9,10] and also, scientific work [11] such as X-ray diffraction analyses. In

many of these applications, the DFT itself may not be fast enough and in such cases, a Fast Fourier Transform (FFT) algorithm may be used, of which there are many. The FFT is particularly useful in performing high speed convolutions where two signals may be convolved together by performing multiplication in the frequency domain followed by an inverse Fourier Transform. This approach can result in real computational savings [12] for medium to large convolutions (lengths greater than 64 or 128 samples).

Described by Cooley and Tukey, [13] the Fast Fourier Transform (FFT) algorithm, allows the DFT to be computed very efficiently for transform sizes that are some positive integer power of two. Subsequently, a large number of similar algorithms [14,15,16,17,18] have been described for computing the DFT very efficiently. Even with the improvement offered by FFT algorithms, however, high bandwidth signal processing often requires that special purpose hardware be used instead of general purpose hardware.

The advent of Very Large Scale Integration (VLSI) has greatly influenced the design of digital systems, allowing partitioning of systems to be considered at ever increasing levels of functionality.

The precise partitioning of a given FFT system depends on the degree of arithmetic concurrency required in the system, however, in general three major partitions of the FFT can be identified. These are :

a) Memory to buffer and store intermediate data and results.

b) Control unit for coordinating memory and arithmetic.

c) Arithmetic unit which must handle complex arithmetic.

1.1.1. Memory

Current memory technology and designs have greatly advanced [19] to what could almost be described as an art form. The design of high performance memory is an extremely skilled and profoundly complex task which industry has been addressing for many years now. Special purpose memory designs for FFT work would therefore have to offer substantial gains over general purpose Random Access Memories (RAM) to be considered for use in FFT systems. There do appear to be special memory architectures which could offer some advantages over standard RAM for FFT computation which will be discussed in Chapter 2, however, none of the ideas will be pursued since ordinary RAM can be used [12] without much inconvenience.

1.1.2. Control

The control requirement is highly algorithm dependent, which dictates that the control unit be general purpose, or easily programmable. Ordinary Read Only Memory (ROM) can be used to store control data, however, a single memory is not usually an optimum approach [20] as this does not support a control hierarchy efficiently. This is because all communications to and from internal memory registers must take place via the data port which therefore limits the bandwidth of a control unit. Also, a single large memory

will generally be slower than a number of (interconnected) smaller memories, each handling a specific elementary control routine.

The design of a control unit and the design of a data storage unit (memory) are thus closely related, since both operation codes and data can be held in memory. Although random logic can offer lower area and higher speed than memory based logic, a general purpose, or programmable control unit would have to be ROM based, to allow ease of programming which is the main consideration in VLSI design. The problem of control is primarily seen therefore, as a software problem, with the hardware design being closely related to memory hardware design.

General principles and approaches to FFT control will, however, be discussed later with the Programmable Logic Array (PLA) being advocated for the construction of Finite State Machines (FSM's). The PLA is highly suited for incorporation into a silicon compiler as one of the basic cells for producing dedicated control chips. A silicon compiler is a piece of software which can translate a high level description of a circuit or system to an actual design layout which conforms to the layout rules for a given fabrication process, and can thus be used directly for the generation of masks for that process. The compiler may also have the facility for simulation and test vector generation so that when the device is fabricated, the testing can be run automatically, by comparing the device output data with computer simulated data.

1.1.3. Fast Arithmetic

High performance, complex arithmetic processors are not generally available and are most likely to be the limiting part of a system, particularly with large wordlengths. Also, the availability of a FFT arithmetic unit as a major partition of the FFT does not impose any major constraints on a system designer, in terms of transforms sizes or degree of arithmetic concurrency in the system. A high bandwidth FFT arithmetic unit, streamlined to performing complex arithmetic is therefore an important feature in any FFT system. As mentioned therefore, most of the original work described in this thesis is centred on the design of a number of high bandwidth complex number arithmetic datapaths which represent highly optimised structures, streamlined to the computation of the DFT and FFT arithmetic requirements and are therefore major building blocks in such systems.

This work has involved a study of bit-level arithmetic algorithms with a view to achieving the most efficient mapping onto silicon that is likely to be possible. It was noted that distributed arithmetic techniques appeared to offer good properties in relation to mapping onto silicon, and this realisation eventually led to the design of a number of special purpose arithmetic units which made use of distributed arithmetic techniques to efficiently compute the complex multiply which is a central arithmetic requirement of the DFT and FFT.

1.2. Layout of Thesis

The second chapter, which follows this introductory chapter discusses the DFT and FFT computation techniques. The prime DFT is also discussed, as this can be computed in a unique manner. The FFT system is then looked at in detail in regard to partitioning, arithmetic concurrency, control methodology, signal growth, input and output signal conditioning and problems in achieving sufficient versatility in high bandwidth systems.

Having discussed the various requirements of the FFT system, it is concluded that a VLSI arithmetic processor chip would represent a highly suitable partition of the FFT which would not constrain the system designer in regard to arithmetic concurrency or transform size.

This leads directly on to chapter 3 which investigates the mathematical basis of a number of algorithms for highly efficient arithmetic processing some of which are highly suited to silicon implementation. This includes distributed arithmetic, which allows the re-formulation of well behaved mathematical functions, and can be used in computing the complex multiply, through the merging of multiplier partial products.

Chapter 4 then looks at specific VLSI datapath architectures and a comparison is made between conventional arithmetic datapaths and those that make use of distributed arithmetic. In particular, distributed arithmetic is applied to parallel data computation of the complex multiply which is a dominant arithmetic requirement of the DFT and FFT. Structures with various degrees of pipelining are

considered, offering a variety of processing bandwidths.

In chapter 5 some of the architectures described in chapter 4 are applied to some specific VLSI implementations for fabrication in two Metal Oxide Silicon (MOS) technologies, n-channel MOS (nMOS) and Silicon on Sapphire complementary MOS (SOS-CMOS). This chapter is largely devoted to the description of actual silicon devices, and the simulations and digital testing of some of these devices will be described along with their performance.

This leads finally to Chapter 6 which contains the conclusions so far reached with this work, and some suggestions for future VLSI processor architectures based on distributed arithmetic techniques.

Chapters 1 to 3 are thus largely devoted to introducing the subject area and reviewing literature that is relevant to the DFT, FFT and VLSI signal processing in general. There is some original content present in these chapters mixed in with the literature review during discussion. Chapters 4 and 5 which describes some specific parallel data distributed arithmetic architectures and three silicon implementations is original work. A speed programmable nMOS PLA generator, the concept of which was discussed in chapter 2, and is described in appendix [1], is also original material.

References

1. I. R. Mactaggart and M. A. Jack, "Radix-2 FFT Butterfly Processor using Distributed Arithmetic," Electronic Letters, Vol. 19, pp. 43-44 (20th January 1983).

2. I. R. Mactaggart and M. A. Jack, "A Distributed Arithmetic Radix-2 FFT Butterfly Processor," IEE Procds. on The Impact of High Speed and VLSI Technology on Communication Systems, pp. 50-52 (November 1983).

3. I. R. Mactaggart and M. A. Jack, "A Distributed Arithmetic Radix-2 FFT Butterfly Processor," Ninth European Solid-State Circuits Conference, pp. 41-44 (21-23 September 1983).

4. I. R. Mactaggart and M. A. Jack, "A Single Chip Radix-2 FFT Butterfly Architecture Using Parallel Data Distributed Arithmetic," IEEE Journal of Solid State Circuits, Vol. SC-19 No.3, pp. 368-373 (June 1984).

5. E. O. Brigham, The Fast Fourier Transform, Prentice Hall (1974).

6. M. Schwartz and L. Shaw, Signal Processing, McGraw Hill and Kogakusha (1975).

7. K. Ainslie, "Synthetic Aperture Radar," BSc Hons. Dissertation HD 132, (May 1981).

8. P. E. Blankenship, "A Review of Narrowband Speech Processing Techniques," IEE Conference Publications 180, pp. 108-118 ().

9. M. C. Davie, "Low Bit Rate Speech Communication Based on Charge Coupled Device Fourier Transform Processors," PhD Thesis, Dept. of Electrical Eng., Univ. of Edinburgh, (September 1979).

10. M. C. Davie, "A Channel Vocoder Based on CCD DFT Processors," Proceedings IEE Part F, Vol. 127, Issue 2, (April 1980).

11. C. Kittel, Introduction to Solid State Physics, Wiley (1976 (5th Edition)).

12. L. R. Rabiner and B. I. Gold, Theory and Application of Digital Signal Processing, Prentice Hall (1975).

13. J. M. Cooley and J. M. Tukey, "An Algorithm for Machine Computation of Complex Fourier Series.," Math Comp., pp. 297-301 (April 1965).

14. E. Dubois and A. N. Venetsanopoulos, "A New Algorithm for the Radix-3 FFT," IEEE Trans. ASSP, Vol. ASSP-26, pp. 222-225 (June 1978).

15. S. Prakesh and V. V. Rao, "A New Radix-6 FFT Algorithm," IEEE Trans. ASSP, Vol. ASSP-29, pp. 939-941 (August 1981).

16. R. C. Singleton, "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," IEEE Trans. Audio Electroacoustics, Vol. AU-15, pp. 45 - 55 (June 1967).

17. L. R. Rabiner and C. M. Rader, Digital Signal Processing, IEEE Press (1972).

18. C. S. Burrus and P. W. Eschenbacher, "An In-Place In-Order Prime Factor FFT Algorithm," IEEE Trans. on Acoustics, Speech and Signal Proceesing, Vol. ASSP-29, pp. 806-817 (August 1981).

19. S. K. Wiedmann, "Advancements in Bipolar VLSI Circuits and Technologies," IEEE Journal of Solid State Circuits, Vol. Vol. SC-19 No. 3, pp. 282-290 (June 1984).

20. C. Mead and L. Conway, Introduction to VLSI Systems, Addison Wesley (1980).

Chapter 2 - Fourier Transform Processing

2.1. Introduction

In this chapter, the emphasis will be on the influence of VLSI (Very Large Scale Integration) on Fourier transform processing with the focus being on digital techniques which allows higher precision and performance control than can be achieved using analogue approaches. The latter will be discussed only briefly in connection with prime length Discrete Fourier Transforms.

The various ways of computing the Discrete Fourier Transform (DFT) from direct methods to algorithmic Fast Fourier Transform (FFT) approaches will be reviewed, and much of the emphasis will be placed on how these algorithms can be partitioned and mapped on to silicon to produce high performance VLSI processing elements. Existing signal processing devices and system design methodologies relevant to Fourier transform processing will therefore be reviewed. Most of the structures discussed are not general purpose in the true sense but could be reconfigured in real time, to produce a variety of signal processing functions.

The chapter will start by discussing the theoretical aspects of the Discrete Fourier Transform (DFT) and vector rotation alongside silicon realisations of the basic DFT. This will be followed by a summary of the original Cooley Tukey FFT with its associated hardware implications. The Prime length DFT will then be reviewed separately from the non-prime length DFT. The final section in this chapter looks at system considerations of the FFT processor such as partitioning, the problems of input and output signal conditioning

and bandwidth matching of the various processing elements in a hardware FFT system.

2.2. The Discrete Fourier Transform

Any periodic waveform can be represented [1] as the sum of an infinite number of orthogonal periodic functions. If these functions are complex exponentials, then determination of the phase and magnitude of these functions is known as Fourier analysis. For band limited, sampled signals, it is possible to represent the input waveform with a finite number of complex exponential functions to the required degree of accuracy. If an input time domain waveform consists of N complex samples, then the Discrete Fourier Transform (DFT) allows this waveform to be represented as an N sample record of the phases and amplitudes of the N complex exponentials from which the time domain waveform can be synthesised. The DFT can thus be expressed as shown in equation (2.2.1) :

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{(-2\pi\, jnk/N)} \qquad (2.2.1)$$

Where x(n) is the time domain sampled at intervals n=0,1,..N-1 ; X(k) is the frequency domain at intervals k=0,1,..N-1, and N is the transform size.

To form each frequency domain result (X(k)), requires N complex multiply and accumulate operations, so that if all frequency domain results are computed, then N^2 complex multiply and accumulate operations must be computed. Future equations will be simplified by

defining the complex exponential in the DFT equation as shown in equation (2.2.2) :

$$W_N = e^{(-2\pi\ j/N)} \qquad (2.2.2)$$

Where the transform size (N) shall be implied from the text. An eight point (N=8) DFT, expressed in matrix form and using the shortened expression of equation (2.2.2), would appear as shown in Figure (2.2.1).

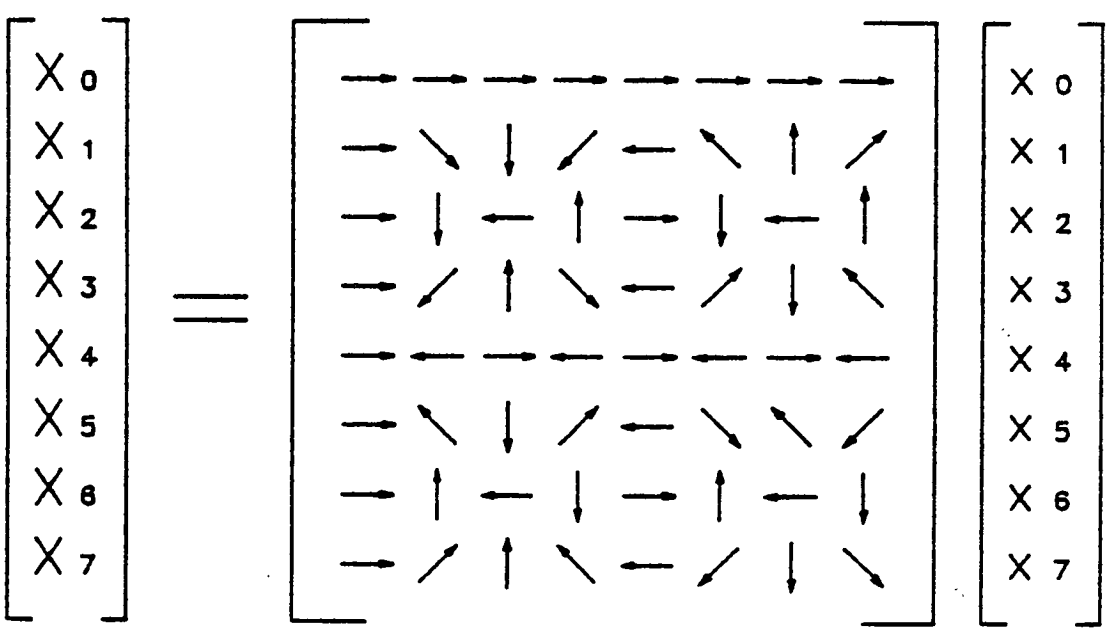
The matrix of W values shown above, has been simplified by noting the periodicity of W. This periodicity is expressed in equation (2.2.3).

$$W^{(n+mN)(k+lN)} = W^{nk} \qquad m,l = 0,+/-\ 1,+/-\ 2.... \qquad (2.2.3)$$

The unit vectors, W_N in the W matrix of the eight point DFT can be illustrated by the vector-matrix notation of Figure (2.2.2). This shows more clearly how the different rows contain successively increasing integer rotation rates, moving from the top row, which represents zero rotation rate (DC), to the bottom row, which represents maximum rotation rate. If the time domain input sequence, consists of a simple unity magnitude vector rotating anti-clockwise at a frequency of one cycle per transform (8 sample periods), then multiplication and summation of this sequence with row 2 of the W matrix, results in a large output at X_1 since the vector sequence of row 2 in the W matrix rotates in an equal and opposite direction. The time domain data vectors are thus

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ W^0 & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ W^0 & W^5 & W^2 & W^7 & W^4 & W^1 & W^6 & W^3 \\ W^0 & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

Figure(2.2.1.) The Eight-Point DFT

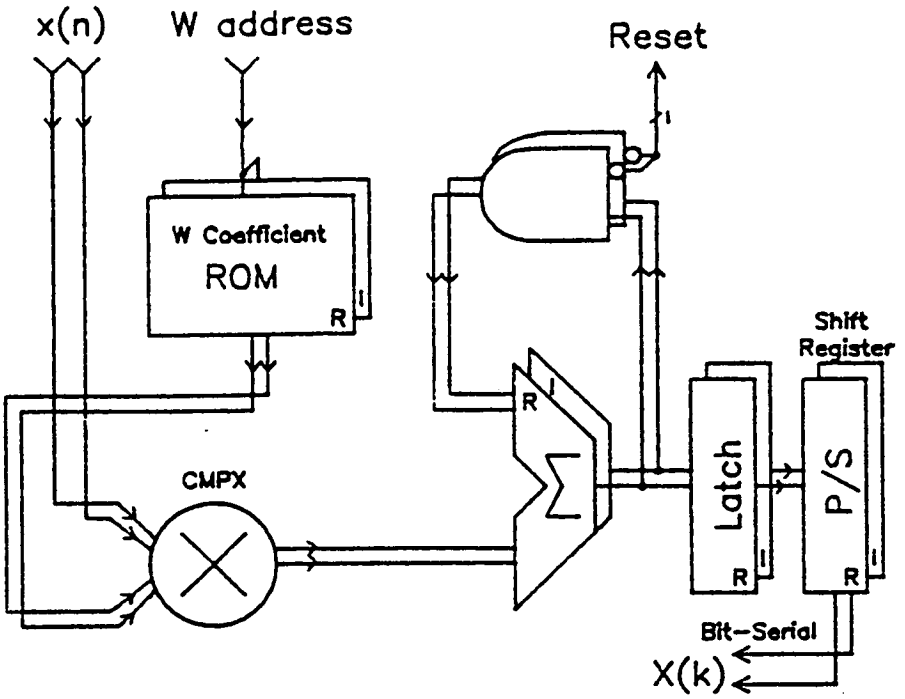


Figure(2.2.2.) The Vectors in the W matrix of an Eight Point DFT

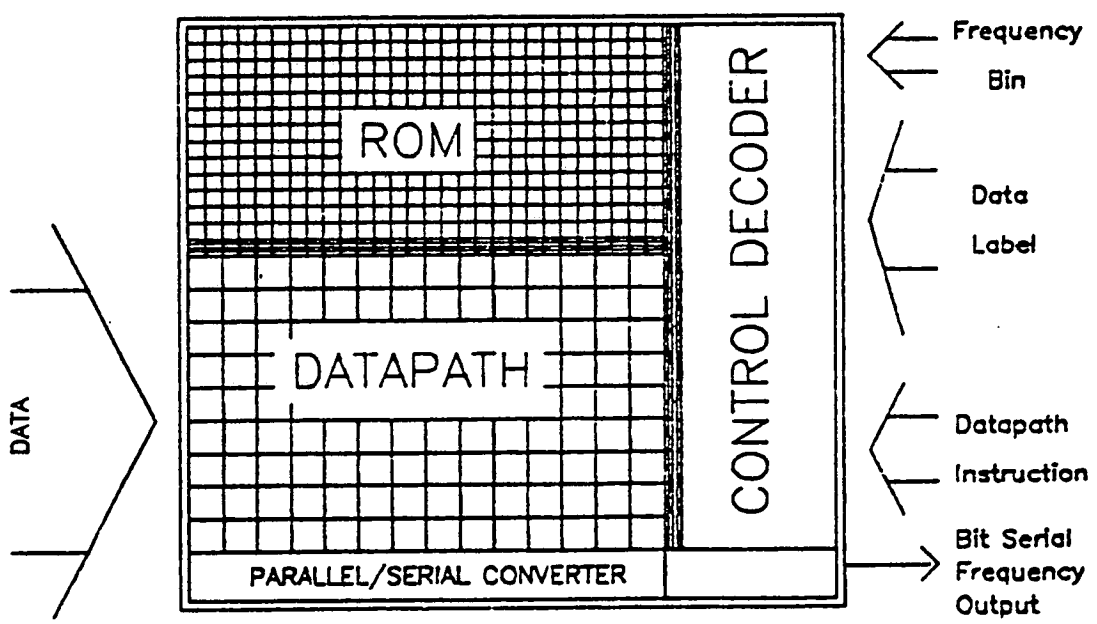
effectively made stationary when multiplied by successive terms in row 2 of the W matrix. All other rows, containing different rotation directions and rates will not make the input data sequence stationary, so these (X(k), k ≠ 2) outputs will all be zero.

In practical situations, the time domain sequence may not contain any frequency components which are perfectly coherent with one of the integer frequencies represented by the W matrix. This results in a slightly reduced output in the nearest discrete frequency "bin" and a small output in adjacent bins. This leakage effect to adjacent bins can be minimised [2] by input signal conditioning (data windowing) as described by Harris.

Analysis of the DFT equation (2.2.1), shows that each frequency domain result is formed by a complex multiply and accumulate operation between the data sequence and individual rows of the W matrix. Control requirements of the DFT therefore consists of accumulator initialisation and sequential transfer of the time domain data and coefficient (W) data to a complex multiplier. Latching of the frequency domain result present in the accumulator would finally be required. Figure (2.2.3) indicates the basic hardware requirements of the DFT processor. This shows the frequency domain output in bit-serial [3] form. The output could also be presented in parallel form, but this was shown in the diagram to emphasise the fact that the data rate at which frequency domain is produced with the DFT is very low and bit-serial data transmission is capable of handling these bandwidths (except for very small DFT's where the computation time is greatly reduced).



Figure(2.2.3.) The Basic Hardware Requirements of the DFT



Figure(2.2.4.) The DFT is well suited to VLSI, as it Yields Highly Regular Structures.

Compared with Fast Fourier Transform algorithms (FFT's) - used to compute the DFT very efficiently for large transforms (discussed in section 2.3) - the control requirements of the DFT are relatively simple. Consequently, for limited transform sizes, up to around 256-points, the hardware DFT may be preferable to the hardware FFT even though at this size of transform the DFT requires about 64 times the number of vector rotation operations than does the FFT. Other attractions of the DFT processor over the FFT processor include marginally improved noise performance [4] and the ability to compute part of the spectrum efficiently if the whole spectrum is not required [4] from a single processor, as might be necessary if it was desired to employ some degree of arithmetic concurrency to enhance the system bandwidth. (Reasons for marginally improved noise performance in the DFT are due to practical considerations related to the finite word lengths used in the hardware rather than any intrinsic failings of the FFT algorithm.)

VLSI allows the DFT to be integrated onto a single chip [4] allowing the computation of up to 256 points in as short a time as 6.5 mS. This chip can process data sufficiently fast to cover most audio applications, (Bandwidth =((1/2) x (256/6.5)) kHz = 19.7 kHz and illustrates pointedly why the hardware DFT is to be preferred for all but large transforms and high bandwidths.

A single chip DFT processor would ideally include coefficient storage, and make use of parallel arithmetic processing. A typical chip architecture is shown in Figure (2.2.4) which is representative of a VLSI single chip DFT processor design based on parallel arithmetic data. This type of controlled datapath architecture allows

for a more general purpose approach to be taken than was the case with the dedicated DFT chip just described. The DFT chip architecture of Figure (2.2.4) could easily be applied in a system by labelling the time-domain data to provide a pointer to the W coefficient stored on-chip. This might be expressed in shorthand as shown below (see also equation 2.2.1) where the "+" refers to a requirement for information, not an arithmetic operation.

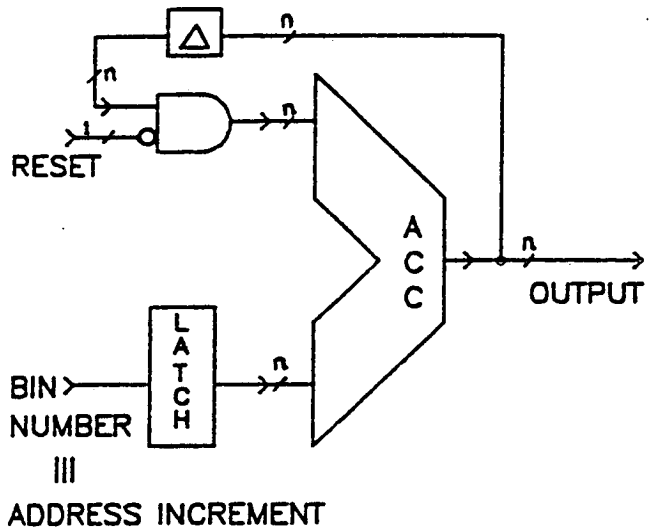
(Data Label) + (Bin Number) -> (W Coeft. Address)

Where "Bin Number" is the number of the frequency domain bin that is being computed. (Bin Number would be constant for each bin computation.) This can be computed very easily in practice by looking at the remainder of the product of (Bin Number) and (Data Label) when divided by the transform size (N) as a consequence of the periodicity of the complex exponential described in equation (2.2.3). Data Label in this case is chosen to be equal to the time domain sample number. The division by N, however, need not take place if N is a power of 2. In this case the bottom $\log_2$ N bits of the product yield the correct W coefficient address. Thus if the transform length (N) is a power of two, the correct DFT W coefficient address can be computed with a single multiply operation. In practise, the same multiplier used to perform vector rotation operations would be used to generate the W coefficient addresses in this manner. The general case for any value of N can be expressed as shown in equation (2.2.4).

$$W_{Address} = \text{Remainder o f [Bin Number x Sample Number/N]} \qquad (2.2.4)$$

Although this appears to be a sound approach to the generation of W addresses in a DFT processor, for higher speeds it is desirable to free the arithmetic unit for operation on data only. A less computationally intensive method to generation of W addresses is to perform an accumulation, which allows a new address to be computed as a function of a previous address and the bin number. This allows the W addresses to be computed in their natural order (for direct complex multiplication with data samples) but has the disadvantage common to all recursions - that the effects of a single error taking place during the recursion will remain until the recursion is terminated. Thus if an error occurs during the computation of a frequency bin, the whole bin computation needs to be repeated unless there is a suitable mechanism for detection and removal of the error as soon as it occurs. In a carefully designed digital circuit, however, errors should be very infrequent and should not cause serious problems in small recursions of this nature. The approach to generating DFT W coefficient addresses using a single accumulator is shown in Figure (2.2.5). The bin number is latched from a counter and the new coefficient address is computed by adding this to the old coefficient address. An example is included in Figure (2.2.5) for rows 4 and 8 of an eight point DFT. These were chosen to show the wrap-around effect which is not entirely an obvious phenomenon.

A potential hardware reduction feature of the DFT is that an input buffer is not required if one chip per frequency bin is used since no temporary storage or sorting of input data is required. This approach is quite attractive if bit-serial arithmetic [3] processors are employed. The use of parallel arithmetic processors



A DFT Coefficient Address Generation Scheme

Row 4	Row 8
000 = 0	000= 0
+011	+111
011 = 3(10)	111= 7(10)
+011	+111
110 = 6(10)	[1]110= 6(10)
+011	+111
[1]001 = 1(10)	[1]101= 5(10)
+011	+111
100 = 4(10)	[1]100= 4(10) etc
⋮ ⋮	⋮ ⋮
101 = 5(10)	[1]001= 1(10)
+011	+111
[1]000 = 0	[1]000= 0

Note :

Bracket indicates discarded overflow bit.

Rows 4 and 8 show Wrap-around effect

Figure(2.2.5.) DFT, W coefficient Address Generation using Accumulator-Eight Point Example uses a 3-Bit Accumulator.

would not normally be considered due to high costs for all but limited transform sizes, or very high bandwidth systems. The use of N arithmetic processors [5] (one for each frequency channel) provides a useful means of high bandwidth computation of the DFT, as an alternative to the FFT for limited transform sizes. Although such a system is algorithmically less efficient than the FFT, a failure in a single processor would only affect the frequency bin which that processor was associated with, thus other bins would not be affected. In the FFT, described shortly, single arithmetic failures can affect many frequency bins, not just one, and thus have catastrophic effects. In DFT systems using a single chip, the input data sequence may be held in a simple shift register buffer which can be re-circulated for computing each new frequency component. Such a buffer has minimal control requirements. One obstacle to implementing all this on a single chip is that with complex arithmetic, the memory requirements are doubled. The DFT does, however, require minimal control and so for small transforms and limited bandwidths it is the ideal approach to computing the Fourier Transform digitally. When large transforms and high bandwidths are required then it is necessary to use algorithmic approaches which involves additional control data and temporary storage in order to reduce the amount of arithmetic required.

2.3. The Fast Fourier Transform

An analysis of the (W) matrix of a highly composite DFT such as the eight-point example shown in Figure (2.2.2), reveals that many of the complex multiplications are redundant, in a manner which is

independent of the input time domain sequence, being a feature of the (W) matrix itself. Although there are obvious redundancies in the DFT W matrix where some of the multiplications are ...

1) Trivial (Multiplications by (+/-) 1 or (+/-) j)

2) Repetative (Due to periodicity of the complex exponential) - see equation (2.2.3)

3) Related (Multiplications that are negative or complex conjugates of each other)

the efficiency of the common radix-2,4 and 8 FFT algorithms hinges on the fact that when the transform length (N) is some integer power of two (as is the case in all Radix-2,4 and 8 FFT's) then each element in the W matrix can be generated iteratively by $\log_2 N$ complex products from a set of elementary unity magnitude vectors whose angles are binary weighted multiples of W_N. These elementary unity magnitude vectors are commonly referred to as "twiddle factors" in literature, [6] however, the algorithm has not yet been fully formulated, so for the moment, this name only applies to the numerical value of these vectors.

This now allows each frequency bin computation to be factorised into these twiddle factors and other terms containing the input sequence. The factorisation may be chosen to be such that many terms are produced which are common to each frequency bin in a systematic manner thus allowing a reduction in the number of arithmetic operations. The DFT output may then be generated by $\log_2 N$

iterations (or passes) on the input sequence, in which the time domain is operated on successively by these twiddle factors. The mathematical basis of the FFT re-formulation of the DFT will be discussed in more detail in the section dealing with the Radix-2 Decimation-in-time FFT algorithm.

Thus, the FFT algorithm is based upon the principle of revealing redundancies through factorisation made possible by the iterative re-construction of the W elements in the DFT matrix and exploiting these redundancies through data-routing. As previously mentioned, more than one factorisation is possible and so also there are a number of ways of ordering the data-routing in the computation to produce a variety of "Fast Fourier Transform" (FFT) algorithms.

The original Cooley-Tukey FFT algorithm [7] described the general technique of breaking down large DFT's, whose lengths are highly composite, into a large number of much smaller DFT's. Thus, a large DFT could be computed by combining together several much smaller DFT's. As the number of complex multiplications to compute the DFT increases as N^2, and the combination of trivial DFT's in the FFT algorithm increases only as $N/2\log_2 N$, the basic Cooley-Tukey FFT algorithm is more efficient than the DFT by a factor of $2N/\log_2 N$ times.

A mathematical proof of the most important FFT algorithms already exists [6] so a formal proof is not repeated here. Instead, the probable thinking that led to the realisation of one of the best known and most popular Fast Fourier Transform (FFT) Algorithms, the Radix-2 Decimation-in-Time FFT, will be outlined. There are several

FFT algorithms all closely related, such as the radix-4, the radix-8 and mixed radix algorithms [6] however, the radix-2 FFT is the most versatile, since with the lowest radix (two), it permits transform sizes which are integer multiples of two. It also allows for simple time domain windowing on the first pass for reasons which will be apparent when this type of algorithm is examined in the next section. (The first pass is trivial).

2.3.1. The Radix-2 Decimation-in-Time FFT.

In the DFT itself, there is no restriction on the transform length (N), however, if N is restricted to powers of 2, then it is possible to express any of the N distinct W_N^{nk} terms in the (N x N) DFT coefficient array as the complex product of $\log_2 N$ fixed vectors which are binary weighted multiple powers of W_N. This is expressed in equation (2.3.1.1).

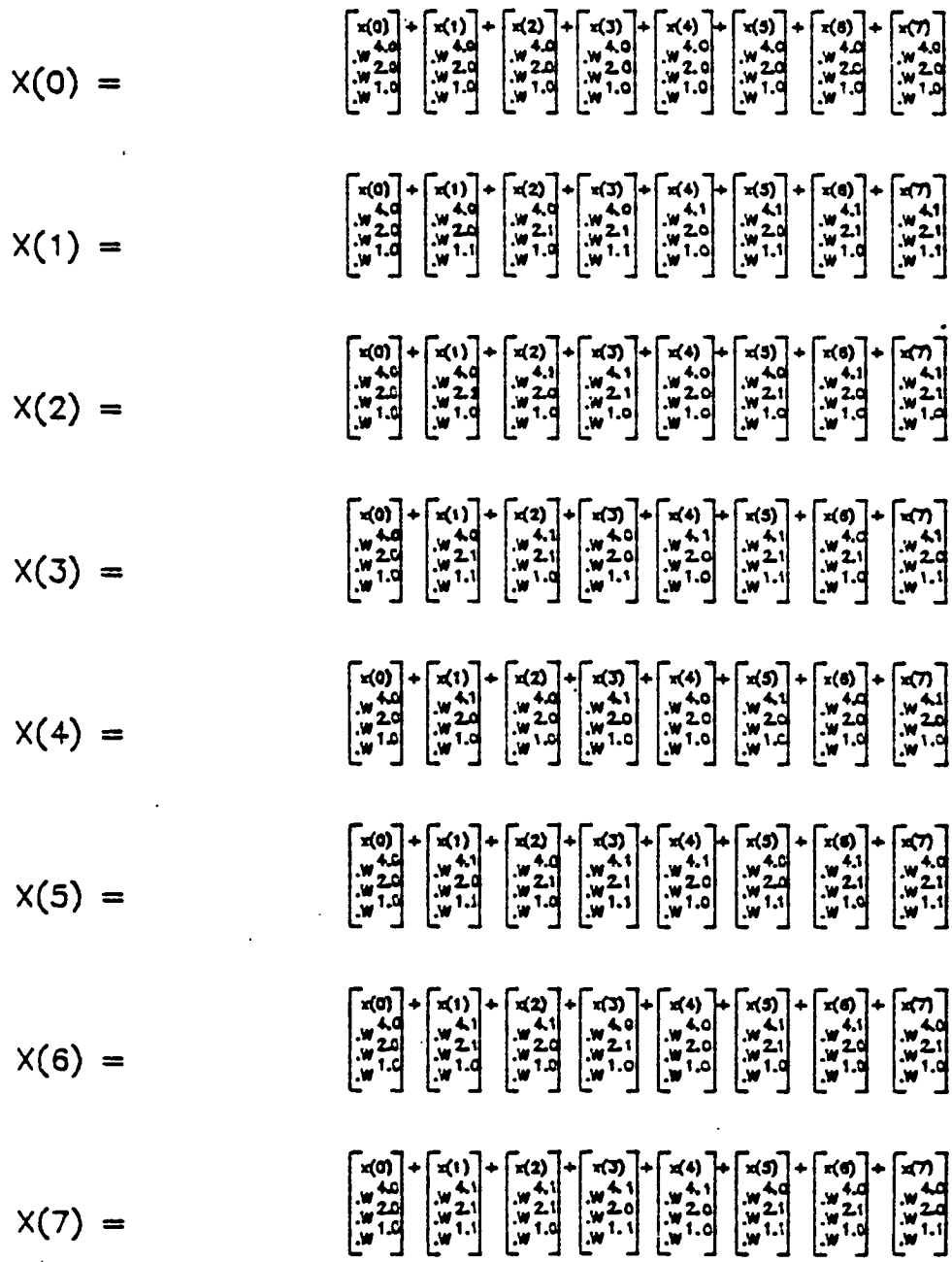
$$W_N^{nk} = \prod_{m=0}^{(\log_2 N)-1} W_N^X$$

$$\text{where} \quad X = X_m = d_m\, 2^m \qquad (d_m = 0 \text{ or } 1 \text{ only}) \qquad (2.3.1.1)$$

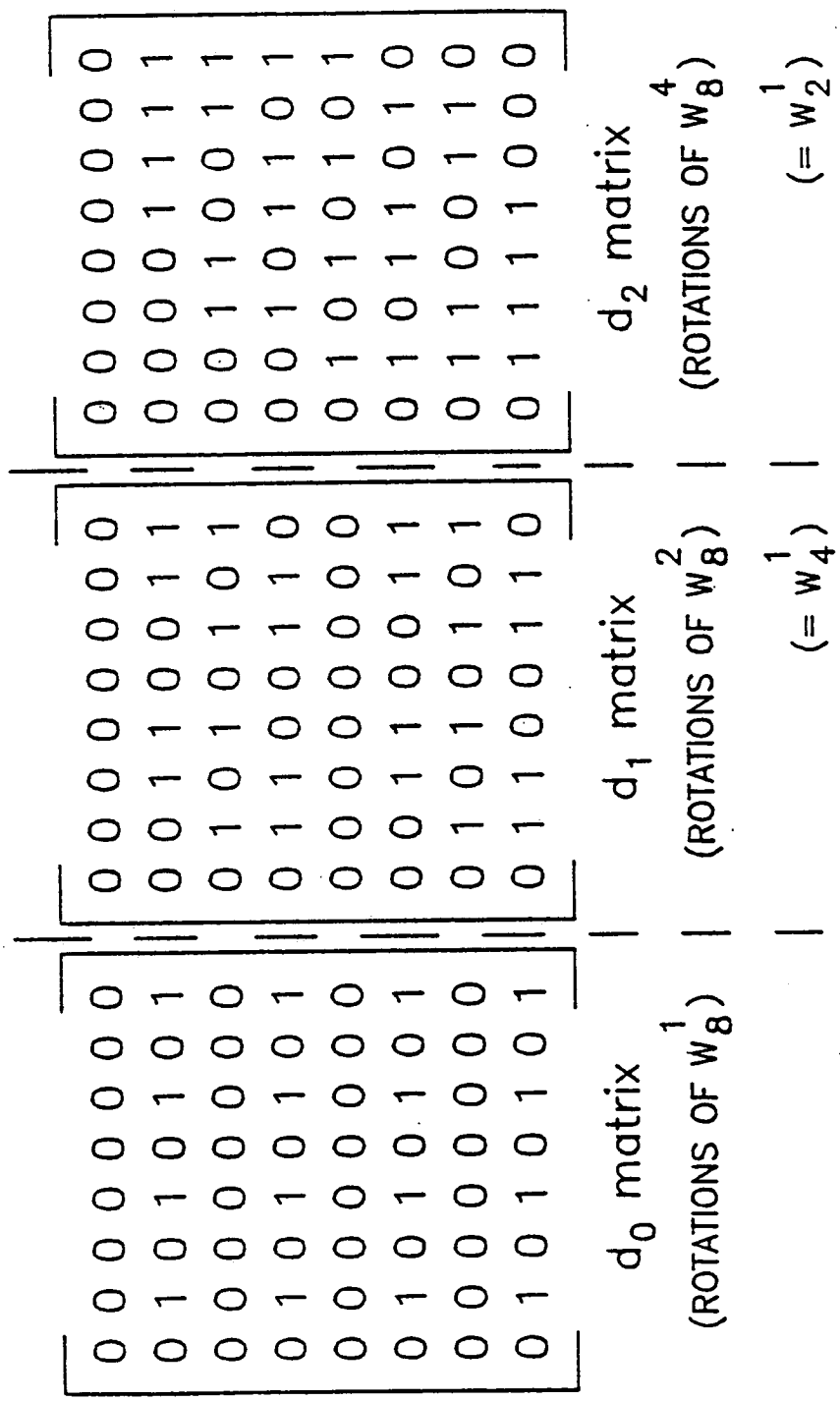
(For example, $W_8^5 = W_8^4 \cdot W_8^0 \cdot W_8^1$)

The DFT may thus be conceptualised as shown in equation (2.3.1.2)

$$X(k) = \sum_{n=0}^{N-1} \left(\left(\prod_{m=0}^{(\log_2 N)-1} W_N^X \right) x(n) \right) \qquad (2.3.1.2)$$



Figure(2.3.1.1.) The Eight Point DFT Showing Factorisation of the W Coefficient Terms.



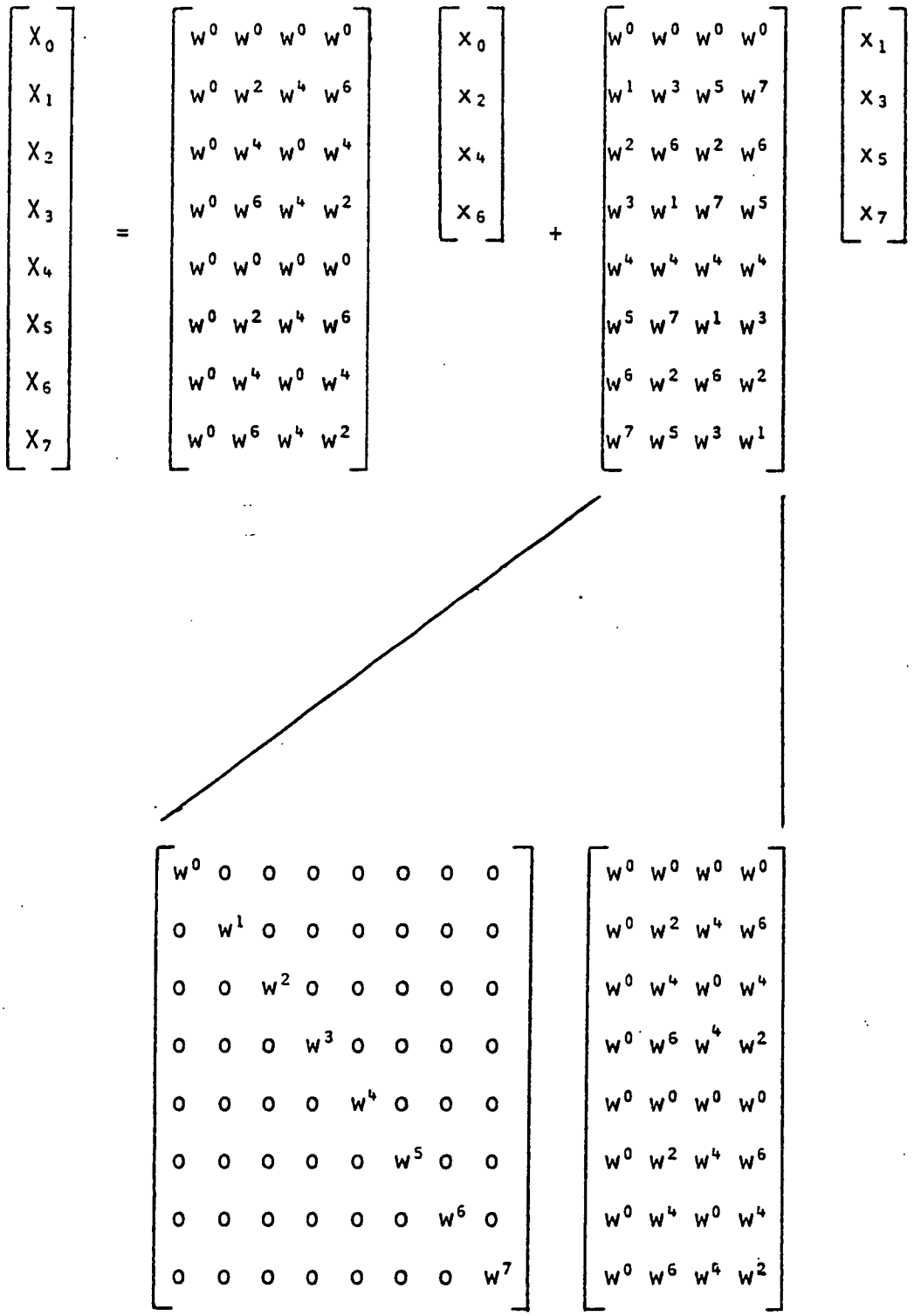
Figure(2.3.1.2.) The Component Rotations in the W Matrix of an Eight Point DFT. (A "1" indicates a given non-zero Rotation).

In the above equation each element in the W matrix of the DFT is now expressed as $\log_2 N$ complex products. This is written out in full for an eight point example in Figure (2.3.1.1). Figure (2.3.1.2) shows three matrices containing the binary values of 'd' in equation (2.3.1.1) for the eight point example indicating how each term in the original W matrix of the eight point DFT can be considered to be composed of a product of binary weighted rotations. (The concept of these 'd' matrices may not be described in literature)

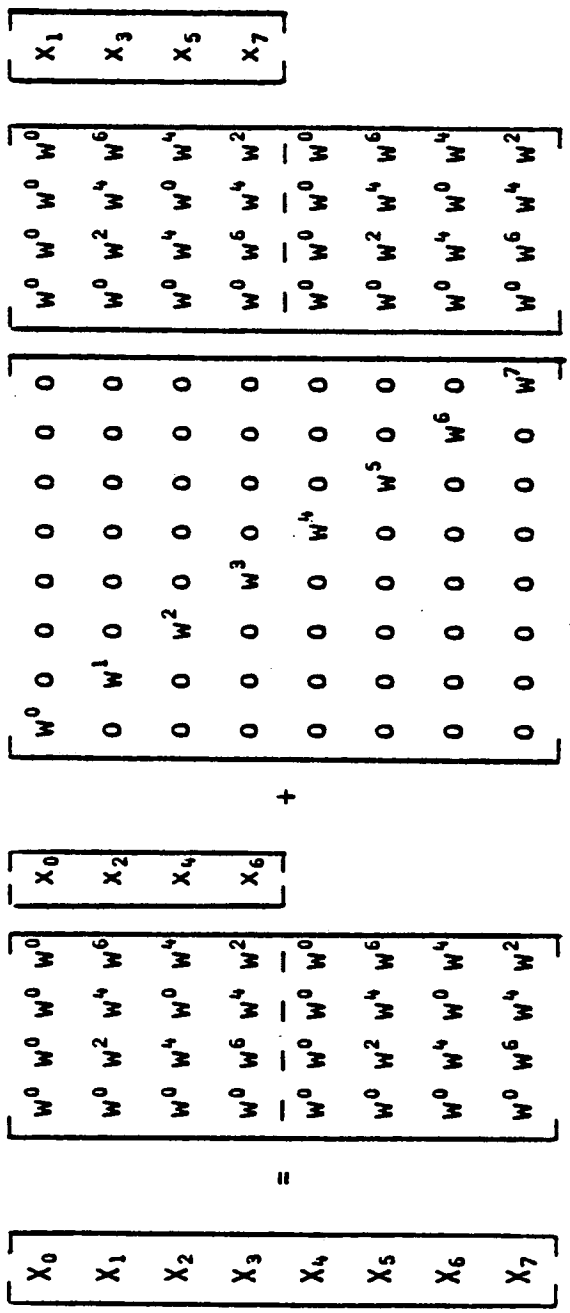
Analyses of the matrix d_0 which indicates which DFT coefficients require minimum rotations (W_N), reveals exceptionally high symmetry. It is fairly clear that this matrix would be of the same type for any size (N where N > 2) of DFT provided that N was some integer power of two.

If this symmetry allowed the N point DFT to be split into two and expressed as two N/2 point DFT's each of which would have the same type of d_0 matrix (describing minimum rotations for that size of DFT), then this splitting process could be continued right down to N=2. If a computational saving could be made each time a DFT was split into two, then the overall computational saving (N>>2) would be significant.

Further examination of the d_0 matrix that describes which elements in the W matrix of the DFT contains minimum rotations reveals that the EVEN columns contain no such rotations. This implies that half of the N=8 point DFT may be computed as a single N(=8)/2 point DFT operating only on the EVEN numbered time domain samples. The ODD



Figure(2.3.1.3.) The Process of Decimating an Eight Point DFT into two Four Point DFT's.



Figure(2.3.1.4.) The Complete Decimation of an Eight Point DFT into two Four Point DFT's.

columns however, do contain minimum rotations. These can be removed at the expense of N (=8) rotations of which N(=8)/2 are non-trivial, allowing the remainder of the original N=8 point DFT to be computed with another N(=8)/2 point DFT this time based on ODD valued sample numbers. This process is shown for the eight point example in Figure (2.3.1.3) and (2.3.1.4) showing the construction of an 8-point DFT from two 4-point DFT's. In general it can be said that instead of N^2 rotations to compute an N-point DFT, only $(N/2)^2 + (N/2)^2 + N/2$ rotations are required if it is computed as two N/2 point DFT's. This of course is less than the direct approach. If N is large and a power of two then the complete decomposition of the N point DFT into N/2, 2-point DFT's can be accomplished by performing the above process iteratively resulting in a radix-2 algorithm, so called, because the transform is built up from 2-point DFT's. This complete decomposition of the DFT is shown for N=8 in Figure (2.3.1.5). This is shown for each frequency bin in Figure (2.3.1.6) in full. This reformulation of the DFT is highly significant in that fewer arithmetic operations are required and the whole DFT can in principle be built up from a simple arithmetic function shown in Figure (2.3.1.7) (frequently described in literature as the radix-2 "butterfly" - cf. wings of a butterfly!) using the signal flow graphs shown in Figure (2.3.1.8) which can be derived from Figure (2.3.1.6). This produces un-scrambled frequency domain from time-domain which has been scrambled into bit-reversed address locations. Figure (2.3.1.9) shows an alternative flow graph which allows unscrambled time domain as an input to produce frequency domain in bit-reversed address locations. Bit-reversal which applies to the address of stored data is defined as the mirror image of the address word when mirrored in the Y-axis,

$$X(k) = \sum_{n=0}^{N-1} W_N^{nk}\, x(n) \qquad \text{(DFT)}$$

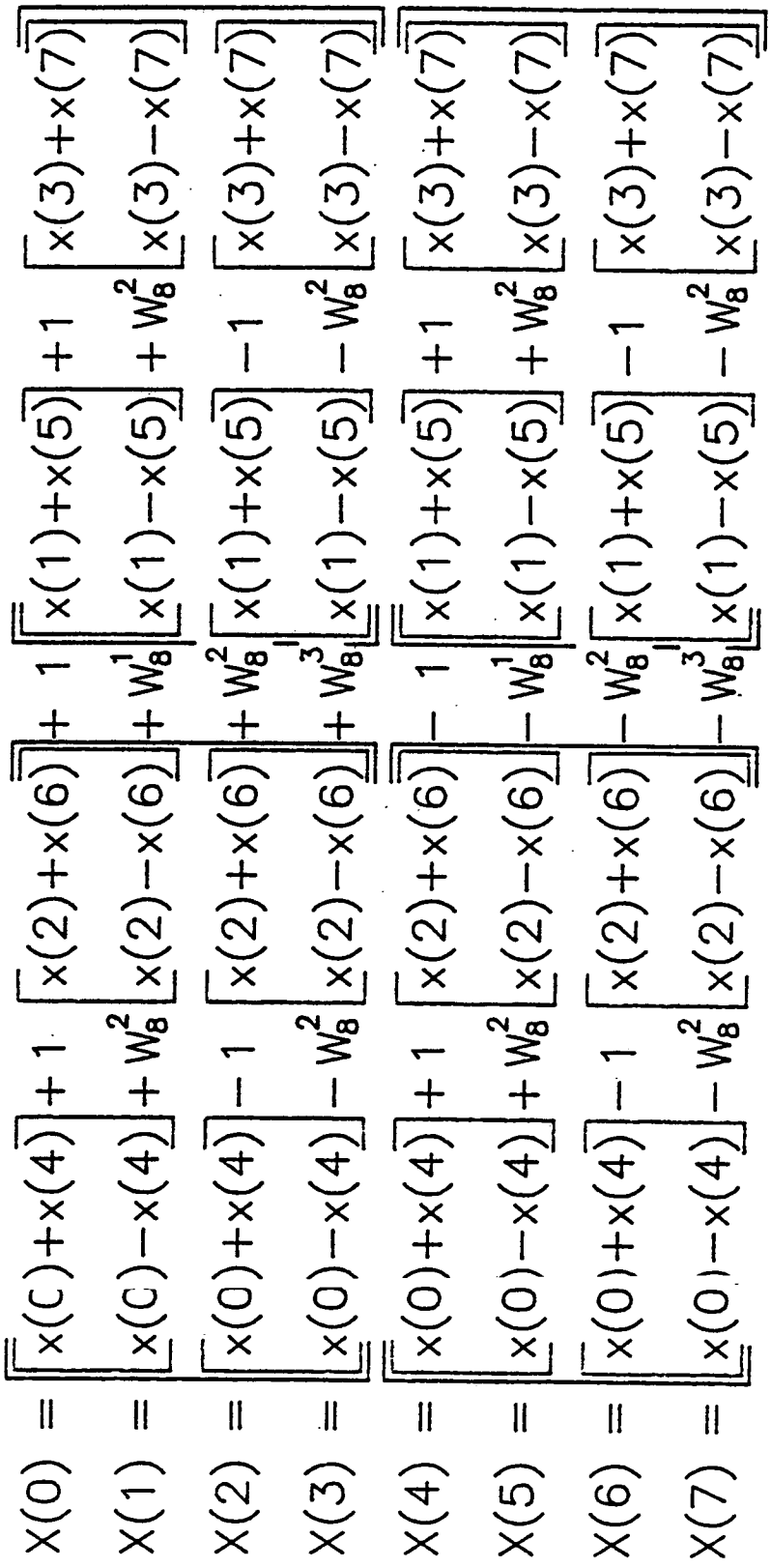
$$= \sum_{n=0}^{(N/2)-1} W_{N/2}^{nk}\, x(2n) \;+\; W_N^k \sum_{n=0}^{(N/2)-1} W_{N/2}^{nk}\, x(2n+1)$$

$$= \sum_{n=0}^{(N/4)-1} W_{N/4}^{nk}\, x(4n) \;+\; W_{N/2}^k \sum_{n=0}^{(N/4)-1} W_{N/4}^{nk}\, x(4n+2) \;+\; W_N^k \left[\sum_{n=0}^{(N/4)-1} W_{N/4}^{nk}\, x(4n+1) \;+\; W_{N/2}^k \sum_{n=0}^{(N/4)-1} W_{N/4}^{nk}\, x(4n+3) \right]$$

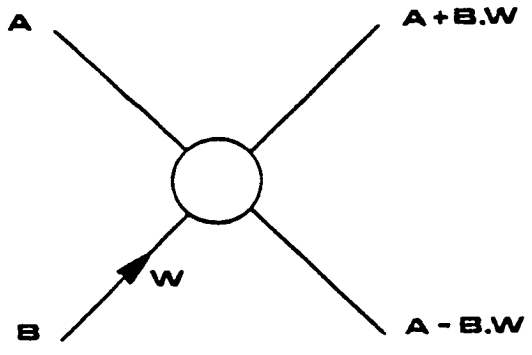
Thus, for N=8

$$X(k) = W^0 x(0) + W_2^k x(4) + W_4^k (W^0 x(2) + W_2^k x(6)) + W_8^k [(W^0 x(1) + W_2^k x(5)) + W_4^k (W^0 x(3) + W_2^k x(7))] \qquad \text{(Radix-2 FFT)}$$

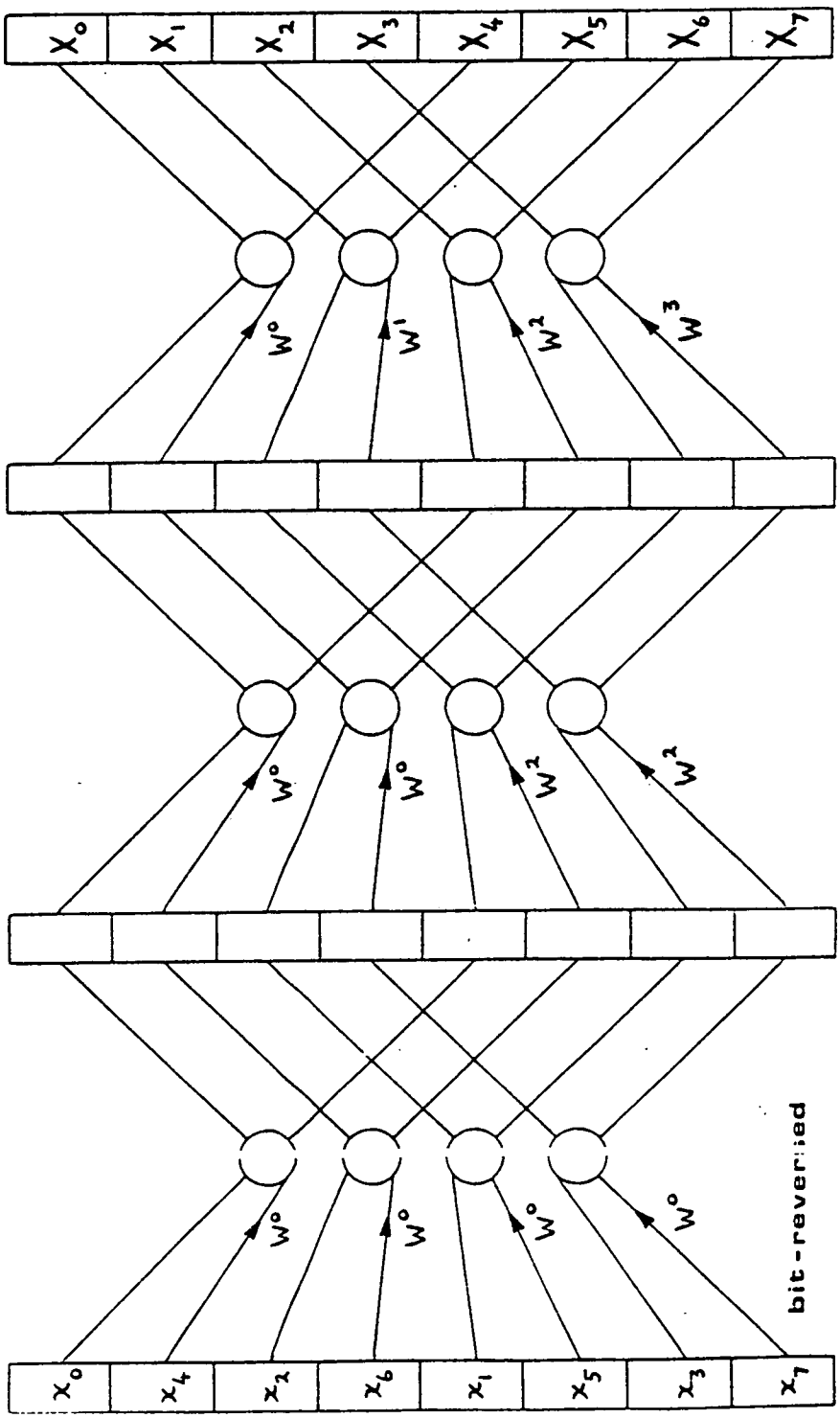
Figure(2.3.1.5.) The Decomposition of an N-point DFT with an example for an N=8 length DFT.



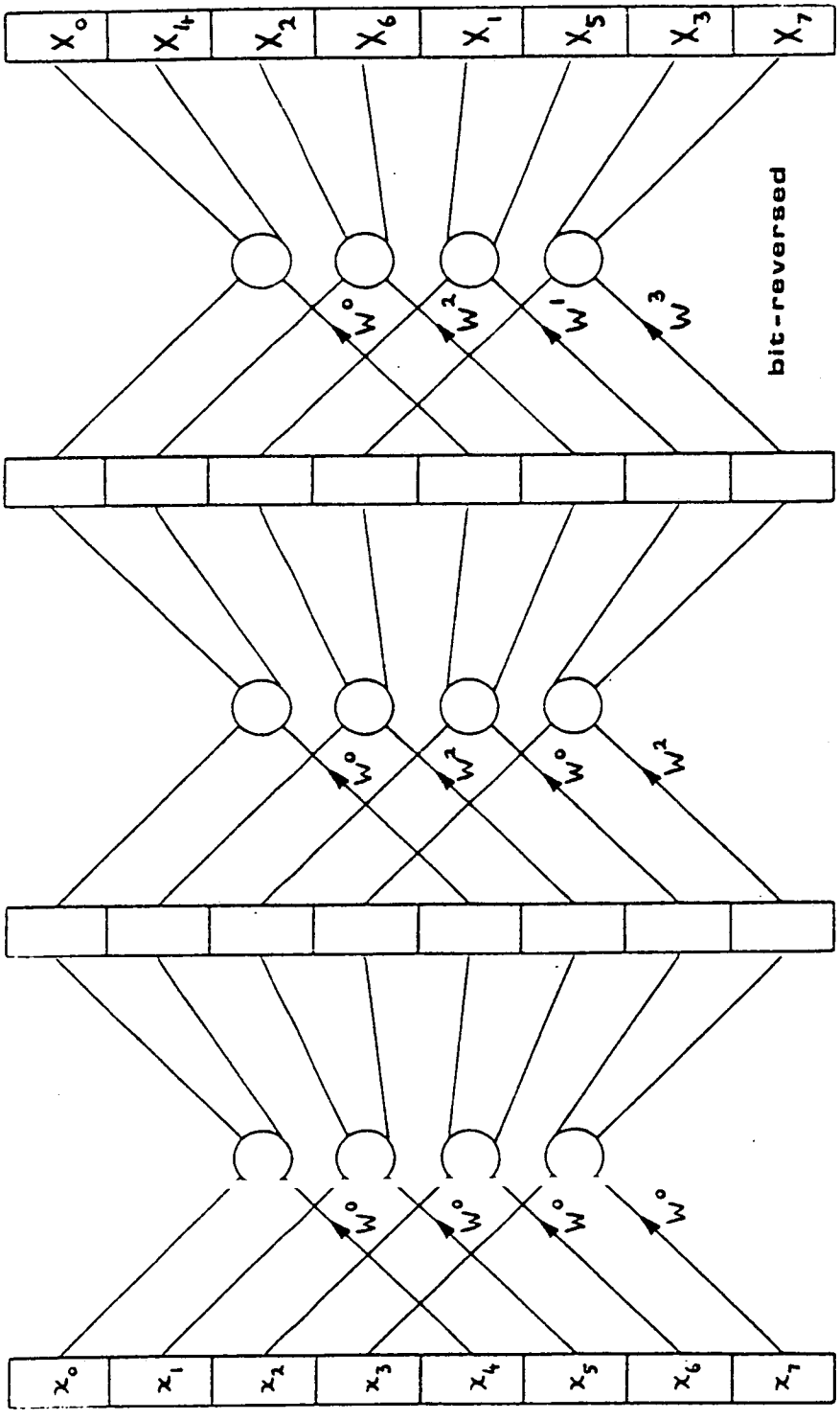
Figure(2.3.1.6.) The Decomposition of an Eight Point DFT into Two Point DFT's showing Redundancy.



Figure(2.3.1.7.) The Basic Computational Element of the Radix-2 FFT Algorithm Described in Text (Decimation-in-Time). Frequently known as a Radix-2 "Butterfly".



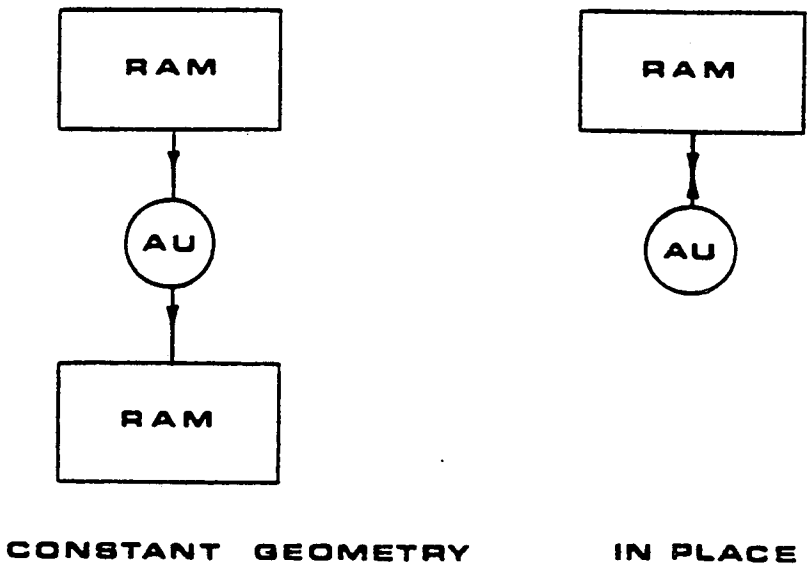
Figure(2.3.1.8.) Data Flow Graph of the Radix-2 FFT Algorithm described in Figure(2.3.1.6).



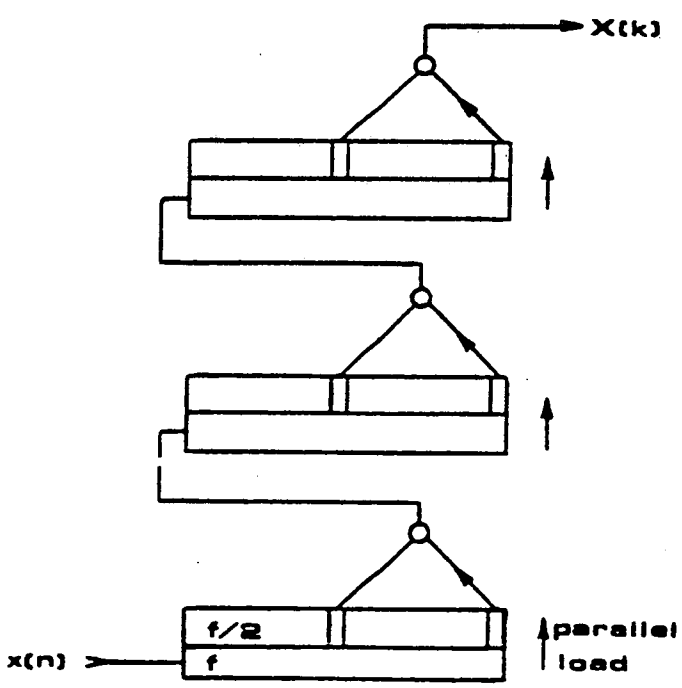
Figure(2.3.1.9.) Alternative Data Flow Graph of Radix-2 FFT with Un-Shuffled Input Sequence.

so that the most significant bits become the least significant bits and vice-versa.

This particular FFT is normally known as the Radix-2 Decimation-in-Time (DIT) algorithm because at each stage in the computation, the input time sequence is divided into smaller stages for processing as outlined in the previous 8-point FFT example. An alternative decimation-in-frequency algorithm [6] is similar and uses the same number of operations. The DIT, FFT algorithm is, however, more useful in hardware oriented systems because the W coefficients are all unity on the first pass, thus allowing the possibility of using this pass to perform time domain windowing functions [6] that are often essential to the operation of real FFT systems. It will be noted that in both flow graphs of the 8-point FFT, the transform is built up from three distinct stages or "passes". A significant feature of Figures (2.3.1.8) and (2.3.1.9) is that the data address sequence is the same for each pass. This type of addressing is thus normally referred to as constant geometry addressing, because of the pass-independent addressing. If pass dependent addressing is used, [6] it is possible to return the butterfly outputs to the same memory address locations from which the inputs were derived, [6] thus halving the memory requirements. This type of algorithm, known as "in-place" places greater demands on the memory speed performance. Figure (2.3.1.10) shows the two different FFT systems based on these different addressing approaches. Figure (2.3.1.11) shows a possible pipelined FFT approach based on serial shift register memory, and constant geometry data flow. The time domain is clocked into a shift register at clock rate (f) and is



Figure(2.3.1.10) Two Distinct Single Arithmetic Unit FFT System Memory Configurations.



Figure(2.3.1.11.) An example of Serial Memory (Shift Register) to Implement a Radix-2 FFT in a Pipelined Processor. (Parallel Load Facility Requires Custom VLSI).

then loaded in parallel into another shift register. This is tapped at the half-way position and clocked at (f/2), thus sorting the data into the correct order for the butterfly.

Data flow graphs can be very helpful in hardware FFT system design as they allow the various possible addressing strategies to be clearly differentiated. They also show how the addressing of data registers relates to the particular FFT algorithm being considered.

It is known that there are a large number of FFT algorithms which work for a variety of data block lengths not necessarily powers of 2. There exist Radix-3 FFT algorithms, [8] and Radix-6 algorithms, [9] as well as the more conventional Radix-4 and Radix-8 [6] FFT's. These unusual Radix FFT's do not offer computational advantages over the Radix-2 and 4 FFT's involving comparable complex multiply operations, however, their main advantage lies in the ability to compute the FFT for input time domain sequence lengths that are not necessarily powers of 2.

2.4. The Prime Radix Fourier Transform

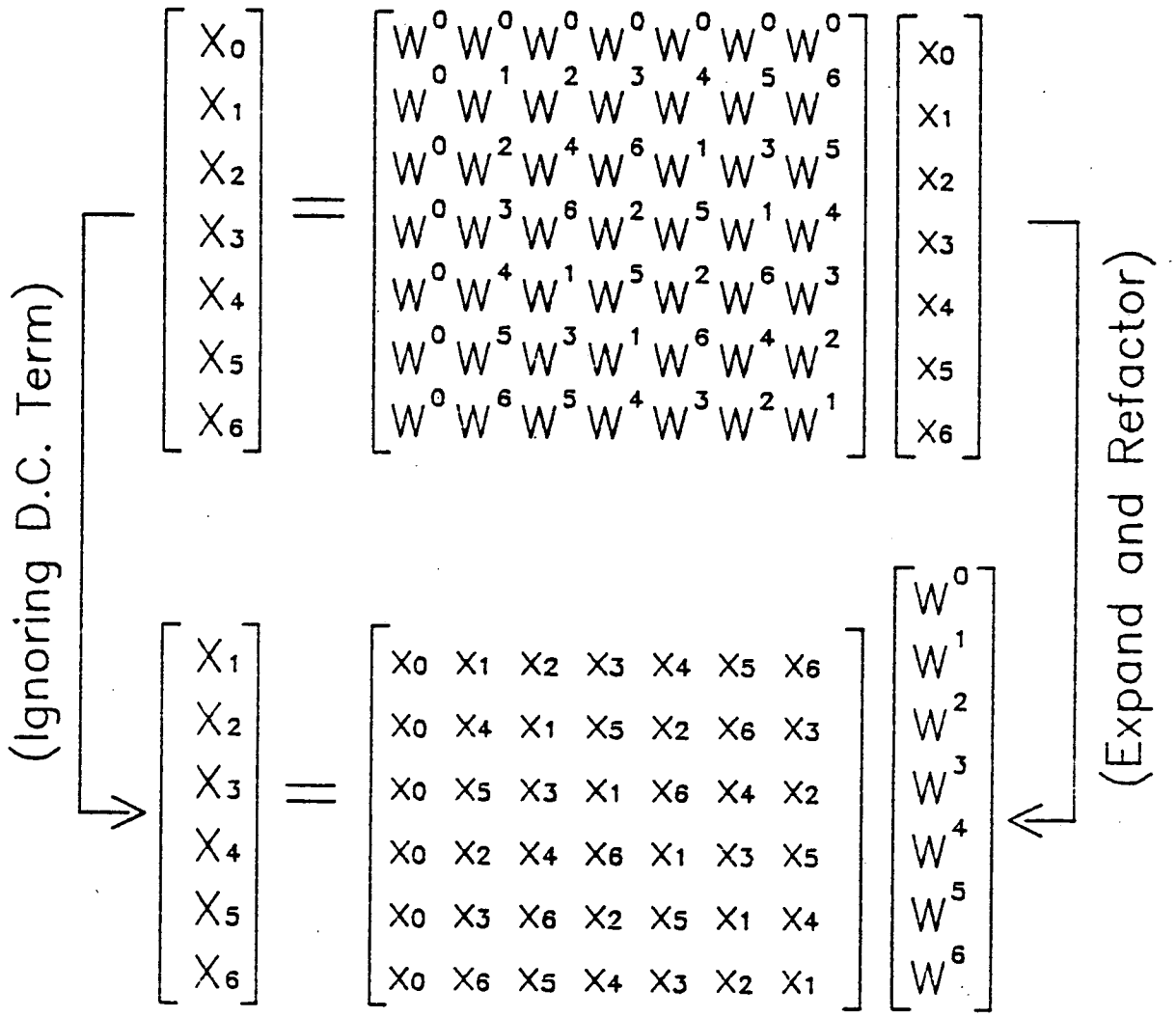
As previously mentioned, the Cooley Tukey FFT algorithm is at best restricted to transform sizes that are some positive integer power of two whereas the Prime Radix FFT algorithms [10,11] involve the building up of a large transform from DFT's some or all of whose lengths are prime. The simplest case involves building up a P = N.M point DFT from N, M point DFT's followed by M, N point DFT's. Unfortunately, the joining up process involves non-trivial rotations

which calls for complex multiplication, thus the only apparent advantage of Prime FFT type algorithms is the ability to compute unusual length DFT's that are not possible using the standard Cooley-Tukey FFT or a radix-3 FFT.

It is worth noting, however, that the Prime DFT has itself some unusual features which make hardware implementations particularly attractive.

2.4.1. The Prime DFT

The N point DFT, where N is prime, is unique, in that each row of the W coefficient matrix contains all Nth roots of unity, with the exception of the row used to compute the DC term. With this one exception then, each row used to compute non-zero frequency components contains every one of the N vector coefficient angles, which are a multiple of (2 x PI/N) radians. This is true only when N is prime and does not apply to the row corresponding to the DC term. Since every other row contains all Nth roots of unity, it is possible to re-order the expansion of the (N x 1) time domain matrix with the (N-1 x N) W coefficient matrix, such that the time domain is scrambled, instead of the W matrix, which can now be made the same for each frequency domain channel, consisting of a rotation of (2 x PI/N) radians between adjacent elements. The prime length DFT can now be expressed as a scrambled (N-1 x N) time domain matrix multiplied by a single column (N x 1) W coefficient matrix, representing a single integer frequency (rotation). An example of this re-ordering is shown in Figure (2.4.1), for N=7. The fact that this single integer frequency can be used to compute all the frequency



Figure(2.4.1.) The Seven Point Prime DFT Showing Re-ordering of the W matrix and Input Sequence.

domain terms, allows techniques such as distributed arithmetic to be used - discussed later in Chapter 3 - where the additions in the multipliers and adders are re-ordered to allow a memory and accumulate reformulation. Siu and Chen [12] describe a system based on a 6800 (2MHz) microprocessor which uses distributed arithmetic techniques to compute a 61 point DFT in only 3.1 ms. Distributed arithmetic can be considered to be a very useful technique in microcomputer systems, although it should be noted that both hardware and instruction set usually need to be specifically "geared" to such programming which includes the ability to efficiently look at individual bits in a word as well as perform conventional word arithmetic. Siu and Chen achieved their high performance by providing hardware to perform such bit-level functions.

The Prime DFT can be further simplified, however, by using conventional arithmetic in a recursive mode. If a given row of the scrambled time domain sequence in the 7 point example (shown in Figure (2.4.1)) is represented as x'_0 x'_6 then, any one of the frequency bins X_n may be represented by the recursive expression of equation (2.4.1).

$$X_n = (((((x'_6 W+x'_5)W+x'_4)W+x'_3)W+x'_2)W+x'_1)W+x'_0 \quad (2.4.1)$$

In this example W has a fixed angle of (2 x PI/7) radians. This equation shows how each frequency cell of the 7 point DFT can be computed recursively by performing a repeated fixed angle vector-rotate/vector-add operation on the scrambled time-domain sequence. The main advantage of this re-formulation is that a fixed-angle

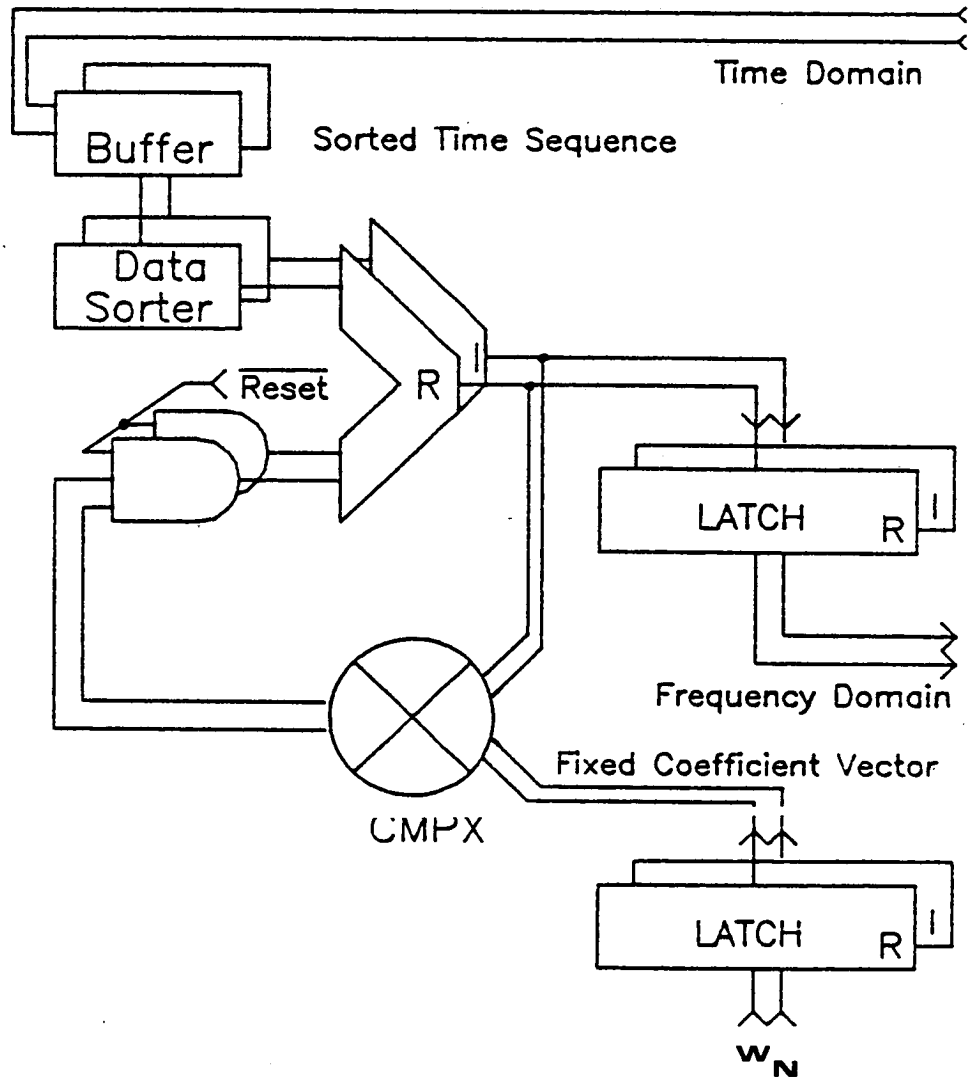


Figure (2.4.2.) Showing Basic Prime DFT Hardware Requirements.

vector rotate can be implemented more efficiently than a variable-angle vector rotate in both hardware or software by using techniques such as shift and add or partial table look-up. This again is particularly attractive in micro-computer systems that are poor at performing signed multiplications but can perform shift and add operations or table look-up operations with efficiency comparable to that of dedicated hardware. In hardware implementations, the main advantage of this approach in computing the Prime DFT is that an extensive coefficient Read Only Memory (ROM) is unnecessary, however, the need for a data-sorter still limits the size of DFT that can be computed on chip. It is likely however that the resulting structure would be more easy to design and permit a larger size of DFT to be computed than could be done conventionally. Figure (2.4.2) shows the basic hardware required to implement the prime DFT.

2.4.2. Analogue Prime DFT Computation

In analogue circuits, the current and/or voltage relationships of one or more active or passive devices are exploited in such a way as to model a mathematical relationship. At room temperatures this model will only exhibit a limited accuracy due to electronic noise of various types. Also, even in the absence of noise, the model itself may only approximate the desired one due to device non-idealities. Such effects can limit the applications of analogue techniques to algorithms requiring only low degrees of precision. This often means that only low degrees of recursion may be tolerated to minimise error build up.

In general, the DFT itself is not implemented effectively by using analogue techniques due to excessive inaccuracies, however, the Prime DFT offers an exception to this rule since in this special case, the frequency domain may be computed using a fixed angle vector rotate circuit, so that the multiplier only has to operate on a fixed coefficient word. A number of papers have already been written on the use of analogue techniques [13,14] in this area. Charged Coupled Devices (CCD's) can be used to store data, and the multiplier can be implemented by using capacitive charge sharing techniques, or by using active devices (Higher Bandwidths). In the case of the charge sharing analogue multiplier, digital techniques can be used to switch in small trimming capacitors, thus avoiding the need to take sensitive analogue signals, off chip. The performance obtained is then acceptable, offering low power, and medium to high bandwidths. In general, however, accuracy is still restricted. Jack, Park and Grant [13] project a possible 0.5% rms transform accuracy, from results of a prototype device which displayed an initial 2% transform accuracy. Although accuracy is not high, low power at high bandwidths up to 5 MHz can be obtained, offering the possibility of real time signal processing tasks at very low costs.

The effect of device scaling, however, has a quite drastic effect on analogue circuits. In general, dynamic range, linearity, and noise performance are all degraded [15] and the need to handle information in a noise tolerant manner must be considered. This gives rise to a need for digital circuits. All digital circuits are analogue in nature, but by defining a threshold, data can be freed from the corrupting influence of noise, and be represented as a

string of binary (two state) data of any desired precision. It has long been recognised, however, that there is no advantage in setting voltage swings (V) and therefore thresholds in logic gates at levels that vastly exceeds the level needed to ensure good noise immunity as this serves only to greatly increase the power required to change the state of a given node (proportional to V^2). This has led to the consideration of special logic gates such as in the case of multiple valued logic [16,17] where more than one analogue threshold is set within a given voltage range and therefore more than two possible states are considered to exist over that range. This quest for lower power and higher speed has also led to the search for lower logic voltage swings and thresholds in conventional digital circuitry, which can still offer good noise immunity with reduced thresholds. Multiple valued (MV) logic is in effect, a half way house between digital and analogue approaches. It is likely, however, that for the same reasons that analogue circuits are declining in usefulness because of device scaling, (which favours digital approaches), MV logic will eventually decline in its usefulness as well. By then, digital VLSI circuits will be operating at more optimised thresholds with lower voltage swings which will offer higher speeds and lower powers, but still retain an adequate noise immunity. This thesis will not be pursuing these analogue approaches any further which are not suited to VLSI. The prime DFT and FFT will not be pursued any further either as it is more complex than the Cooley-Tukey FFT to implement in hardware and does not appear to offer any outstanding advantages. The computational efficiency of both the prime DFT and Prime FFT is not very high compared to conventional DFT and FFT approaches.

2.5. FFT System Considerations

2.5.1. Arithmetic Concurrency in FFT Systems

One of the notable features of all FFT algorithms is that various levels of arithmetic concurrency are possible allowing a wide range of bandwidths as well as system design philosophies.

The three most important levels of arithmetic concurrency which apply to Radix-2 systems involve either a single arithmetic unit, or $\log_2 N$, or N/2 arithmetic units, where N is the transform size. The last approach is really only sensible with serial arithmetic processors. For a 1024 point FFT therefore, one might consider using 1, 10 or 512 butterfly processors. To use numbers other than this is possible, but would involve additional control, and some inefficiency. These three different approaches will be discussed briefly.

2.5.1.1. Single Arithmetic Unit FFT System

The single arithmetic unit processor allows low to medium bandwidth operation, depending on whether serial or parallel arithmetic processing is used. Assuming a butterfly time of 1 microsecond a 1024 point complex FFT based on a single arithmetic unit would take 5.12 milli-seconds to complete a 1024-point transform.

One feature of the single arithmetic unit FFT system, is that input (time domain) and output (frequency domain) buffering must be performed in order that the A/D conversion may take place on a continuous basis. This entails extra memory and control requirements. The constant-geometry algorithm is preferred to the in-place

algorithm for this type of system where it would be very difficult to use a greater than unity latency arithmetic unit efficiently. The latency of an arithmetic processor is defined as the number of clock cycles required between the input of data to the arithmetic processor and the output of a result. This may be much greater than unity even if the processor can absorb data every clock cycle through the use of pipelining [5] techniques. The reason that the constant-geometry algorithm may be preferred is that it would be necessary to execute short bursts of read and write cycles, in order to take advantage of an AU with latency much greater than one. In this case, the AU would only be operating 50% of the time. Even when the constant geometry algorithm is used, problems can arise with an arithmetic unit with latency much greater than one. There is a potential delay in moving from the end of one pass to the beginning of another pass, corresponding to the latency of the arithmetic unit. This is so because the memory used to "sink" pass n data is the same memory that will be used as a source of pass n+1 data, and since a greater than unity latency AU, will still be computing pass n data (even though it is ready to process pass n+1 data), it is not possible to switch the memory round from being a data sink to a data source until the arithmetic pipeline containing the last pass n output data is empty. Thus, the arithmetic processer must be idle for a period corresponding to its latency at the end of each pass. In practice, for large transforms this would entail a fairly small bandwidth penalty, however, it complicates the control requirements still further. All these problems are probably best solved by using a unity latency fast arithmetic unit instead of a pipelined unit. Whilst potentially slowing down the system, this

would simplify the control requirements for the reasons given above.

2.5.1.2. Pipelined FFT System using one Arithmetic Unit per Pass

This type of radix-2 FFT system configuration uses $\log_2 N$ arithmetic units (AU's). The transform is thus computed at a rate comparable to the system clock. With a butterfly time of 1 micro-second, a 1024 point transform would take 512 micro-seconds to compute. The most notable point about using this level of arithmetic concurrency is that the throughput of the system is independent of transform size and input and output buffering is not required. The pipelined system has the further attribute that each AU can be made to operate continuously, since the memory can be configured as a swinging buffer thus allowing a greater than unity latency processor to output data from a current pass, whilst receiving data from an earlier pass. Since this type of system works equally well for unity latency and much greater than unity latency arithmetic processors, this approach must be considered to be very effective in minimising control and maximising bandwidth. The bandwidth of this type of system is high enough to cover most FFT applications including high bandwidth radars if a parallel AU is used. Furthermore, the overall control requirements must be regarded as minimal, with the arithmetic units comprising a much greater percentage of the system gate count than with the previous single AU system. It will be noted, that the only difference between passes are the W coefficient values. The possibility of constructing a single pass FFT sub-system which could be programmed to compute any pass can thus be considered. Then, $\log_2 N$ of these boards could be used in a high

bandwidth FFT system.

This approach is highly versatile as it allows very high bandwidths and also a highly modular system design approach. Also, the possibility of electronically switching in spare pass boards to replace defective boards exists, allowing a highly reliable system. These advantages, together with the absence of input and output buffering suggest that this approach to a FFT system design is probable the most cost-effective for a wide range of operating bandwidths.

2.5.1.3. Highly Concurrent FFT System using N/2 Arithmetic Units

This type of FFT system effectively computes a pass at a time, which might take just a few clock cycles. Bandwidths, therefore are very high. With a butterfly time of 1 micro-second, a 1024 point transform would take only 10 micro-seconds to compute. This approach, is usually only considered viable if bit-serial data communications and processors are used to avoid a potential plethora of wires to connect them together. Such highly concurrent FFT systems cannot be regarded as versatile since there is no simple way to allow expansion to compute larger transforms, for example. Bandwidths are so high that any further processing that must be performed such as conversion from real and imaginary data to magnitudes and logarithm computation for example, must also be performed at unusually high data-rates, necessitating further specialised hardware. This type of approach is however quite realistic with the availability of a bit-serial silicon compiler such as FIRST [18]

which is intended for implementing digital signal processing functions. Such a complier might, for example, be used to produce usable chip designs for a) A Butterfly Processor b) A Vector Magnitude Processor and c) A Logarithm Processor, and possibly, d) A Data-Windowing Processor, thus enabling the diverse requirements of a real system to be met with a minimum of effort.

2.5.2. Control Distribution

System control distribution and structure is of crucial importance, in all digital systems such as the FFT, since if this is not carried out methodically, the resulting system may be highly inefficient in its internal operation and also be difficult to modify. Many current hardware-based single AU FFT systems, may be efficient in their use of components, but have these sort of deficiencies caused by the absence of a structured control hierarchy which makes it impossible for the control store to hold information efficiently. Distributed control is a prerequisite for efficient control data storage in all types of complex control tasks. It is not therefore optimum to have one single control unit in a complex system. Instead the approach should be to have local control associated with each distinct system function such as the memory and arithmetic unit, for example. This would then allow the main control unit to communicate with the rest of the processor at much lower bandwidths and also reduce the degree of abstraction involved in the data communicated between the various units in the system. Another serious deficiency of many hardware FFT and DFT processors is that a corrupted control unit memory address instruction would allow incorrect data to be

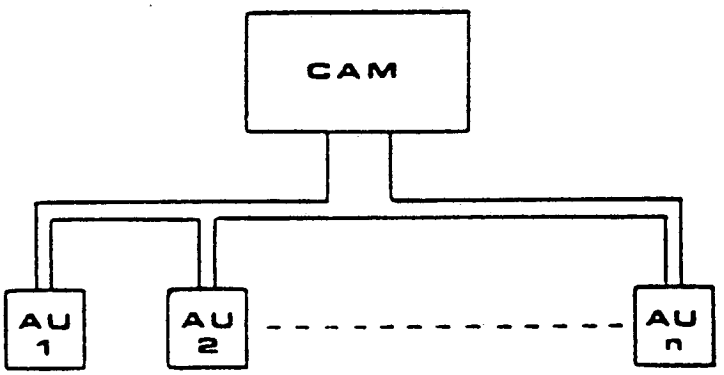
sent to the arithmetic unit which would then combine two unrelated data and coefficient vectors to produce an incorrect result. The arithmetic unit is then fooled into receiving incorrect data passed to it, whilst at the same time, the control unit expects the arithmetic unit to produce a given result in a given time frame. In otherwords, such a system has by itself no means of determining that it is operating correctly, instead, it is the system engineer who analyses the system and declares it to be sound. Although this is how many systems are designed today, it is not the best way to tackle more sophisticated systems that will appear in the future as a result of VLSI. One possible approach which would provide a powerful check that the system was operating correctly, would be to add to each piece of data in memory, a word which could act as a label or tag for that piece of data. This would then allow the arithmetic unit to perform a simple check that the control unit had sent the right types of data to it for processing. When the arithmetic unit had received all the required labels and data, it would then compute the new data together with new labels for the data. Thus, although the control unit is responsible for addressing memory and sending instructions to the arithmetic unit, it is possible for the arithmetic unit to verify that the data types that it receives are correct. This may seem to be superfluous but it does offer a means of verifying correct system operation in real time. As most digital system designers are trained to minimise gate counts and not add to them, this approach is often not thought of as desirable though, and consequently such additional features are not usually employed in most current hardware FFT systems. The main advantage is in reduced system-debugging time, increased versatility and verifying that

system memory accesses are correct. System debugging can be very expensive, so if this is also considered, it could well prove cheaper to add such additional hardware which serves to monitor the systems operation. Owing to the high levels of integration involved, however, these ideas were not pursued towards actual silicon designs, but it is felt that they would be worth considering in the construction of digital FFT system based on a number of standard VLSI parts.

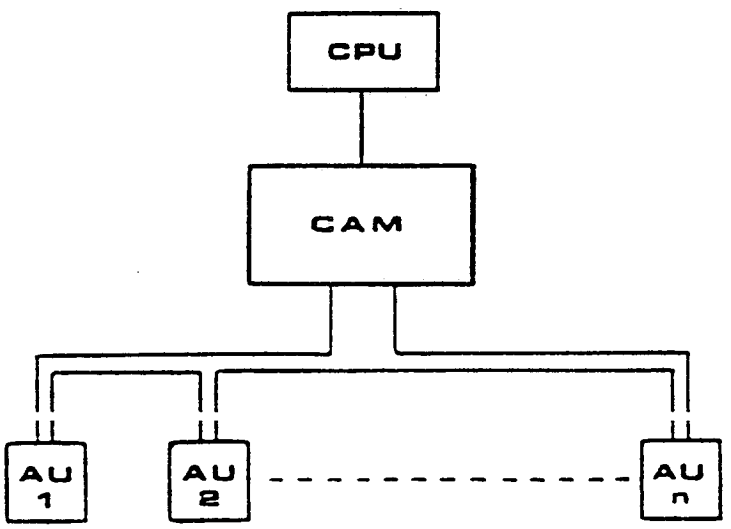
2.5.3. The use of Associative Memory

If a content addressable (associative) memory (CAM) was available which could perform the function "search for any data label of type corresponding to pass "n" and return one such label plus associated data" (this is a typical CAM function [5]) then this feature could be used to ensure that each pass was completed whilst the arithmetic unit could request the correct data within each pass. To perform the FFT, then, the main system control unit would instruct the arithmetic unit to request any pass 1 data present in the CAM (which of course there would be to start with) and the CAM would then respond with one valid example. The arithmetic unit would then compute the labelled data associated with the label fed to it and output a new piece of data with associated label. Having completed this task, the arithmetic unit would then request any additional pass 1 data still available. If there was any, then another pass 1 sample would be sent to the arithmetic unit and so on. The control is therefore reduced to instructing the arithmetic unit to maximise the pass number which it does by requesting data of certain types

(pass n data) from the CAM. This represents a highly distributed control approach. A control methodology, involving the use of "intelligent" memory could offer programming simplifications in general purpose systems, however, it is possibly overkill for FFT and similar work. This is so because CAM's do not offer such a low cost per bit than do conventional coordinate addresses RAM's. The main advantage, possibly of using a content instead of a coordinate addressed RAM is that multi-processor tasks are considerably eased, since individual processors can determine almost instantly, the current state of the computation, from memory, without needing to communicate with another processor. Indeed, it appears that many of the draw-backs [19] of multi-processor based systems can be overcome by using CAM's. Multi-processing, however, is only of use, when memory bandwidths greatly exceed processor bandwidths. This suggests the possibility of an FFT machine based on fast, expandable CAM connected to a variable number (non-critical) of low bandwidth bit-serial processors. Figure (2.5.3.1) shows such a system configuration. The CAM may have its own dedicated processor in order to extend its versatility and perform bus arbitration as shown in Figure (2.5.3.2). This type of approach is quite useful in that low bandwidth processors may be added or subtracted from the system to produce the required overall bandwidth, with an upper limit being dictated by the relative CAM to processor bandwidth. As the precise number of processors is non-critical in such a system, the possibility of switching out defective low bandwidth processors might provide a basis for yield enhancement which would allow the possibility of wafer-scale-integration. The low bandwidth processors would be designed to be capable of computing any arithmetic task in order to



Figure(2.5.3.1.) Possible Multiprocessor FFT Scheme based on Fast Responsive Content Addressable Memory(CAM).



Figure(2.5.3.2.) Similar CAM based FFT Scheme with CPU to Extend Overall System Versatility and Remove some Load from the smaller Arithmetic Units (AU1-AUn).

allow other system requirements such as input and output conditioning to be realised as well as the basic FFT. Unfortunately, the levels of integration involved in pursuing such ideas are currently too high to consider specific silicon implementations.

2.5.4. Input Conditioning

Windowing of data is not part of the DFT or FFT itself, but such a system which is processing real-life signals, (from an analogue to digital converter for example) will not necessarily receive a data sequence which is periodic. A discontinuity of undetermined value will exist between the first input sample, and the last input sample. DFT theory, however, requires that the time signal sampled for processing be periodic [20] over the data block length. A discontinuity in the time domain would have the effect of introducing strong frequency domain components which were a feature of the discontinuity. These extra frequency components are manifest not only as a localised spreading (main lobe spread) around a spectral line but also in a much wider spread (side lobe spread) through the whole of the frequency domain produced by the transform. Data windowing is one way of reducing this problem, where the idea is to slightly modify the time domain sequence in such a way as to force periodicity over the block length, without greatly influencing the true frequency domain content of the signal. This can be done by multiplying the time domain with a window function. The many different window functions reach a compromise between main lobe widening and side lobe reduction. In general the window function will have a form similar to :

$$W(n) = Sin^{a} [n.PI/N] \qquad \text{where } 1.0 < a < 4.0$$

although more complex windows do exist such as the Blackman, Gaussian, Dolph-Chebyshev and Kaiser-Bessel windows which offer varying types of sidelobe reduction. Much work has already been covered in relation to finding optimum windows [2] for given applications by Harris. When a = 2 the general function (above) yields :

$$0.5 [1.0 - Cos [2n.PI/N]]$$

which is very close to the popular Hamming window which is described as :

$$W(n) = 0.54 - 0.46.Cos [2n.PI/N].$$

The window has the effect of removing the discontinuity in time domain, with a consequent improvement in the quality of frequency domain output. This is an important topic primarily because it posses a considerable processing overhead which must be considered alongside the butterfly arithmetic requirement for the FFT itself. On the first pass of the Radix-2 Decimation in Time FFT, the phase factors are all unity, suggesting that the multiplier in the butterfly might be used to perform data windowing on the first pass.

This requirement for data-windowing influenced the design of the CMOS-SOS 16 bit arithmetic processor device described in Chapter 5. The same complex multiplier used to compute vector rotation could also be used to perform a time domain windowing function.

2.5.5. Output Conditioning

The output of the DFT or FFT takes the form of Real and Imaginary components of the frequency bin vectors. In many applications the magnitude of the frequency domain vectors is required, for example, if the spectrum is to be subsequently displayed. The magnitude (Modulus) of a vector is computed as the square root of the sum of the squares of the real and imaginary parts. This is an additional processing overhead that may well need to be met. It is possible to achieve a good approximation to the square root with a reduced computational effort if the data is known to lie within certain ranges, however, in general this processing requirement may differ substantially from the butterfly arithmetic requirements, and is non-trivial.

In the section on data windowing it was mentioned that the time domain sequence must be periodic in order that a true spectrum be produced. Even with data windowing, the true spectrum is only evaluated to a fairly good approximation, since the discontinuity between beginning and end of the time data sequence is only greatly reduced and not entirely eliminated. If it is expected that the true spectrum of a signal is only varying very slowly, then if the magnitude of the frequency domain is averaged at a corresponding rate, the unwanted effects in frequency domain of the discontinuities in the time domain can be reduced. In addition, noise will tend to whiten, and weak frequency domain components become more predominant. The averaging of frequency domain is thus a very useful function, which although not computationally intensive like the butterfly of the FFT, for example, may be required in a real FFT

system.

Logarithm computations are usually only performed on either large transforms and or frequency domain averaged transforms. They are useful in presenting the transform results because of the normally very high dynamic range of the output. As this may only need to be performed on averaged frequency domain the computational bandwidth may not be very high, however, thought must be given to the inclusion of a "log processor" in a real FFT system, which means that bandwidth matching to the FFT must be carefully considered.

2.5.6. Signal Growth and Data Scaling

From the DFT equation, it is clear that if the number of data samples is equal to N, then a potential signal growth of N times ($\log_2$ N bits) could take place in the case of a unity magnitude time domain signal that was coherent with one of the integer frequencies in the W coefficient matrix. This signal growth would also take place in the FFT if there was no data scaling. A course of action often taken with the DFT, is to use an accumulator of larger bit length than the complex multiplier used to perform vector rotation, and allow signal growth to occur. This simple approach is not possible in the FFT however where the vector rotate and accumulate operations are shared out so that both vector rotator and accumulator must handle the signal at equivalent levels. In the Radix-2 FFT algorithm, the DFT is broken down into 2-point DFT's. Vector-rotation does not change the magnitude of a data vector, however, the 2-point DFT which consists of an add and subtract will introduce a possible signal growth of two (ie one bit of growth). This is the

potential growth therefore which may take place with each Radix-2 butterfly operation. As the complex multiply which is used to perform vector-rotation is a computationally intensive operation, and therefore expensive, data scaling is normally employed at each butterfly to avoid excessive signal growth taking place. This allows optimisation of processor wordlengths. If in the Radix-2 FFT, the output of the butterfly is scaled down unconditionally by one bit then signal growth in the system is held to around zero. The gain of the system is thus held at around unity. This requirement can be implemented in hardware, trivially as a simple shifter, designed to shift down the output of the butterfly.

2.5.7. Noise Considerations in a Practical FFT System

This is a fairly large subject on which a substantial amount of work has already been accomplished, [6] both in the area of theory and practical simulations as there are quite a large number of potential noise sources in an FFT system. The main components are detailed below.

2.5.7.1. Analogue to Digital Conversion Noise

This is not related to the FFT itself of course but is an important noise source in a practical system. The quantisation in the Analogue to Digital conversion and non-linearities can give rise to additional frequency components which show up as harmonics and distortion products. This problem is purely related to A/D design. The FFT is a useful tool in adjusting high precision A/D devices as it allows for these harmonics and distortion products to be

minimised, given a sufficiently pure sine wave as an analogue input.

2.5.7.2. Coefficient Quantisation Noise (W)

The W "twiddle factors" that are required to implement the FFT will typically be stored as a finite length string of binary 2's complement fixed-point data, (defined later in chapter 3, section 5) which is the same as would be used for signal data throughout the transform. Floating point operations may also be used, but in general the above format is considered to be most appropriate in a system that is processing real signals from an A/D converter, since the A/D converter will generate this or a similar digital format.

There are a number of intuative observations that can be made about the W noise and the precision needed to represent the W twiddle factors in a given transform size.

Firstly, the absolute noise introduced into the transform as a whole will depend on the magnitude and nature of the signal data, and how this varies throughout the transform. This will be so because W, is directly multiplied with signal data. Secondly, the precision needed to represent the W twiddle factors will be greater than was required to represent the equivalent W coefficients in the DFT's W matrix and this will crucially depend on transform size. The reason that the W twiddle factors will require a greater precision than the W coefficients of the DFT, are that in the FFT, each W element from the DFT W matrix is effectively synthesised from $\log_R N$ iterations - (one at each pass) - where R is the radix of the transform and N is the transform size. W quantisation in the DFT

itself should also be increased with transform size, however, in order to ensure good angular resolution of all the N W_N^{nk} vectors. Thus in the FFT, W precision must increase with transform size slightly faster than is required by the DFT such that after $\log_R N$ iterations in the same number of passes, the angular resolution of W is still maintained. A mathematical treatment of the W quantisation noise is quite complex [6,21] and it is probably easier to simulate these effects using a digital computer which allows comparison with a near ideal transform using floating point arithmetic. In the passing, it is worth mentioning that signal data itself must also be held at higher precision for larger transforms for similar reasons. This can make the A/D conversion very costly for larger transforms.

2.5.7.3. Roundoff Noise due to Data Scaling

This might be expected to be a relatively large source of noise when compared to the W quantisation noise because data scaling directly affects the main signal path, and indeed, noise resulting from data scaling [22,23] does tend to dominate, given that the W coefficients are held at the same precision as data. As with W quantisation noise, it would be expected that the final passes would contribute most noise since noise from earlier passes is scaled down along with the signal in any data scaling operations that may have taken place. This noise contribution can be avoided altogether by allowing signal growth to take place. This is undoubtedly the most desirable approach when the extra cost of having higher dynamic range hardware is considered acceptable. This approach may entail a bandwidth penalty, however, particularly if bit-serial arithmetic

processing is employed as the word transfer rate is inversely proportional to the word length used.

2.6. Review of Current Devices

2.6.1. Single Chip FFT Processors

Recent trends in using redundancy with self-test and repair techniques have allowed the possibility of fabricating a fully pipelined FFT processor on a wafer of silicon such as the 16MHz, 16-point dedicated FFT processor [24] using yield enhancement techniques as described by Garverick and Pierce. In this context, pipelined refers to a level of arithmetic concurrency that allows a constant stream of data to flow non-recursively from one arithmetic unit to the next. This is possible where there is one arithmetic unit for each pass in the transform as described earlier in this chapter. It is likely that wafer scale integration will provide the possibility of even more powerful FFT systems in the future. It is not so easy, however, to apply these sort of yield enhancement techniques to any arbitrary architecture, indeed the approach should really be to devise an architecture based on a particular yield enhancement approach rather than try to force yield enhancement on to an existing architecture which does not employ yield enhancement.

It is possible to fabricate dedicated single chip FFT processors without using yield enhancement provided the arithmetic unit is limited in size such as the 32-point FFT processor chip described by Covert [25] which uses a single arithmetic unit. The limited transform size (32-point) is indicative of the substantial memory

and complex control requirements, needed to implement the FFT. Larger transforms can be built up from this basic 32-point transform. The joining up process requires an external complex multiplier to implement the requirement for vector rotation.

The advantages of designing a single chip FFT are much the same as for integrating any digital system. There is the possibility of optimising the speed of each section which in the FFT, includes memory, arithmetic and control. Also, system power consumption can be reduced as high bandwidth memory accesses are contained within the chip. There are, however, more subtle advantages that can be derived from integrating a system such as the FFT. Such advantages can be derived from analysis of the interface between the various sections of the system. For example, rather than hold coefficient in ROM as an actual binary numerical representation of the coefficient vector, it is possible to pre-compute and store in ROM the resulting control logic that would be presented to a Booth's [26] algorithm multiplier as a function of the numerical coefficient vectors. The requirement for on chip Booth's control logic hardware can thus be bypassed by storing the coefficient vectors in the more abstract form of a Booth's multiplier control logic word. As well as reducing gate count, this also reduces power consumptions and propagation delays. This technique was used in Coverts, 32 point FFT processor chip. It is important to consider that holding the coefficient in a more abstract form like this, makes the system more difficult to understand and therefore de-bug or modify. Despite these disadvantages, the general tend in VLSI designs of this dedicated nature has been to adopt higher and higher levels of abstraction. As a general

comment, these trends in dedicated VLSI designs are likely to continue into the future being made easier by the growth in methods and tools for coping with system level abstractions. As dedicated chip design involves some degree of abstraction in the optimisation of the hardware it might be asked if the case for general purpose hardware design is thus strengthened since this involves lower degrees of abstraction at a hardware level by passing system abstractions to the programmer instead. This indeed is the great opportunity presented by VLSI; that system abstractions can be moved more and more into software, thus allowing the hardware designers to concentrate on architectures that lend themselves to fault tolerance, self repair and expandability.

It is not surprising therefore that the general purpose digital signal processor with on chip RAM, is becoming more common. These sort of devices are quite fast at performing the FFT however memory availability usually limits transform sizes to around 64 points such as with the fast Texas Instruments TMS320 processor and the VLSI programmable signal processor [27] described by McWilliam. Both of these processors take around 1-2 milliseconds to compute a 64 point FFT, however the TMS320 can compute a transform in 0.7 milli-seconds if program loops are repeated in memory, thus avoiding instruction branching. It would normally take 1.5 milli-seconds using nested loops. Such general purpose processors, however, are usually substantially slower than can be obtained using dedicated hardware, as the arithmetic is not usually geared to operating on complex data, and consequently cycles are "burnt up", in transferring double length complex data between storage and processing areas, on chip.

Also, of course, a general purpose unit would require extra cycles for the decoding of each instruction. This requirement would typically be bypassed in a dedicated processor although the technique of pipelining the instruction fetch, decode, and execute can overcome speed problems in general purpose units with only branch instructions being left rather slow.

2.6.2. FFT Arithmetic Processors

In chapter 1, it was shown how the FFT algorithm involves the repetition of a complex arithmetic function, very often known as the "butterfly". The precise butterfly function depends exactly on the particular FFT algorithm, but will involve complex multiplication and addition when using an x,y (Cartesian) coordinate system. It has already been mentioned that the FFT butterfly is highly suited to VLSI implementations and there are a number of butterfly processor chips currently available, indicating the usefulness [28] of this partition. This section will look at current arithmetic processors relevant to the FFT and also discuss developments in multiplier technology which is also relevant to the FFT's arithmetic requirements.

Of the number of single chip FFT butterflies, most commercial devices are aimed at Radix-2 or 4 systems and employ parallel arithmetic in internal operation. Whilst bit-serial butterfly processors allow concurrent multiplications on chip, thus reducing control requirements, it seems that parallel processor design figures more prominantly than bit-serial. This is most likely because the design for parallel arithmetic multiply and accumulate chips, usually

already available as a separate product by the large manufacturers can be adapted to compute the FFT butterfly which can then be sold as a separate product. Lyon, [3] however, advocates bit-serial techniques for dedicated digital signal processing (DSP) systems that can be pipelined such as the DFT and FFT. The advantages of this approach appears to be lower pin counts, and easy bandwidth matching, since data flows as a constant stream of serial data through the system. Connections between chips are reduced and therefore system costs are reduced. Bandwidths can be surprisingly high, due mainly to the high level of pipelining, although clearly the rate at which word data is passed is now much more dependent on wordlength. Perhaps the most important feature of the bit-serial methodology is however, the ease and efficiency with which interconnections can be made between different processing elements on chip. This feature can be used to achieve efficient auto-layout and connection of processing elements such as is used in the FIRST [18] silicon compiler. The bit-serial approach is probably best suited to custom designs,(which might involve silicon compilation) and systems which cannot strictly be regarded as general purpose (ie relatively dedicated, though not necessarily non-reconfigurable), however, the parallel arithmetic approach is perhaps more appropriate for general purpose uses because of the higher bandwidths, and suitability for recursive algorithms. Other butterfly FFT chips make use of bit slice techniques to allow various wordlengths to be achieved. Such devices include a serial-parallel 4 bit ECL bit-slice processor and an 8 bit bipolar bit-slice processor (SN74AS888), which operates at around 20 MHz. The bit-slice approach appears at first sight, to be a sensible one, however, it does have serious drawbacks. The time

to propagate carries forward is the limiting factor in all these bit-slice processors and taking a carry-out off one chip and onto another chip involves extra delays due to the extra buffering required. These times add up, resulting in a substantially degraded carry ripple-through time compared with a fully integrated processor. In addition, these carry-out nodes will be expected to operate at high bandwidths, increasing dynamic power consumption substantially. As processing technologies shrink feature sizes further, the costs in taking signals off chip become proportionately greater. It is reasonable therefore to conclude that bit-slice techniques are a remnant from SSI and MSI which have no place at all in LSI and VLSI designs.

2.6.3. Parallel Digital Multiplier Devices

The digital multiplier is particularly relevant to the computation of the DFT and FFT, as it allows the complex multiply function to be readily computed. Most of the parallel multipliers described in literature are non-pipelined (no latching of data internal to the actual multiplier except possibly at the input or output) and operate therefore only in a unity latency configuration (the output appears a single clock cycle after the input). Such devices can be used in recursive arithmetic configurations as well as non-recursive. One example of recursive operation is the active computation of a rotating coefficient vector from a fixed vector, by using feedback on a unity latency complex multiplier. The fastest multiplier technologies currently appear to be Gallium Arsenide (GaAs) technologies, such as the 16 x 16 bit multiplier with 10.5 nS

propagation delay [29] described by Nakayama et al. This technology is so fast, that an attempt to pipeline the multiplier design would have been counter productive since the distribution of high integrity clocks could not be efficiently realised at these speeds. This multiplier would require to be interfaced with a memory devices of the same technology, such as the 1Kbit 4 nS access time GaAs device described by Yokoyama [30] which is fast enough to permit this multiplier to operate at its maximum clocking rate.

Zero static power dissipation technologies are best suited to parallel ripple through multiplier designs, as they consume power only during logic transitions. Thus, bulk cMOS and cMOS-SOS offer very low power consumption figures. Table (2.6.3.1) shows a number of high speed multipliers that represent a high level of performance.

Maker	Device Number	Availability	Word Length	Latency	Technology	Speed (nS)	Power (mW)
TRW	MPY12HJ	Yes	12 x 12	Unity	Bipolar	80	3000
TRW	MPY16HJ	Yes	16 x 16	Unity	Bipolar	100	4500
TRW	MPY24HJ	Yes	24 x 24	Unity	Bipolar	200	5000
GEC	/	No	16 x 16	Unity	cMOS-SOS	250	40
FUJITSU	/	No	16 x 16	Unity	GaAs	10.5	952

cont..

cont..

Lerouge	ESSCIRC83	No	16 x 16	Unity	nMOS	120	200

Table (2.6.3.1)

More details of the above TRW devices may be found in [31] and the nMOS device is described fully in both [32,33] which describes an interesting speed enhancement technique.

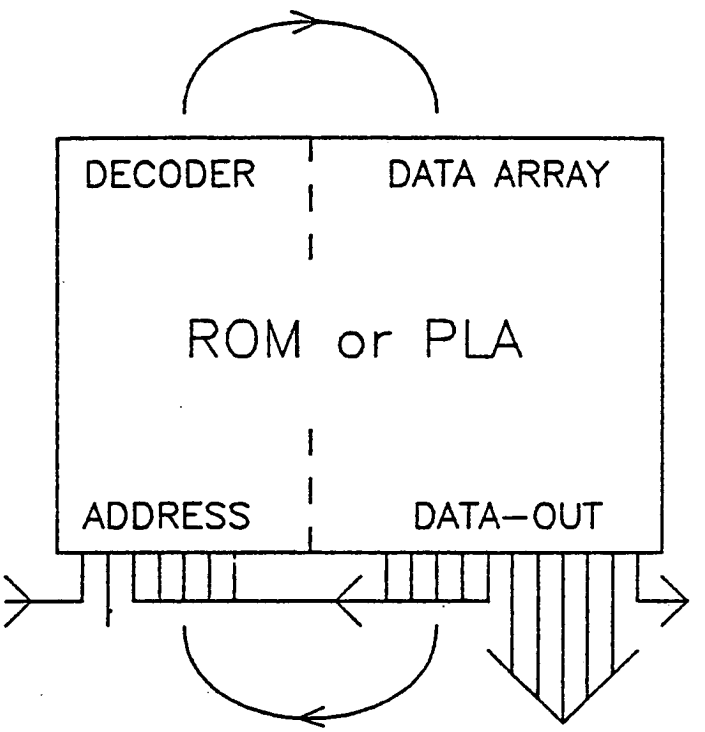
2.7. FFT Control Chips

Random Access Memory will typically be used for data storage in the FFT, so that some means of addressing this memory is required. It is apparent that serially based memory storage can also be used in the computation of most FFT's, [6] but this tends to degrade system versatility and should therefore be avoided unless a given Random Access Memory (RAM) is too slow. In the special case where data can be stored serially, a serial memory will generally offer a higher bandwidth than a RAM. As well as addressing this memory, read/write control signals will be required and the arithmetic unit may have its own special control requirements such as data scaling for example.

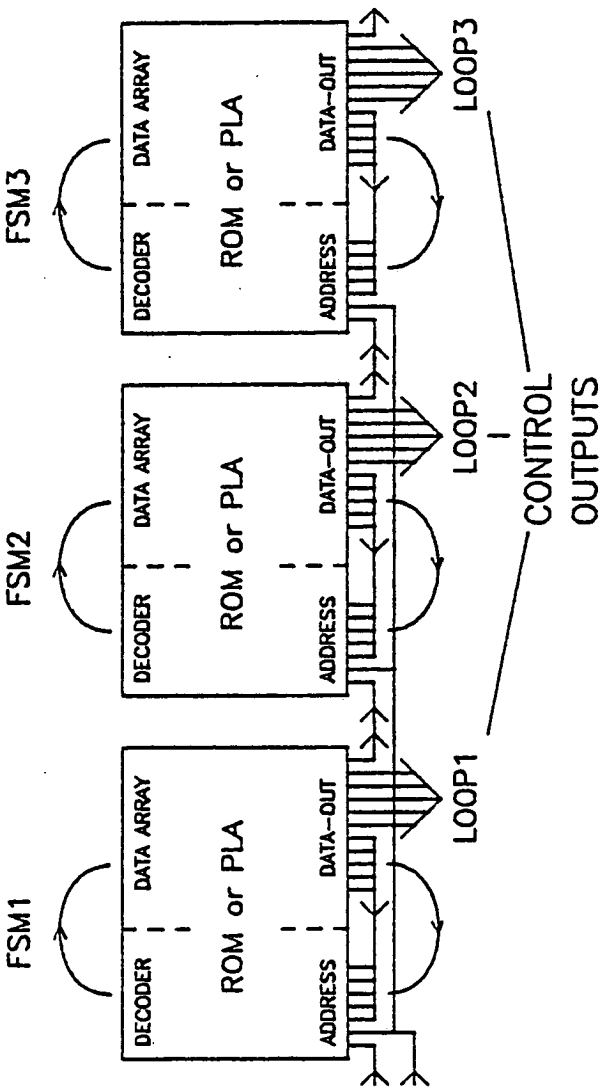
The control unit must offer some flexibility for the system designer, and should therefore either be programmable or be produced by techniques such as silicon compilation which might be required if a general purpose unit was too slow or inefficient. Some manufacturers produce FFT chip sets which include FFT address generation. The AMD29540 FFT control sequencer is one example.

2.7.1. General Purpose Control Units

The general purpose control unit is one which is programmable and will be RAM or ROM based. The memory will contain all or some of the state addresses of the FFT and will output the appropriate control data associated with each address. Each word in the memory will comprise a state address, (used to point to the next state address) and a control word which is read out at each cycle. Part of the data output of the memory is therefore fed back into the address input in order to implement a finite state machine (FSM). Figure (2.7.1.1) shows a typical memory configuration for implementing a FSM. The programmer, must ensure that each state address that is output from the memory, points to an existing and correct address present in the memory or the state loop will reach a "dead end". An important feature of the FSM, is that it is possible to include several state loops in the memory and thus data or external control can be applied to leap between one loop and another loop without any time penalties. (The other loop may involve only one single state, feeding back on itself, thus holding the FSM output at some fixed value.) One problem, however, is that it cannot handle loops within loops without repeating the inner loops for as long as they must appear. This is because the FSM can only recognise a single state at any one time and not several states associated with every loop. Thus, massive redundancy, would be involved in using a single FSM controller in computing the FFT, for example. This can be overcome by using a single FSM for each loop required in the control sequence as shown in Figure (2.7.1.2). It should be noted, however, that even in a single loop, there can be hidden loops which could be



Figure(2.7.1.1.) Memory or a Programmable Logic Array may be Configured as a Finite State Machine.

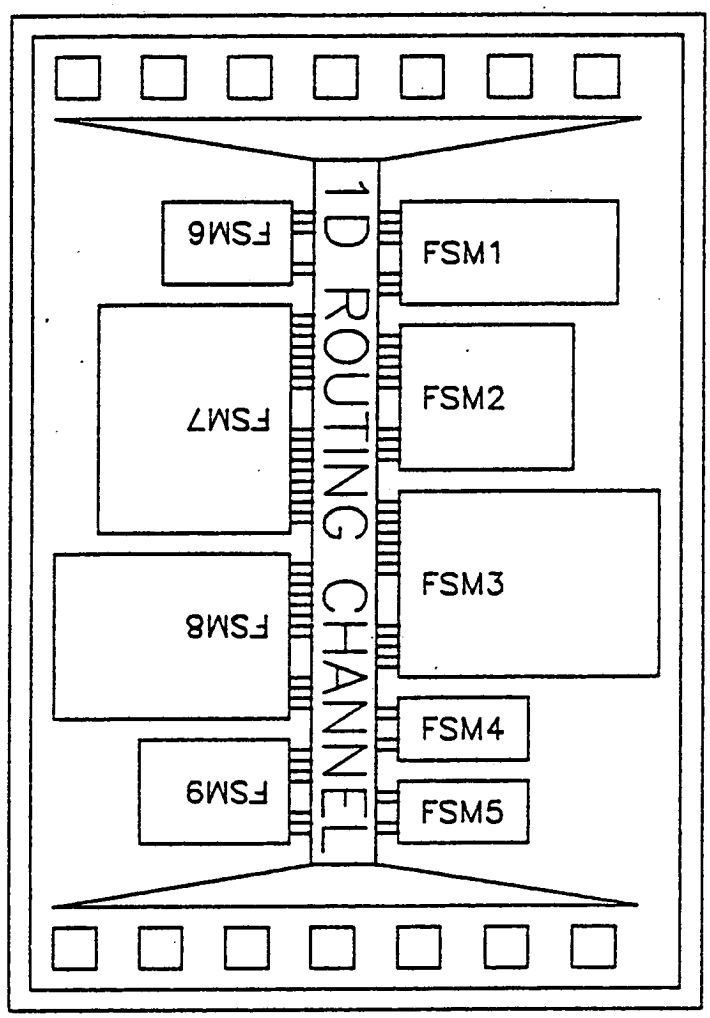


Figure(2.7.1.2.) The use of several Finite State Machines to Generate Control from a Nested Loop.

implemented more efficiently by using one or more extra FSM's. The decade counter is one simple example of a FSM which could be broken down into four binary counter FSM's. It is clear then, that a general purpose control unit, must have as many FSM's as there are likely to be loops, and each FSM must have a large enough memory to hold all the states required for each loop. In practise, if the number of loops within loops is not very great, the decision to implement the whole control sequence with a single FSM with some redundancy, may be made by the system designer as this would invariably result in fewer system components. It is clear that a silicon compiler which could code up, place and interconnect FSM's as determined by a simple input language, would allow the size and number of the FSM's to be tailored to a specific control problem particularly efficiently.

2.7.2. Silicon Compilation for FFT Control Units

The standard and well proven structure used to implement FSM's, is the Programmable Logic Array (PLA) [5] described in detail by Mead and Conway. The PLA is essentially a ROM except that only a fraction of the 2^N possible combinations of the N bit input address are actually decoded as only these input states need to be considered. In such cases the use of a ROM would be wasteful. (This means that it is possible to find some input address word which will not be decoded and therefore produce no meaningful result.) The PLA can be constructed entirely from NOR type logic gates and is thus highly suited to technologies such as nMOS which employ ratioed logic gates.(Gate pull up resistance must be ratioed with worst-case



Figure(2.7.2.2.) Generalised Floorplan showing the Constrained Architecture that is suited to Silicon Compilation. A One-Dimensional Routing Channel Connects a number of Finite State Machines Together.

pull down resistance to achieve satisfactory logic low.) The PLA is also a highly regular structure, and is thus well suited to silicon compilation techniques. With a "PLA generator", it is possible to convert Boolean Logic directly into silicon layout. Such a program would also be capable of automatically feeding back some of the outputs of the PLA to produce a FSM. The regularity of the PLA, however, also makes it possible to estimate its size very accurately and simply, making it easy to feed accurate information to placement software. Ultimately, it would be possible then, to write a program which could read some high level language, describing a control task, and place a number of PLA based Finite State Machines (FSM's) along a one Dimensional routing bus which would handle all the connections automatically. Figure (2.7.2.2) shows a typical floorplan that such a compiler might produce.

Performance estimation and control are important in silicon compilers. The possibility of devising a speed programmable structure such as the PLA was considered, and some software was written to assess whether it would be possible to achieve a large enough degree of control to be useful. This work and its results are shown in appendix 1.

2.8. Special Memories for the FFT

There is an extensive range of general purpose RAM chips available, and as a consequence of this there is a general lack of special purpose memories, geared to FFT processing. There are, however, some FFT system memory configurations such as the swinging buffer, which, if implemented using standard RAM components, result

in very high chip counts owing to the extra components needed to switch data and address from one memory device to its neighbour. There is a strong case for a case for a swinging buffer type memory or alternatively a special twin memory which could execute a flash load of one memories contents (from an A/D for example) into another memory (for FFT processing for example) as time and frequency domain buffering is essential to implementing the FFT in a real time environment. The use of associative (content addressable) memories may provide easier multiprocessor FFT system design as discussed earlier in section 4. Multiport memory (where more than one data write and/or access can take place simultaneously) may be useful in bandwidth enhancement of the FFT. Certain serial memory architectures (shift register based) may offer very high speeds in those FFT algorithms that allow for some degree of serial data storage (most FFT algorithms). Programmable, tapped shift register with parallel load facilities seems to be particularly attractive in this respect, particularly as yield enhancement is fairly trivial with this sort of memory by employing simple bypassing and redundancy techniques.

In general, however, standard RAM can usually be configured to suit most FFT system architectures that have so far been discussed in literature, and it is not therefore proposed to pursue this aspect of the FFT's system requirements further.

2.9. Summary

The memory and control requirements of the FFT involve the design of fairly general purpose hardware using techniques which are difficult to improve on. The FFT arithmetic requirement is however,

quite significant, involving vector rotation and addition at very high data rates and is independent of the degree of concurrency in the system or the size of transform to be computed. This processing unit known as the "butterfly" is an ideal candidate for VLSI as it does not impose any major restrictions on the system designer other than the normal word-length restrictions experienced with any digital processor.

The next chapter will consider ways of streamlining the butterfly arithmetic requirements by using conventional arithmetic and also distributed arithmetic techniques.

References

1. E. Kreyszig, Advanced Engineering Mathematics, Wiley (1972 (3rd Edition)).

2. F. J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," Proc. IEEE, Vol. 66, No.1, (January 1978).

3. R. F. Lyon, "A Bit-Serial VLSI Architectural Methodology for Signal Processing," VLSI'81, pp. 131-140 (August 1981).

4. J. L. vanMeerbergen and F. J. vanWyk, "A 2 Micron NMOS 256-Point Discrete Fourier Transform Processor," Thirtieth International Solid State Circuits Conference, pp. 124-125 (23-25 February 1983).

5. C. Mead and L. Conway, Introduction to VLSI Systems, Addison Wesley (1980).

6. L. R. Rabiner and B. I. Gold, Theory and Application of Digital Signal Processing, Prentice Hall (1975).

7. J. M. Cooley and J. M. Tukey, "An Algorithm for Machine Computation of Complex Fourier Series.," Math Comp., pp. 297-301 (April 1965).

8. E. Dubois and A. N. Venetsanopoulos, "A New Algorithm for the Radix-3 FFT," IEEE Trans. ASSP, Vol. ASSP-26, pp. 222-225 (June 1978).

9. S. Prakesh and V. V. Rao, "A New Radix-6 FFT Algorithm," IEEE Trans. ASSP, Vol. ASSP-29, pp. 939-941 (August 1981).

10. L. R. Rabiner and C. M. Rader, Digital Signal Processing, IEEE Press (1972).

11. T. E. Curtis and J. T. Wickenden, "Hardware-Based Fourier Transforms: Algorithms and Architectures," Proc IEE Part F, CRSP, Vol. 130, pp. 423-432 (August 1983).

12. W. C. Siu and B. S. Chen, "New Realisation Technique of high-speed discrete Fourier Transform Described by Distributed Arithmetic," IEE Proceedings, Vol. 130, Part E No.6, pp. 177-182 (November 1983).

13. M. A. Jack, D. G. Park, and P. M. Grant, "CCD Spectrum Analyser using Prime Transform Algorithm," Electronic Letters, Vol. 13 No. 15, pp. 431-432 (21st July 1977).

14. N. Kapur, J. Mavor, and M. A. Jack, "Convolution Architectures for Spectrum Analyses employing CCD Programmable Transversal Filters," Int. J. Electronics, Vol. 49, No. 2, pp. 131-146 (1980).

15. R. W. Broderson and P. R. Gray, "The Role of Analog Circuits in Future VLSI Technologies," Digest of Technical Papers ESSCIRC'83, pp. 105-110 (September 1983).

16. T. T. Dao and M. Davio, "Complex Number Arithmetic with Odd-Valued Logic," IEEE Trans. on Computers, Vol. c-29, No.7, pp. 604-611 (July 1980).

17. J. G. Tront and D. D. Givone, "A Design for Multiple Valued Logic Gates Based on Mesfets," IEEE Trans. on Computers, Vol. c-28, No.2, p. 854 (July-December 1979).

18. P. B. Denyer, D. Renshaw, and N. Bergmann, "A Silicon Compiler for VLSI Signal Processors," ESSCIRC '82 Digest of Technical Papers, pp. 215-218 (September 1982).

19. R. G. Garside, The Architecture of Digital Computers. 1980.

20. E. O. Brigham, The Fast Fourier Transform, Prentice Hall (1974).

21. A. V. Oppenheim and C. J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," Proc. IEEE, Vol. 60, No.8, pp. 957-976 (August 1972).

22. T. Kaneko and B. Liu, "Accumulation of Round-Off Errors in Fast Fourier Transforms," J. Association Comp. Mach., Vol. 17, No.4, pp. 637-654 (October 1970).

23. P. D. Welch, "A Fixed-Point Fast Fourier Transform Error Analyses," IEEE Trans. on Audio and Electroacoustics, Vol. AU-17, No.3, pp. 151-157 (June 1969).

24. S. L. Garverick and E. A. Pierce, "A Single Wafer 16-Point 16MHz FFT Processor," Proc. 1983 CICC, (Rochester, New York, May 1983).

25. G. D. Covert, "A 32 Point Monolithic FFT Processor Chip," Proceeds of the International Conference on Acoustics, Speech and Signal Processing, pp. 1081-1083 (1982).

26. D. A. Booth, "A Signed Binary Multiplication Technique," Q. J. Mech. Appl. Maths, Vol. 4, pp. 236-240 (1951).

27. A. J. McWilliam, "An Architecture and Design Approach for Programmable Digital Signal Processor VLSI Chip," IEE Procds. on The Impact of High Speed and VLSI Technology on Communication Systems, pp. 67-72 (November 1983).

28. R J. Karwoski, "A Four-Cycle Butterfly Arithmetic Architecture," LSI Publication TP9-4/80, TRW LSI PRODUCTS 2525 E.El Segundo Blvd. El Segundo, CA 90245 (213) 535-1831, (April 1981).

29. Y. Nakayama and K. Suyama, "A GaAs 16*16b Parallel Multiplier using Self-Alignment Technology," Thirtieth International Solid State Circuits Conference, pp. 48-49 (23-25 February 1983).

30. N. Yokoyama and T. Ohnishi, "A GaAs 1K Static RAM using Tungsten-Silicide Gate Self-Alignment Technology," Thirtieth International Solid State Circuits Conference, pp. 44-45 (23-25 February 1983).

31. "Digital Filters and Spectral Analyses," Integrated Computer Systems ICS Publishing Co. (UK) Ltd. Pebblecoombe, Tadworth Surrey KT20 7PA England, pp. 2-4-0

32. C. P. Lerouge, P. Girard, and J. S. Colardelle, "A Fast 16 Bit NMOS Parallel Multiplier," IEEE Journal of Solid State Circuits, Vol. SC-19, No. 3, (June 1984).

33. C. Lerouge and P. Girard, "A Fast 16 Bit NMOS Parallel Multiplier," Ninth European Solid-State Circuits Conference, pp. 29-32 (21-23 September 1983).

Chapter 3 - Algorithms for High Bandwidth Vector Arithmetic

3.1. Introduction

This chapter aims to review various digital approaches to performing high bandwidth vector arithmetic, in particular, vector rotation which is a dominant arithmetic requirement in the computation of the DFT and FFT.

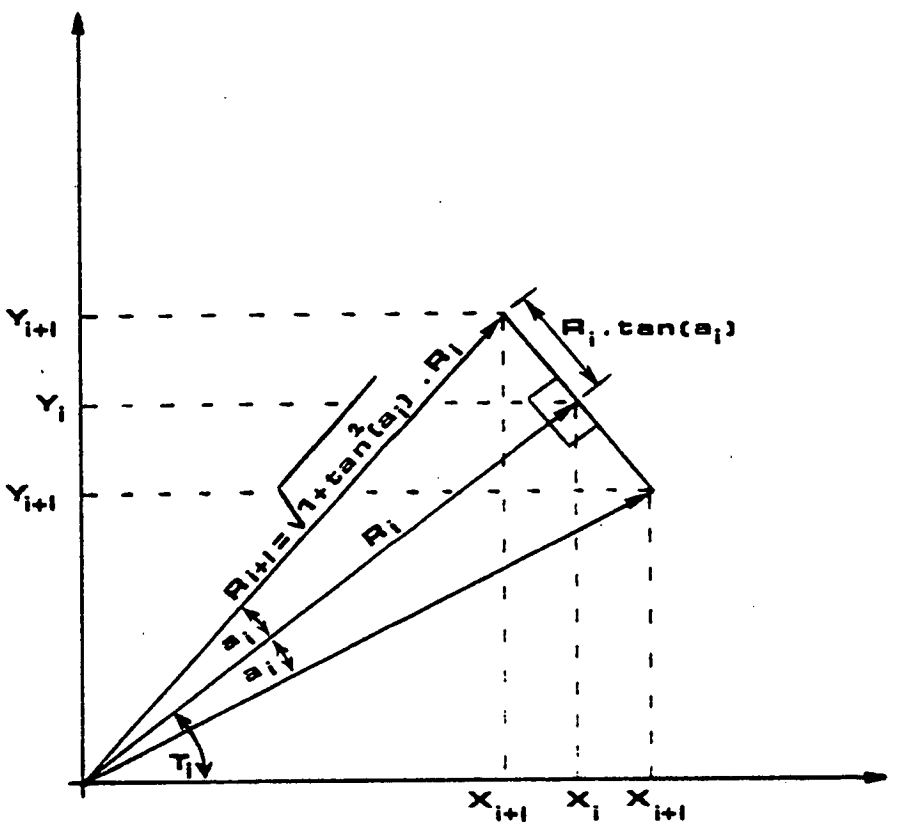
The chapter will start by discussing the CORDIC approach and then move on to look at the complex multiply as a means of performing vector rotation. Various ways of computing this function using real multipliers are discussed. The chapter ends by looking at distributed arithmetic techniques for computing small DFT's directly and also the complex multiply.

3.2. CORDIC Arithmetic Approaches

Although vector rotation can be achieved trivially using addition in polar coordinates for example (the magnitude of the input vector remains unchanged and its new angle is computed by adding the old angle to the rotation angle), vector addition involves trigonometric functions which, ordinarily, would represent a high computational load. Techniques have been developed for efficiently computing trigonometric functions such as used in COordinate Rotation DIgital Computers, (or CORDIC's for short) which allows vector rotation and addition operations to be performed more efficiently than would normally be possible. This technique allows for two basic modes of operation [1] as described by Volder. In the first of these modes, the Rotation mode, the coordinate components of a

vector are modified by an angle of rotation to produce coordinates which have been rotated by that angle. In the second mode, the Vectoring mode, coordinate components of a vector are returned in the form of magnitude and angle. CORDIC arithmetic is sufficiently general purpose in nature to allow vector rotations to take place either on hyperbolas, circles, (as is relevant to the DFT and FFT), or lines.

The rotation of a vector using CORDIC arithmetic is based on the concept of realising a variable rotation as a step-by-step series of pseudo rotations. The angles of these rotations may be chosen to be such that each pseudo rotation step may be computed using only binary shift and add operations. These special angles form a set from which any desired angle of rotation can be built up iteratively. In order to specify this set of angles, it is necessary to consider a typical pseudo rotation as shown in Figure (3.2.1). This shows a vector of magnitude R_i at angle T_i to the X-axis being rotated by either $+ a_i$ or $- a_i$. As well as being rotated, a small increase in the magnitude of the new vector results. This may be calculated by using standard trigonometrical relationships to evaluate the length of the side of the right angle triangle that is opposite to angle a_i. This side has length $[R_i \tan(a_i)]$. The theorem of Pythagoras may then be used to evaluate the new magnitude of the rotated vector. This is equal to $[\mathrm{Sqrt}(1 + \tan^2(a_i))].R_i$. Where "Sqrt" is short for the square root of whatever follows in brackets. Trigonometric rules can then be applied to the two right angled triangles of angle $(T_i + a_i)$ and $(T_i - a_i)$ to the X-axis, to produce an expression for the two values of Y_{i+1}. This is as shown in equation



A CORDIC Pseudo Rotation

when $a_i = \tan^{-1} 2^{-(i-1)}$

$i>1$

Figure(3.2.1.) A Typical "CORDIC" Pseudo-Rotation.

(3.2.1).

$$Y_{i+1} = [\mathrm{Sqrt}(1 + \tan^2(a_i)].R_i.\sin(T_i +/- a_i) \qquad (3.2.1)$$

This is equivalent to

$$[\mathrm{Sqrt}(1/\cos^2(a_i))].R_i.[\sin(T_i).\cos(a_i) +/- \cos(T_i).\sin(a_i)] \qquad (3.2.2)$$

From trigonometric relationships, it may be noted that

$$X_i = R_i.\cos(T_i) \qquad (3.2.3)$$

and similarly

$$Y_i = R_i.\sin(T_i) \qquad (3.2.4)$$

It follows then, that

$$Y_{i+1} = [1/\cos(a_i)].Y_i.\cos(a_i) +/- X_i.\tan(a_i) \qquad (3.2.5)$$

which implies that

$$Y_{i+1} = Y_i \quad +/- \quad X_i.\tan(a_i) \qquad (3.2.6)$$

a similar expression can be derived for X_{i+1} which is

$$X_{i+1} = X_i \quad -/+ \quad Y_i.\tan(a_i) \qquad (3.2.7)$$

The fundamental principle upon which the CORDIC computing technique is based, is that if angle a_i is chosen to be such that its tangent is the reciprocal of some power of 2, then equations (3.2.6) and (3.2.7) which describe a pseudo rotation, can be computed using only shift and add operations. The set of angles therefore which may be considered are described by equation (3.2.8).

$$a_i = \tan^{-1} 2^{-(i-2)} \quad \text{where} \quad i = 2,3,4,5,6... \qquad (3.2.8)$$

Where the number of rotations by angle a_i is chosen to be large enough to produce the required accuracy of rotation. The case of i=1 has been excluded to allow the special case of a_i = 90 degrees for which the tangent cannot be expressed. In this special case, Y_2 = +/- X_1 and X_2 = -/+ Y_1. This step is unique, in allowing a perfect rotation, with no alteration in the magnitude of the vector.

Although the growth in the magnitude of the rotated vector is unavoidable, it can be kept to a constant by imposing the condition that at each pseudo rotation step, there may be no zero-rotations allowed (steps may not be ignored). That is, a decision to rotate by either + a_i or - a_i must be made at each step. If this rule is adhered to, then for a given number of pseudo rotations, the growth in the magnitude of the resultant vector is held constant. One disadvantage of this unavoidable signal growth is that unity magnitude twiddle factors may not be used in the FFT butterfly, so that

both data inputs to the butterfly must be passed through the CORDIC hardware even though only one of the inputs actually needs to be rotated. This makes this approach rather inefficient for computing the FFT butterfly. The fact that rotations can be computed using only shifts and additions makes the CORDIC approach attractive on machines which do not offer fast digital multiplication. This is true of most microprocessors that are not geared to digital signal processing. It is not proposed to cover the mathematical aspects of CORDICS any further here as this is a large subject area and has already been suitably covered by the previous reference [1] as well as [2,3,4,5,6] and more recently [7,8]. Instead it is hoped to summarise the suitability of the CORDIC approach for VLSI implementations of the DFT and FFT.

Work on specific VLSI implementations of CORDIC hardware such as that reported by Maxwell [9] give a good indication of the overall hardware requirements. The CORDIC architecture can be viewed as a controlled datapath, (frequently organised as two or three datapaths - one for each iterative loop) where the datapath is somewhat larger and more complex than a general purpose one. Maxwell for example used a datapath which contained two adder/subtractors, three variable barrel shifters, (two left and one right), two ROMS, five 2:1 multiplexers, one 4:1 multiplexer, two registers and other logic for operation on sign bits. This datapath must be regarded as somewhat special purpose compared with general purpose hardware datapaths, and does not point to a highly regular VLSI structure making interface with the control unit area inefficient. Also the overall CORDIC hardware requirement is quite large due to control

requirements and the intricate datapath that is required. It would appear then, that the CORDIC approach may not be ideally suited to VLSI implementations, where digital multiplication can be achieved at low costs and low power consumptions due to the relatively simple control requirements.

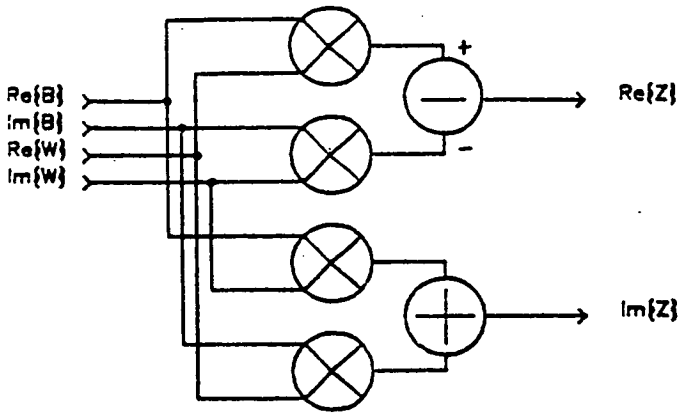
CORDIC's would thus appear to offer a useful general purpose approach to the computation of trigonometric functions, however, vector rotation is probably more conveniently described in a Cartesian coordinate system where absolute angles are not specified. This will now be investigated.

3.2.1. Vector Rotation in a Cartesian Coordinate System

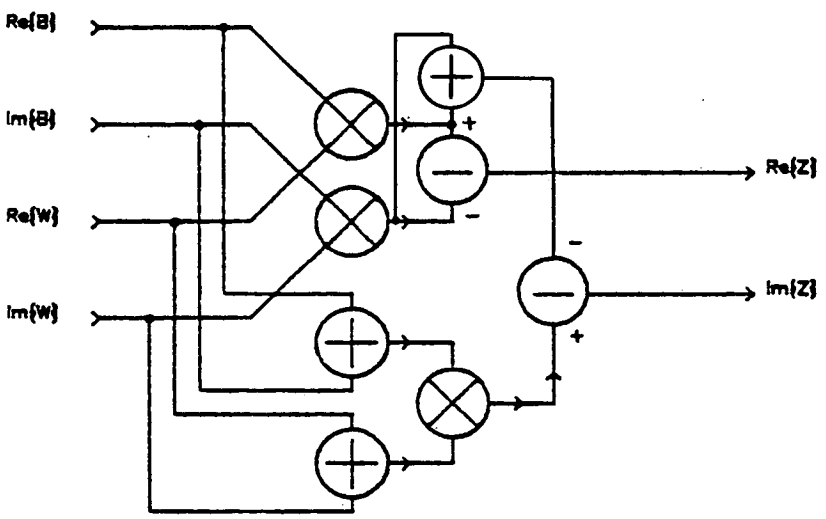
Vector rotation can be conveniently described in a Cartesian coordinate system using the trigonometric relation described in equations (3.2.1.1) and (3.1.2.2). In this formulation of the rotation function it is not necessary to compute sines or cosines of angles with both data and coefficient stored in this form at all times. These equations describe how the sine and cosine of the sum of two angles can be expressed as the sum or difference of the multiple of the sine and cosine terms of the individual angles. Moreover with this coordinate system, vector addition is equivalent to ordinary addition which is relatively trivial.

$$\cos(x+y)=\cos(x).\cos(y)-\sin(x).\sin(y) \qquad (3.2.1.1)$$

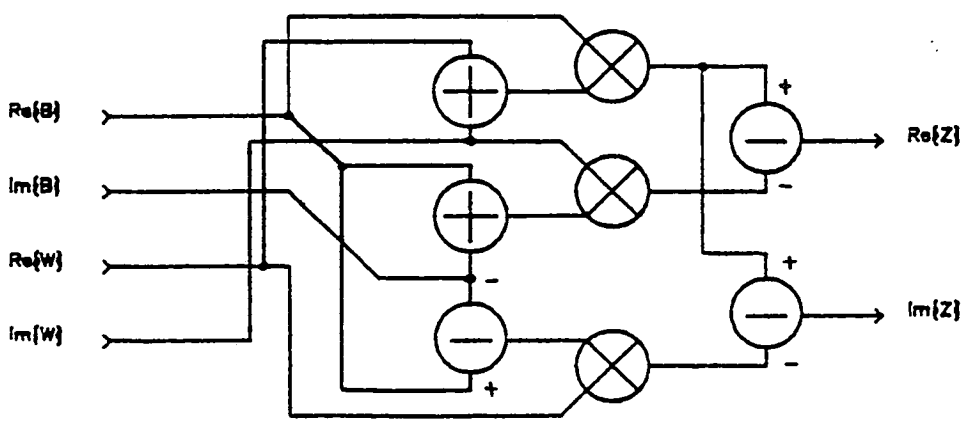
$$\sin(x+y)=\cos(x).\sin(y)+\cos(y).\sin(x) \qquad (3.2.1.2)$$



Figure(3.2.1.1.) Direct Computation of the Complex Multiply



gure(3.2.1.2.) The Complex Multiply Implemented Using Three Real Multiplications and Four Signed Additions.



Figure(3.2.1.3.) The Complex Multiply Implemented Using Three Real Multiplications and Five Signed Additions.

If the x and y axis are used to represent the real and imaginary dimensions of the complex plane and it is desired to rotate an input data vector (B) by the angle of some coefficient vector (W) to form a resultant vector (Z), then the equations which describe this rotation are simply as shown in equation 3.2.1.3 and 3.2.1.4. (If only a rotation of (B) is to be performed, with no alteration to its magnitude, then the magnitude of (W) must be unity.) It can be seen that four multiplies and two (signed) addition operations must be performed.

$$Re(Z)=Re(W)Re(B)-Im(W)Im(B) \qquad (3.2.1.3)$$

$$Im(Z)=Re(W)Im(B)+Re(B)Im(W) \qquad (3.2.1.4)$$

This might be computed by using a single multiplexed multiplier with an accumulator or by using distributed arithmetic techniques which are discussed in the next section. The total number of multiplies required to compute the above two equations can, however, be reduced from four to three with a small increase in the number of additions [10,11] as described by Golub (Golub's method is described in a footnote) and Buneman. This reduction in the number of multiplies is achieved in both cases by expressing parts of equations (3.2.1.1) and (3.2.1.2) as a product of sums and not just as a sum of products. This yields a common term in both equations, as shown in (3.2.1.5),(3.2.1.6) and similarly in (3.2.1.7),(3.2.1.8).

$$Re(Z)=Re(B)Re(W)-Im(B)Im(W)(as(3.2.1.3)) \qquad (3.2.1.5)$$

$$Im(Z)=(Re(B)+Im(B))(Re(W)+Im(W)) \qquad (3.2.1.6)$$

$$-[Re(B)Re(W)+Im(B)Im(W)]$$

and also

$$Re(Z)=Re(B)(Re(W)+Im(W))-Im(W)(Re(B)+Im(B)) \qquad (3.2.1.7)$$

$$Im(Z)=Re(B)(Re(W)+Im(W))-Re(W)(Re(B)-Im(B)) \qquad (3.2.1.8)$$

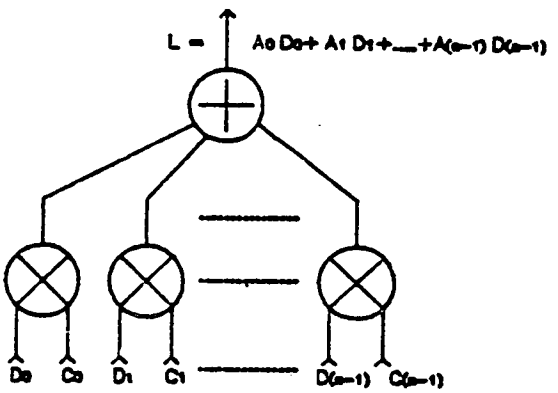
Further reductions can be achieved if the vector coefficient Re(W) and Im(W) is also stored as (Re(W)+Im(W)) in ROM, so that one less addition is required per complex multiply described above. Thus (Re(W)+Im(W)) would not be actively computed. This, however, is not a highly significant saving and would only be chosen if ROM was readily available. In summary of the above approaches to performing vector rotation using real multipliers and adders, Figure (3.2.1.1) shows the conventional implementation of the complex multiply based on (3.2.1.1) and (3.2.1.2), whilst Figures (3.2.1.2) and (3.2.1.3) show implementations based on equations (3.2.1.5) to (3.2.1.8). It should be noted that the dynamic range requirements of the hardware implementations in Figures (3.2.1.2) and (3.2.1.3) is slightly greater than that of the conventional complex multiply shown in Figure (3.2.1.1) to the extent of one extra bit of precision being required at some points in the computation. This is particularly inconvenient in bit-serial implementations where the extra bit of wordlength reduces the word transfer rate. Although these reformulations of the complex multiply are algorithmically slightly more efficient than a direct computation, they do not point to any

specific VLSI architectures. On a digital computer the direct approach might even be faster if hardware multiplication was employed.

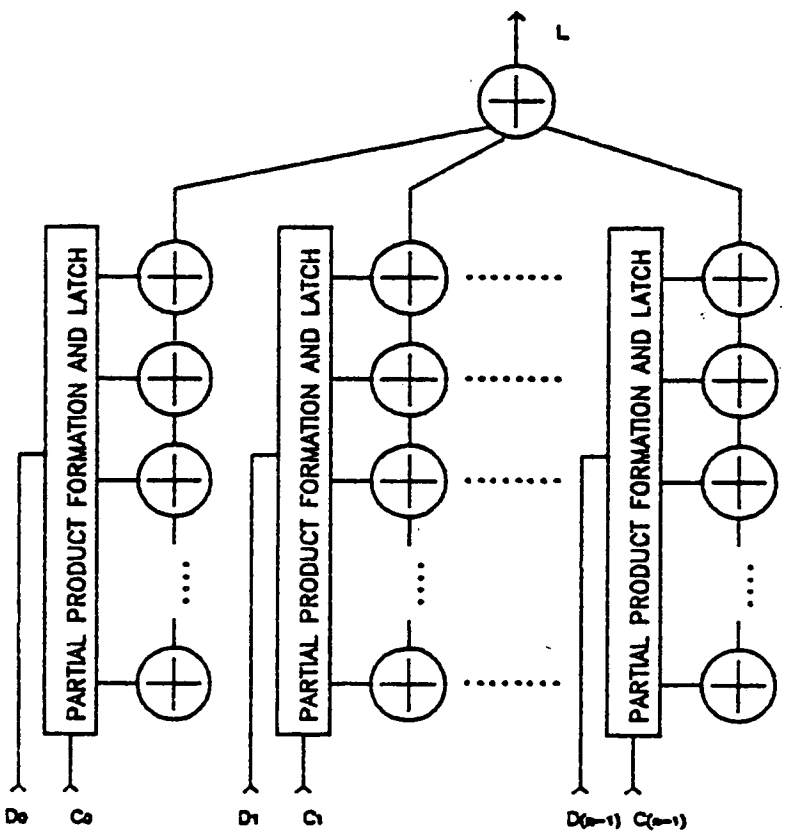
3.3. Distributed Arithmetic Methodology

3.3.1. Introduction

Almost all common arithmetic functions can be built up sequentially from additions. Multiplication, for example, is simply the sequential addition of a number of partial products which are closely related to the data and coefficient words. Addition is a commutative mathematical process, which means that the order in which the additions are performed does not in any way affect the final result. This fact means that the arithmetic of many signal processing structures can be re-configured in a number of ways to form alternative distributed arithmetic structures. In particular, where two or more multiplier outputs are combined in an adder (Figure (3.3.1.1)), it is possible to view each of the multipliers as a collection of adders connected to this output adder (Figure (3.3.1.2)). As the whole process can be seen in terms of additions only where it is possible to bring forward the final combining addition to form new unique merged multiplier partial products which can be selected and accumulated (in a similar fashion to the accumulation of partial products in the multipliers originally) to form the same result (Figure (3.3.1.3)). This reformulation allows computation of the function by using a data controlled table look-up and accumulate operation which offers a highly regular design approach -



Figure(3.3.1.1.) Linear Equation Implemented using Conventional Arithmetic.



Figure(3.3.1.2.) Linear Equation showing Additions Present in Shift and Add Multipliers.

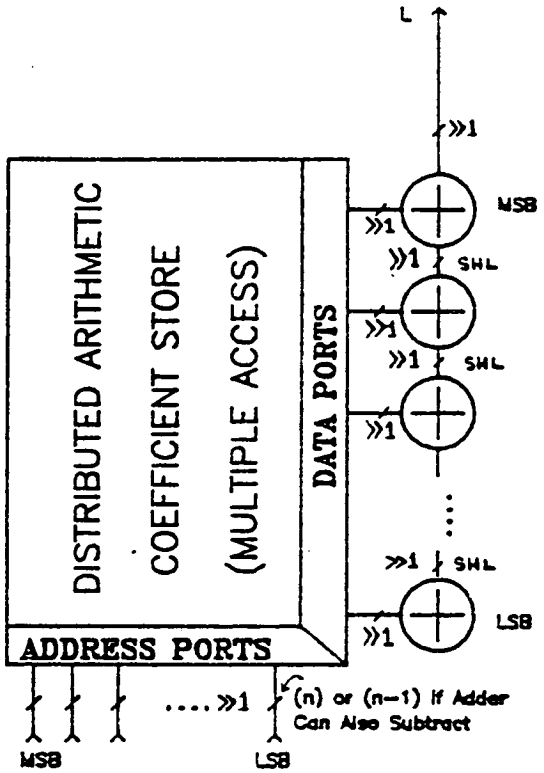
well suited to VLSI design techniques.

Much of the original work in distributed arithmetic was centred on the design of digital filter structures where it was seen as a way of replacing relatively expensive multiply and accumulate structures with cheaper memory and accumulate structures such as shown in Figure (3.3.1.4) [12,13] however it was later recognised that these techniques could also be applied to other computations [14,15,16] including the DFT and FFT.

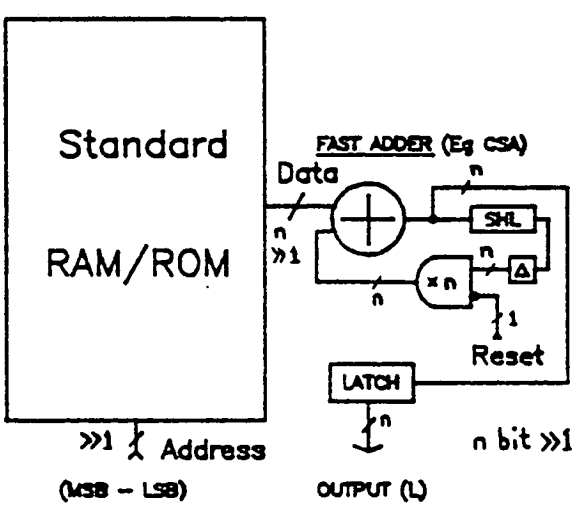
Linear equations of the general form shown in equation 3.3.1 are fundamental to the computation of the DFT and FFT, as the two term (n=2) linear equation describes the complex multiply, which is one of the most popular ways of performing vector rotations. D represents data and A represents the coefficient.

$$L = A_0 D_0 + A_1 D_1 + \ldots + A_{n-1} D_{n-1} \qquad (3.3.1)$$

Thus when vector rotation is implemented using real multipliers, the real or imaginary output of the n point DFT becomes a linear equation with 2n terms - discussed in section 4. It can be seen from equation 3.3.1 that a linear equation with n terms requires n multiplications and (n-1) additions. If it is desired to implement such a function using distributed arithmetic then 2^n arithmetic combinations of multiplier partial products must be stored since these are the number of combinations of multiplier partial products that are possible. This assumes that none of the distributed arithmetic merged partial products are allowed to be stored



Figure(3.3.1.3.) Distributed Arithmetic Linear Equation Computation - Totally Parallel using Multiple Access Memory.



Figure(3.3.1.4.) Distributed Arithmetic Linear Equation Computation-Serial Accumulator and Standard Memory.

more efficiently which has not been proven. They can be computed in situe to reduce the memory overhead, for example, as described in the next section.

Distributed arithmetic is quite significant therefore in allowing the replacement of multiplier random logic array structures with a simple and regular memory and accumulator structure. This can often mean lower power consumptions, and faster speeds, in addition to a considerably increased regularity in the chip design itself. In VLSI designs, regularity is a key requirement as it allows changes to be made much more easily, as well as making design for yield, such as self repair techniques, more efficient.

3.3.2. Consequences of using Distributed Arithmetic

It has been explained that distributed arithmetic can offer savings in speed and power in actual chip implementations, as well as offering what could be regarded as a complete methodology in allowing a fairly reliable means of producing regular VLSI designs.

One of the disadvantages of distributed arithmetic, however, is that the memory requirement goes up as 2^m where m is the number of multipliers that would have been used in a conventional system. Thus, whilst small linear equations can be implemented quite easily, larger linear equations can call for excessive memory requirements.

A possible solution to this problem may lie in information theory [17,18] which notes that in a large array of data, there often exist constraints which act to reduce the entropy of the array. In such cases, information is not being stored most

efficiently, with much data in the array being closely related.

Identification of these constraints in an array which contains distributed arithmetic coefficients could result in a potential reduction in the storage requirements. It is likely that future research in the area of information theory may yield ideas which would make distributed arithmetic techniques practical for large mathematical functions.

An immediate approach to solving the explosion in the number of distributed arithmetic coefficients that need to be stored as the number of multiplies increases, is to actively compute the distributed arithmetic coefficients in situe. This is not algorithmically more efficient than conventional arithmetic, but in allowing a restructuring of the computation, the possibility of producing more regular layouts suitable for VLSI exists. This is particularly attractive for applying yield enhancement techniques and therefore points to the possibility of wafer scale integration. In effect, the order of bit-level additions has been altered, but the number of additions is kept approximately constant (very slight variations may be required due to dynamic range considerations - ie: word growth).

In chapter 4, the approach of computing the distributed arithmetic coefficients in situe will be compared with that of storing them. This is done for a four term linear equation example which is computed entirely in parallel. The possibility of performing yield enhancement with this type of structure is demonstrated.

For smaller arithmetic functions, the distributed arithmetic coefficients can be stored and do not therefore need to be computed

in situe. Distributed arithmetic then appears to be very attractive. One such arithmetic function mentioned earlier is the complex multiply which is highly relevant to computation of the DFT and FFT as a means of performing vector rotations. The use of distributed arithmetic to compute this function will be discussed in section 5 of this chapter.

3.4. Distributed Arithmetic for Computing Small DFT's

3.4.1. General

It was mentioned, in Section 1, that the real or imaginary output of an n-point DFT, is a linear equation with 2n terms when the complex multiply is evaluated using real multipliers. Distributed arithmetic is therefore applicable, in principle, to the computation of any length of DFT. In practice, however, a distributed arithmetic expansion of the W coefficient sequence would be required for each row of the W coefficient matrix. This would involve an excessive memory storage requirement, and so is undesirable.

One area where it would appear practical to use distributed arithmetic techniques for DFT computation is for prime length DFT's, as the DFT equation can be expanded and re-ordered to be expressed as a circular convolution. This particular case will now be considered.

3.4.2 Distributed Arithmetic and the Prime DFT

Since every (non-DC) row of the W coefficient matrix in the Prime, n-point DFT contains all nth roots of unity, it is possible to expand and re-order the DFT expression so that a single integer frequency term can be used together with a shuffled time domain sequence to produce any one of the non-zero frequency domain results. Since only a single integer frequency sequence is required to compute the DFT in the convolution form, then only a single distributed arithmetic expansion need be stored in memory. This expansion involves combinations of arithmetically merged W coefficient terms that appear in the sequence. This data is then accessed as a function of the time domain bits at various levels of significance. Siu and Chen describe a 6800 8-bit microprocessor based system [19] operating with a 500 nS cycle time, which used distributed arithmetic to compute a 61 point complex DFT in only 3.1 milli-seconds. Distributed arithmetic DFT computation is therefore highly relevant to micro-computer based systems which offer large memory availability. This approach does however result in a high dependence on memory (ROM) which also fixes the size of DFT that can be computed. No specific implementations were considered because of this limitation, however, a possible VLSI architecture is suggested in Figure (3.4.1), which is suited to the computation of DFT's of fixed size.

3.5. Distributed Arithmetic and the FFT

As previously mentioned, the most computationally intensive arithmetic requirement of the FFT, is the complex multiply which is conveniently used to achieve vector rotation in a Cartesian

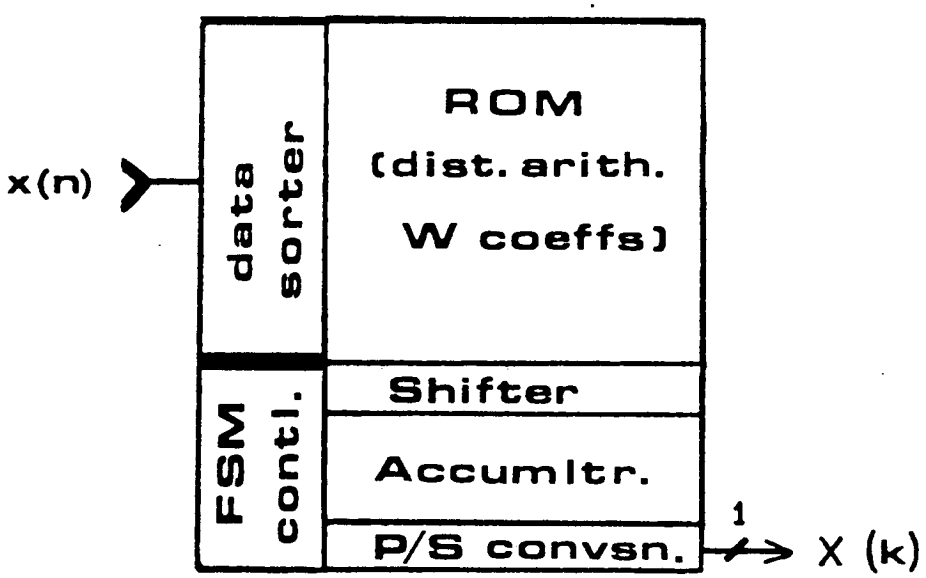


Figure (3.4.1.) Distributed arithmetic allows efficient computation of fixed length Prime DFT's in structures such as shown above.

coordinate based system.

In Chapter 2, section 1 on the DFT, is was described how a well known trigonometric relation could be used to describe the rotation of one vector by another. This meant that a data vector (B) could be rotated by a coefficient vector (W) to produce a resultant vector (Z) as shown in equations (3.5.1) and (3.5.2), whose angle becomes the sum of the angles of W and B (B, W, Z, complex).

$$Re(Z) = Re(B).Re(W) - Im(B).Im(W) \qquad (3.5.1)$$

$$Im(Z) = Re(B).Im(W) + Re(W).Im(B) \qquad (3.5.2)$$

This equation can be realised using conventional arithmetic, but White [16] has shown that distributed arithmetic allows the complex multiply to be realised particularly efficiently in hardware. In his paper he described a TTL based two accumulator bit-serial radix-2 butterfly. This structure is the hardware equivalent of using two multipliers, instead of four multipliers as would normally be needed to implement the two equations. This paper by White was considered to be particularly relevant to this work, as it was recognised that the complex multiply algorithm (using distributed arithmetic) had a considerable potential for parallel data array architectures of a type which is highly suited to VLSI, since it allowed computation of the real or imaginary complex multiply output data using only a single accumulator. This can be realised in a pipelined form as a single array of full adders. The distributed arithmetic complex multiply algorithm described by White appears to be largely correct, however, one apparent error relating to the initialisation of the accumulator was noted which originated from

early on in his derivation. Although this error is not large in numerical terms, it would introduce some degree of unnecessary arithmetic noise, so a separate derivation of the algorithm will be given here in detail, with this correction included. The reasons for each step in the derivation will also be given.

3.5.1. The Complex Multiply using Distributed Arithmetic

The complex multiplication can be computed using four real multiplications and an add and subtract as shown in equations (3.5.1) and (3.5.2), where the data vector (B) is rotated by the the coefficient vector (W) to form the output vector (Z). It is necessary to define the representation of the real and imaginary binary strings used to describe these vectors, before deriving any specific algorithm, as the exact operation of the algorithm is dependent on the way in which data is to be interpreted.

There are several methods of representing numbers using an n-bit string of binary data, however, in digital signal processing, one of the most useful interpretation of such a string is fractional fixed point 2's complement notation, as this allows both positive and negative numbers to be represented and allows direct interface with most types of analogue to digital conversion systems. It is believed, however, that, a distributed arithmetic algorithm is likely to exist for most commonly used numerical representations using binary data. Using the above notation, then, the real and imaginary words that represent the coefficient vector can be represented as the summation shown in equations (3.5.3) and (3.5.4).

$$Re(W) = -W_{RO} + \sum_{n=1}^{N-1} W_{Rn}\, 2^{-n} \qquad (3.5.3)$$

$$Im(W) = -W_{IO} + \sum_{n=1}^{N-1} W_{In}\, 2^{-n} \qquad (3.5.4)$$

This allows equations (3.3.1) and (3.3.2) for Real(Z) and Imag(Z), to be re-written, as can be found in equation (3.5.5) for Re(Z). A similar expression for Im(Z) can be written to allow a distributed arithmetic reformulation of this to be constructed in parallel with that for Re(Z). In this derivation, only the expression for Re(Z) will be continued since the derivation of Im(Z) is based on exactly the same principles used to derive Re(Z).

$$Re(Z) = [-W_{RO} + \sum_{n=1}^{N-1} W_{Rn}\, 2^{-n}\ ]\ Re(B) - [-W_{IO} + \sum_{n=1}^{N-1} W_{In}\, 2^{-n}\ ]\ Im(B) \qquad (3.5.5)$$

Since addition is commutative, the order that the additions are performed in can be altered so that the separate summations shown above can now be combined into a single summation by decoding all the combinations of the Real and Imaginary W bits. This process effectively involves the formation of new unique merged partial products which can be selected by the Real and Imaginary W bits, as shown in equation (3.5.6).

$$Re(Z) = \left[W'_{RO}\ W'_{IO}\ (0) \right] + \left[\sum_{n=1}^{N-1} [W'_{Rn}\ W'_{In}\ (0) \right]$$

cont..

cont..

$$\left[\begin{array}{l} + W'_{RO}\, W_{IO}\, (Im(B)) \\ + W_{RO}\, W'_{IO}\, (-Re(B)) \\ + W_{RO}\, W_{IO}\, (-Re(B)+Im(B)) \end{array}\right] + \left[\begin{array}{l} + W'_{Rn}\, W_{In}\, (-Im(B)) \\ + W_{Rn}\, W'_{In}\, (Re(B)) \\ + W_{Rn}\, W_{In}\, (Re(B)-Im(B))]\ 2^{-n} \end{array}\right] \qquad (3.5.6)$$

As equation (3.5.6) involves only a single summation, the equation for Re(Z) can thus be implemented using only a single accumulator. This equation involves the W controlled selection of one of four merged partial products, however, a more optimum solution can be achieved by describing the merged partial products in terms of K and K^* as defined in equation (3.5.7).

$$K = (Re(B)+Im(B))/2 \quad \text{and} \quad K^* = (Re(B)-Im(B))/2 \qquad (3.5.7)$$

This allows equation (3.5.6) to be re-written as shown in equation (3.5.8).

$$Re(Z) = \left[\begin{array}{l} W'_{RO}\, W'_{IO}\, (-K^* + K^*) \\ + W'_{RO}\, W_{IO}\, (-K^* + K) \\ + W_{RO}\, W'_{RO}\, (-K^* - K) \\ + W_{RO}\, W_{IO}\, (-K^* - K^*) \end{array}\right] + \left[\begin{array}{l} \sum_{n=1}^{N-1} [W'_{Rn}\, W'_{In}\, (K^* - K^*) \\ \quad + W'_{Rn}\, W_{In}\, (K^* - K) \\ \quad + W_{Rn}\, W'_{In}\, (K^* + K) \\ \quad + W_{RO}\, W_{Rn}\, (K^* + K^*)]\ 2^{-n} \end{array}\right] \qquad (3.5.8)$$

Part of this equation can now be simplified by noting the W independence of one of the Ksup* terms to form the final equation, (3.5.9). This fully describes how Re(Z) is formed in a single accumulator, as a function of W and B.

$$Re(Z) = -K^{*}\ 2^{-(N-1)}$$

$$\left[\begin{array}{l} + W'_{RO}\ W'_{IO}\ (+K^{*}) \\ + W'_{RO}\ W_{IO}\ (+K) \\ + W_{RO}\ W'_{IO}\ (-K) \\ + W_{RO}\ W_{IO}\ (-K^{*}) \end{array}\right] + \left[\begin{array}{l} \sum_{n=1}^{N-1} [+ W'_{Rn}\ W'_{In}\ (-K^{*}) \\ \qquad + W'_{Rn}\ W_{In}\ (-K) \\ \qquad + W_{Rn}\ W'_{In}\ (+K) \\ \qquad + W_{Rn}\ W_{In}\ (+K^{*})]\ 2^{-n} \end{array}\right] \qquad (3.5.9)$$

The control signals can conveniently be separated into a data-select control signal and an add/subtract control signal. The data-select control can be derived from an exclusive-NOR type relation between the real and imaginary W bits, whilst the add/subtract control can be derived from the real W bit itself. This is the form in which the control requirements of the algorithm would be best implemented on a chip and was used in the implementations described in the following chapters. This alternative expression of (3.5.9) for Re(Z) is shown in (3.5.10).

$$Re(Z) = -K^{*}\ 2^{-(N-1)}$$

$$\left[\begin{array}{l} + W'_{RO}\ (W_{RO} \overline{\oplus} W_{IO})(+K^*) \\ + W'_{RO}\ (W_{RO} \oplus W_{IO})(+K) \\ + W_{RO}\ (W_{RO} \oplus W_{IO})(-K) \\ + W_{RO}\ (W_{RO} \overline{\oplus} W_{IO})(-K^*) \end{array}\right] + \left[\begin{array}{l} \sum_{n=1}^{N-1} [\ + W'_{Rn}\ (W_{Rn} \overline{\oplus} W_{In})(-K^*) \\ \quad + W'_{Rn}(W_{Rn} \oplus W_{In})(-K) \\ \quad + W_{Rn}\ (W_{Rn} \oplus W_{In})(+K) \\ \quad + W_{Rn}\ (W_{Rn} \overline{\oplus} W_{In})(+K^*)]\ 2^{-n} \end{array}\right]$$

(3.5.10)

The expression for Im(Z) can be generated using the same reasoning and procedure that was used to generate Re(Z), giving equation (3.5.11) which completes the description of this algorithm for computing the complex multiply using distributed arithmetic.

$$Im(Z) = -K\ 2^{-(N-1)}$$

$$\left[\begin{array}{l} + W'_{IO}\ (W_{IO} \oplus W_{RO})(+K^*) \\ + W_{IO}\ (W_{IO} \overline{\oplus} W_{RO})(-K) \\ + W'_{IO}\ (W_{IO} \overline{\oplus} W_{RO})(+K) \\ + W_{IO}\ (W_{IO} \oplus W_{RO})(-K^*) \end{array}\right] + \left[\begin{array}{l} \sum_{n=1}^{N-1} [\ + W'_{In}\ (W_{In} \oplus W_{Rn})(-K^*) \\ \quad + W_{In}(W_{In} \overline{\oplus} W_{Rn})(+K) \\ \quad + W'_{In}\ (W_{In} \overline{\oplus} W_{Rn})(-K) \\ \quad + W_{In}\ (W_{In} \oplus W_{Rn})(+K^*)]\ 2^{-n} \end{array}\right]$$

(3.5.11)

The potential of this algorithm for parallel data implementations of the FFT butterfly and pipelined datapaths geared to high speed processing of complex numbers is considered in the next chapter.

3.6. Summary

This chapter has covered a number of algorithms which allow high bandwidth vector arithmetic. Some of these algorithms exhibit somewhat irregular mappings on to silicon such as CORDICS, whilst other approaches such as distributed arithmetic offer highly regular silicon structures by reformulating conventional shift and add multiplier based arithmetic. It is felt that distributed arithmetic has a considerable potential for parallel data VLSI arithmetic processor implementations and the next chapter investigates a number of possible datapath architectures based on the distributed arithmetic complex multiply algorithm, just described. This would allow very high bandwidth computation of the FFT butterfly with the potential bottleneck resulting from the vector rotation requirement effectively removed.

References

1. J. E. Volder, "The CORDIC Trigonometric Computing Technique," IRE Trans. Electron. Comput., Vol. EC-8, pp. 330 - 334 (August 1959).

2. A. M. Despain, "Fourier Transform Computers using CORDIC Iterations," IEEE Trans. on Computers, Vol. c-23, No.2, pp. 993-1002 (July-December 1974).

3. J. S. Walther, "A Unified Algorithm for Elementary Functions," Spring Joint Computer Conf. Proceedings, pp. 379-385 (1971).

4. G. L. Haviland and A. A. Tuszynski, "A CORDIC Arithmetic Processor Chip," IEEE Trans. on Computers, Vol. c-29, No. 2, pp. 68-79 (February 1980).

5. J. M. Delmose, "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces," ICASSP 83 Proceedings, pp. 927-930 (1983).

6. H. M. Ahmed, P. H. Ang, and M Morf, "A VLSI Speech Analysis Chip Set Utilising Co-ordinate Rotation Arithmetic," IEEE Computers, pp. 737-741 (1981).

7. D. T. L. Lee and M. Morf, "Generalised Cordic for Digital Signal Processing," IEEE ICASSP'82, pp. 1748-1751 (1982).

8. H. M. Ahmed, J. M. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," IEEE Computing, pp. 65-82 (January 1982).

9. P. C. Maxwell, "An nMOS Transcendental Function Generator," VLSI DESIGN, pp. 70-72 (November 1983).

10. R. C. Singleton, "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," IEEE Trans. Audio Electroacoustics, Vol. AU-15, pp. 45 - 55 (June 1967).

11. O. Buneman, "Inversion of the Helmholtz (or Laplace - Poisson) Operator for Slab Geometry," Institute for Plasma Research Stanford University Stanford, California, SUIPR Rep. 467, p. 5 (April 1972).

12. A. Peled and B. Liu, "A New Hardware Realisation Technique of Digital Filters," Presented at the IEEE Arden House Workshop on Digital Signal Processing, Harriman, (January 1974).

13. C. S. Burrus, "Digital Filter Structures Described by Distributed Arithmetic," IEEE Trans. on Circuits and Systems, Vol. CAS-24, No. 12, pp. 674-680 (December 1977).

14. S. A. White, "On Mechanisation of Vector Multiplication," Proc. IEEE, Vol. 63, pp. 730-731 (April 1975).

15. S. Zohar, "Fast Hardware Fourier Transforms through Counting," IEEE Trans. on Computers, Vol. c-23, p. 989 (September 1975).

16. S. A. White, "A Simple FFT Butterfly Arithmetic Unit," IEEE Trans. on Circuits and Systems, Vol. CAS-28 No.4, pp. 352-355 (April 1981).

17. L. Brillouin, Science and Information Theory, Academic Press, New York (1956).

18. R. G. Gallager, Information Theory and Reliable Communication, Wiley, New York (1968).

19. W. C. Siu and B. S. Chen, "New Realisation Technique of high-speed discrete Fourier Transform Described by Distributed Arithmetic," IEE Proceedings, Vol. 130, Part E No.6, pp. 177-182 (November 1983).

Chapter 4 - VLSI Datapath Architectures for Complex Number Arithmetic

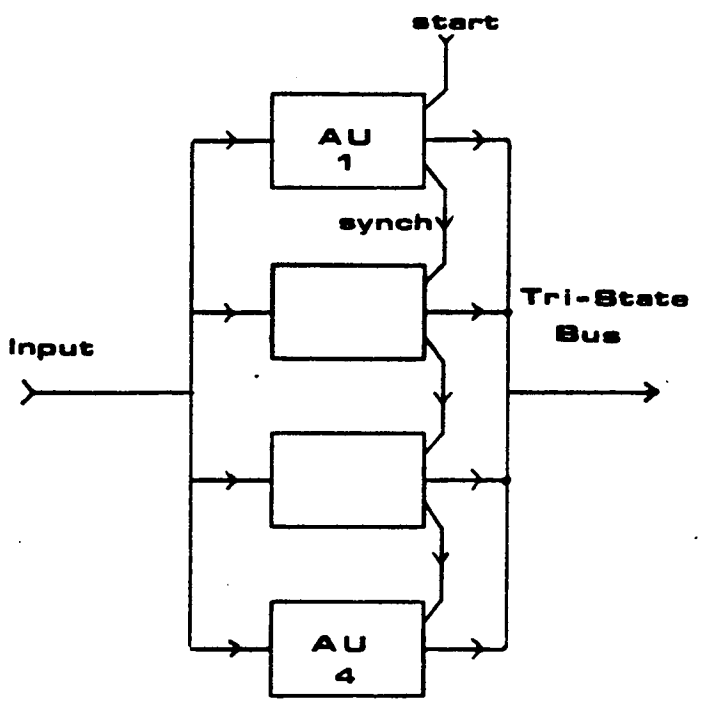
4.1. Introduction

This Chapter will first look at VLSI datapath architectures in general, and then look at datapaths which are specifically optimised for operation on complex number arithmetic. In particular, distributed arithmetic approaches to computing the complex multiply function discussed in the previous chapter, will be compared to conventional approaches using real multipliers. The complex multiply is a central requirement of the DFT and FFT as a means of performing vector rotation but is also the most computationally intensive function required in computing the basic DFT and FFT.

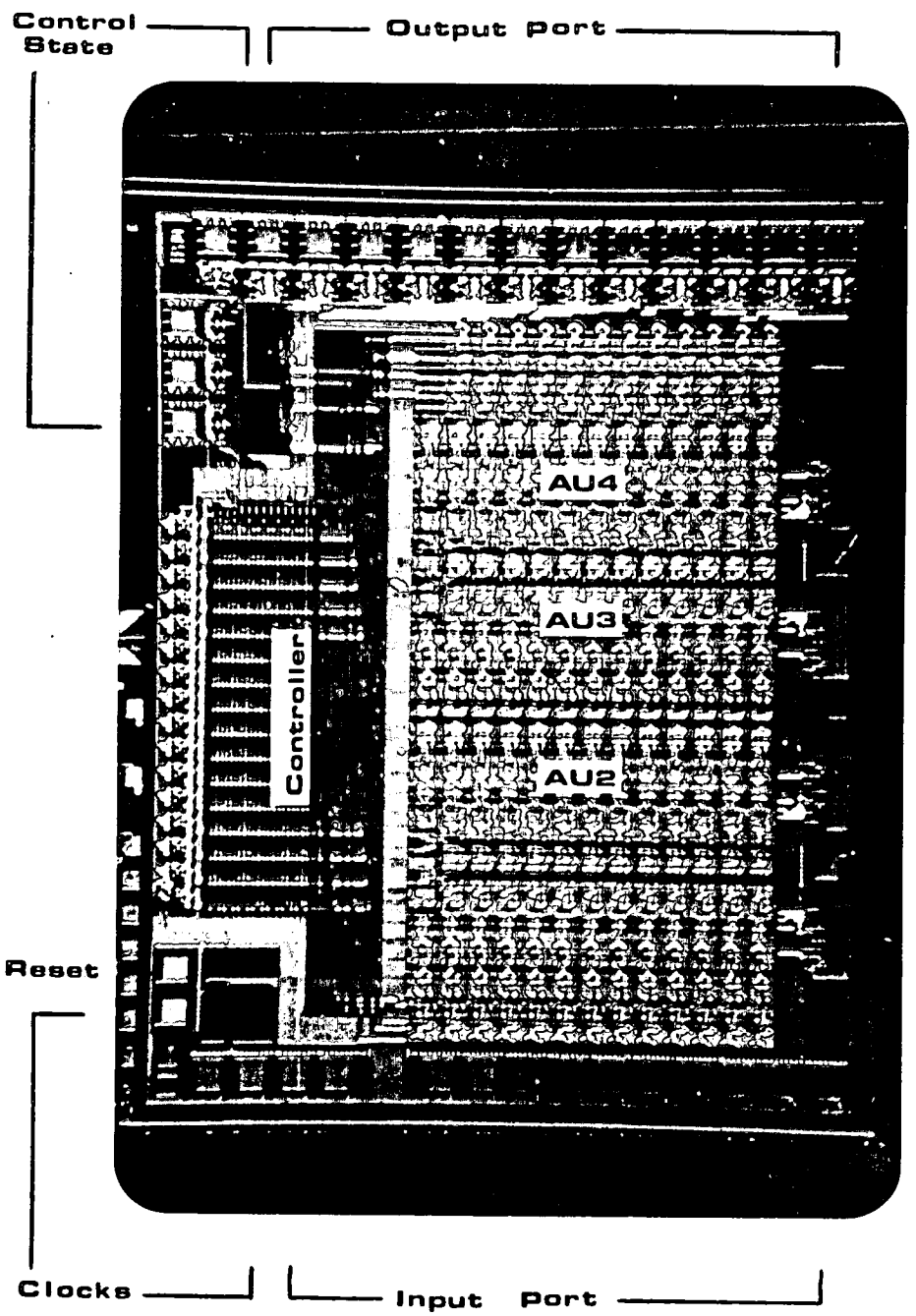
Real time computation of the FFT demands very high data rates from the arithmetic processor and the new architectures considered in this chapter offer hardware computation of the complex multiply with high efficiency. Conventional approaches will be discussed first, however, with a look at arithmetic datapath design.

4.2. Conventional Arithmetic Datapaths

The datapaths used in the early microprocessors (eg 6502, 6800, 8080, Z80) typically relied on a single arithmetic unit to perform the basic add or subtract functions as well as logical operations. Thus, multiplications using such a datapath could be described as slow and complex multiplication as very slow. This bus orientated datapath architecture can be made faster by providing more than one arithmetic unit thus allowing some of the operations to take place



Figure(4.2.1.) Several Low Bandwidth Arithmetic Units can be Stacked on a single bus provided that they are synchronised.



Figure(4.2.2.) Photograph of a Controlled Datapath which allows Arithmetic Concurrency.

concurrently. For example, four arithmetic units would allow the complex multiply, additions and subtractions required by the Radix-2 FFT butterfly to be performed in around 20 clock cycles at 16 bits. This type of datapath would need a small Finite State Machine (FSM) controller to translate specific instructions to direct control of the datapath. Mactaggart [1] describes one such device with a FSM control unit which in addition to controlling the datapath arithmetic, defined inputs and outputs of the datapath and provided tri-state enable signals for the output port. The control unit sequencer could be synchronised with other identical devices (up to four) to enhance arithmetic throughput (by up to four times) by stacking the devices as shown in Figure (4.2.1). Figure (4.2.2) shows a photograph of the fabricated nMOS device.

More recent processors such as the TMS320 signal processor, described in chapter 2, make use of a hardware multiplier, thus allowing the real multiply function to be computed in a single clock cycle. Although not available as an instruction on the TMS320, double precision multiplication can still be computed efficiently (as might be required for simulation work) if a barrel shifter is included in the datapath. Four single precision multiplies, two shifts and three additions are then required to compute a double precision (dp) multiplication as proven in (4.2.1) to (4.2.5).

$$\text{Define} \quad A_{dp} = (2^N w) + x \tag{4.2.1}$$

$$\text{and} \quad B_{dp} = (2^N y) + z \tag{4.2.2}$$

where w,x,y and z are single precision numbers (N bits each)

$$\text{Also } C_{dp} = A_{dp} \cdot B_{dp} \tag{4.2.3}$$

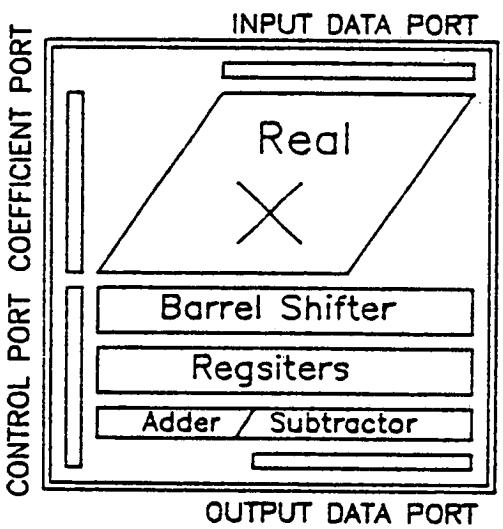
Thus

$$C_{dp} = ((2^N w) + x)((2^N y) + z) \tag{4.2.4}$$

So that

$$C_{dp} = (2^{2N} wy) + (2^N (wz+xy)) + xz \tag{4.2.5}$$

The single precision hardware multiplier can thus be used to compute (wy), (wz), (xy), and (xz), with the barrel shifter performing the two shifts that are required. It is important to note however that this datapath is only efficient if the accumulator used is a double precision device. This is because of the gain introduced by the multiplier. So in a 16 bit datapath, for example, a 16 by 16 bit multiplier would be used in conjunction with a 32 bit adder/subtractor. In order not to make single precision (16 bit) addition inefficient, the adder/subtractor might also be configured as two 16 bit devices which could be allowed to operate concurrently. Thus the datapath would cater for single and double precision arithmetic with high efficiency. A typical floorplan for such a datapath is outlined in Figure (4.2.3). This datapath could be controlled by a relatively simple sequencer which would allow a variety of functions to be computed with single or double precision



(With Fast Single and

Double Precision Multiply)

Figure(4.2.3.) A Datapath Geared to High Speed Processing Should Ideally Incorporate a Hardware Multiplier.

arithmetic. The emphasis with this datapath is on versatility, and it would therefore use a ripple-through (unity latency) multiplier to allow efficient recursive arithmetic operations to take place if required.

High bandwidth datapaths would not normally make use of more than one parallel real multiplier, although clearly the complex multiply instruction could potentially use up to four. Thus a single multiplier would be multiplexed to perform the same function.

In extreme cases, a conventional arithmetic datapath might use two real multipliers to allow the complex multiply to be executed at very high bandwidths. Two multipliers, however, are not likely to map particularly efficiently onto silicon due to irregular multiplier structure and extra bus interconnections.

The distributed arithmetic approach to computing the complex multiply, described in the previous chapter is likely to map onto silicon much more efficiently because it is based on a single accumulation process. The next section looks therefore at distributed arithmetic VLSI datapath architectures with hardware orientated complex multiplication.

4.3. Distributed Arithmetic Datapaths

The main significance of the distributed arithmetic complex multiply algorithm described in Chapter 3 is that it allows the real or imaginary part of the complex product to be computed in a single accumulator, as described in equations (3.5.10) and (3.5.11). The use of a single accumulator (instead of the usual two needed to

compute each half of the complex multiply - one for each real multiplier) allows the possibility of constructing a highly regular array for parallel data operation which has the throughput equivalence of two parallel multipliers although consuming only slightly more area than a single multiplier. Also, a single distributed arithmetic array does not require awkward bus interconnections as would be required to route the outputs of several parallel data sources in a conventional arithmetic processor, for example from two parallel multipliers to a parallel adder or subtractor. For this reason as well as for yield considerations, a conventional parallel arithmetic approach to the complex multiply would be to use a single multiplexed multiplier. The distributed arithmetic approach, however, allows the possibility of realising on a single chip of modest size, a structure which has the functional equivalence of two parallel multipliers when configured to compute (3.5.10) and (3.5.11). This would probably not even be contemplated using conventional parallel arithmetic except at limited wordlengths.

In considering possible structures for implementing the complex multiply using distributed arithmetic, it is necessary to consider whether the technique of pipelining would be appropriate as this can often allow further bandwidth enhancements to be achieved.

4.3.1. Pipelining - Bandwidth Enhancement in non-Recursive Processes

An important feature of the DFT and FFT is that arithmetic operations may be carried out continuously, as there is no high bandwidth recursion required between arithmetic processor outputs

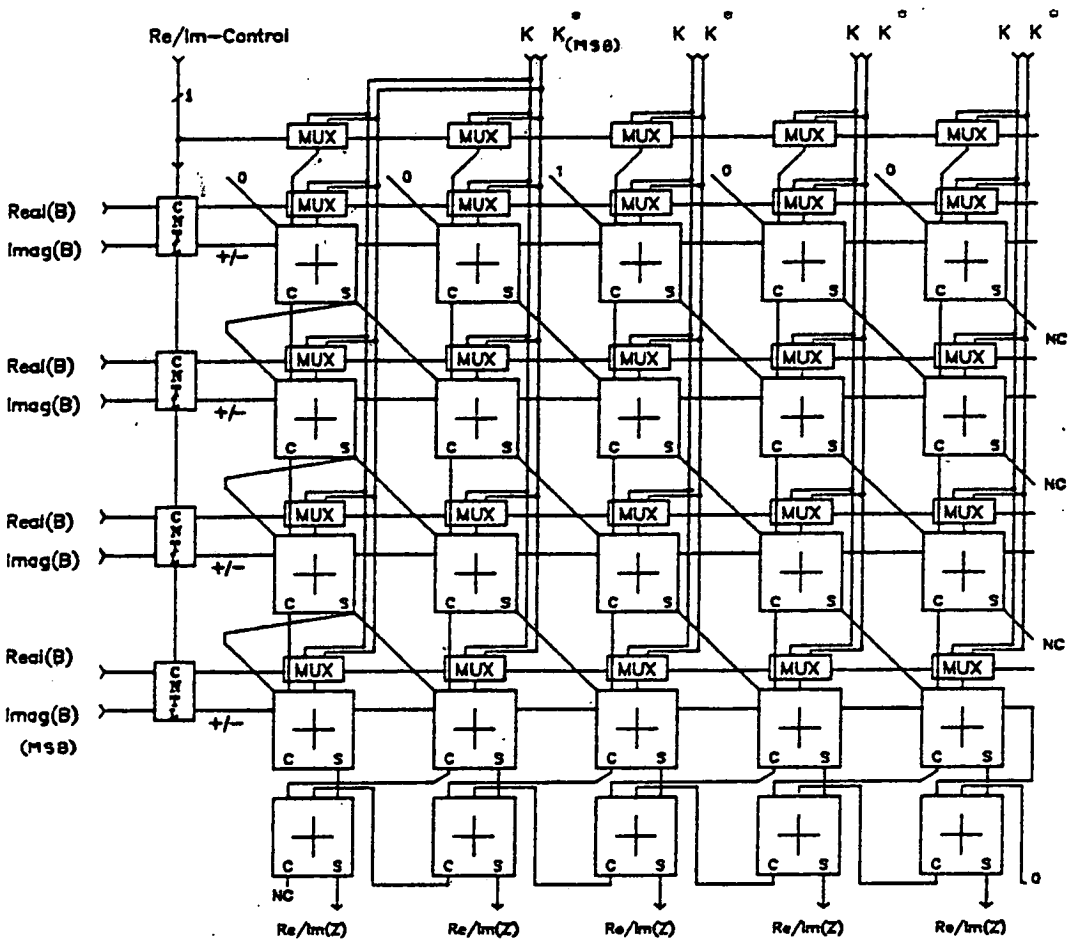
and inputs. This allows the possibility of employing pipelining techniques within the discrete arithmetic stages of a structure to achieve high clocking rates. As a substantial enhancement of bandwidth can be achieved through this technique, it was decided to specifically consider structures with some degree of pipelining. It is important to note, however, that very high levels of pipelining can cause problems in clock distribution, so that an optimum level of pipelining must be sought for a given technology. Some of the very fast technologies such as Emitter Coupled Logic and the newer Gallium Arsenide technologies are so fast that it is difficult to consider pipelining anything much smaller than a large parallel multiplier for example, which would operate on a ripple through basis, at times in the order of 10 nS. In general, the slower the technology, the higher the degree of pipelining that is possible without running into problems of race as a result of poor clock distribution.

Equations (3.5.10) and (3.5.11), which describe the distributed arithmetic complex multiply algorithm, indicate that both real and imaginary results of the complex product are formed by the coefficient controlled selection of + or - (K or K^*) (K's defined in equation (3.5.7)). The equations which describe the algorithm further indicate that this selected word must then be added into an accumulator at some level of significance, which can be achieved by shifting. The main circuit element required to implement the algorithm is therefore an accumulator so considering only parallel data array implementations (for which distributed arithmetic techniques are likely to be most appropriate), the resulting structure

resembles a standard shift-and-add parallel multiplier, and can therefore similarly be constructed around a skeleton comprising an array of full adder cells. In such an array, it is possible to route the carries, sums and the distributed arithmetic coefficients in a variety of ways to achieve different structures with various levels of pipelining.

Simple non-pipelined structures will also be considered however, because although pipelined structures offer higher bandwidths than do non-pipelines structures, there are many applications where the arithmetic may be required to operate in a recursive mode, such as to generate a rotating vector for example, or in a Prime DFT processor as discussed in chapter 2.

A non-pipelined (ripple-through) parallel distributed arithmetic array requires multiple access of the distributed arithmetic coefficients. For small functions such as the two term linear equation for each half of the complex multiply function, it is feasable to run the coefficients through the chip from top to bottom. The distributed arithmetic coefficients can be fed vertically down through the chip producing the architecture shown in Figure (4.3.1.1) which implements equations (3.5.10) and (3.5.11). This however, is not the most efficient approach in a ripple through structure where it is better to implement equation (3.5.9) for Re(Z) and its counterpart (for Im(Z) - not shown), which is the W controlled selection of either $+K,-K,+K^*,-K^*$. This avoids computing the complements of K and K^* at each cell in the array (Boolean inversion on each bit) at the expense of feeding the complements through the chip which is a small communications overhead in this case. Figure

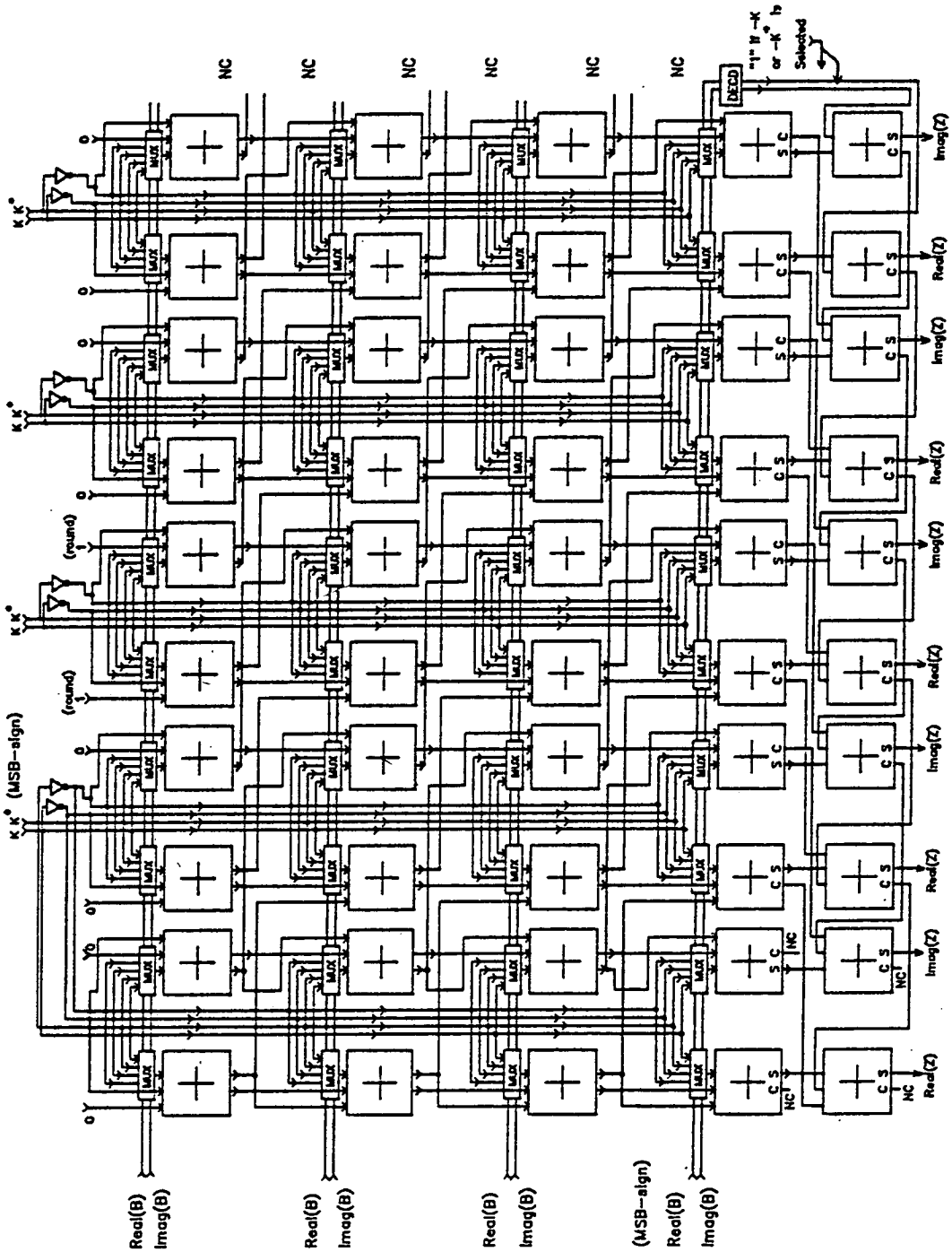


Figure(4.3.1.1.) A Real/Imaginary Programmable Ripple-Through Complex-Multiply Architecture.

(4.3.1.2) shows this slightly better alternative to (4.3.1.1) which has been expanded to a fully parallel implementation by employing two full adders and multiplexers per basic cell to allow the complex multiply function to be computed completely in parallel. This structure is quite area efficient because the distributed arithmetic coefficients required to compute the real output are also used to compute the imaginary output. Thus the communications overhead of four bus lines per cell is minimal, with this approach. Performance could be slightly improved further by using a common decoder for the multiplexers in each row of cells since the control to each decoder in a given row is always the same. In Figure (4.3.1.2), this would involve four horizontal control lines instead of two, as shown. Were K and K^* required to be shifted down through the chip as might be required in a pipelined structure then it would be better to implement (3.5.10) and (3.5.11) instead.

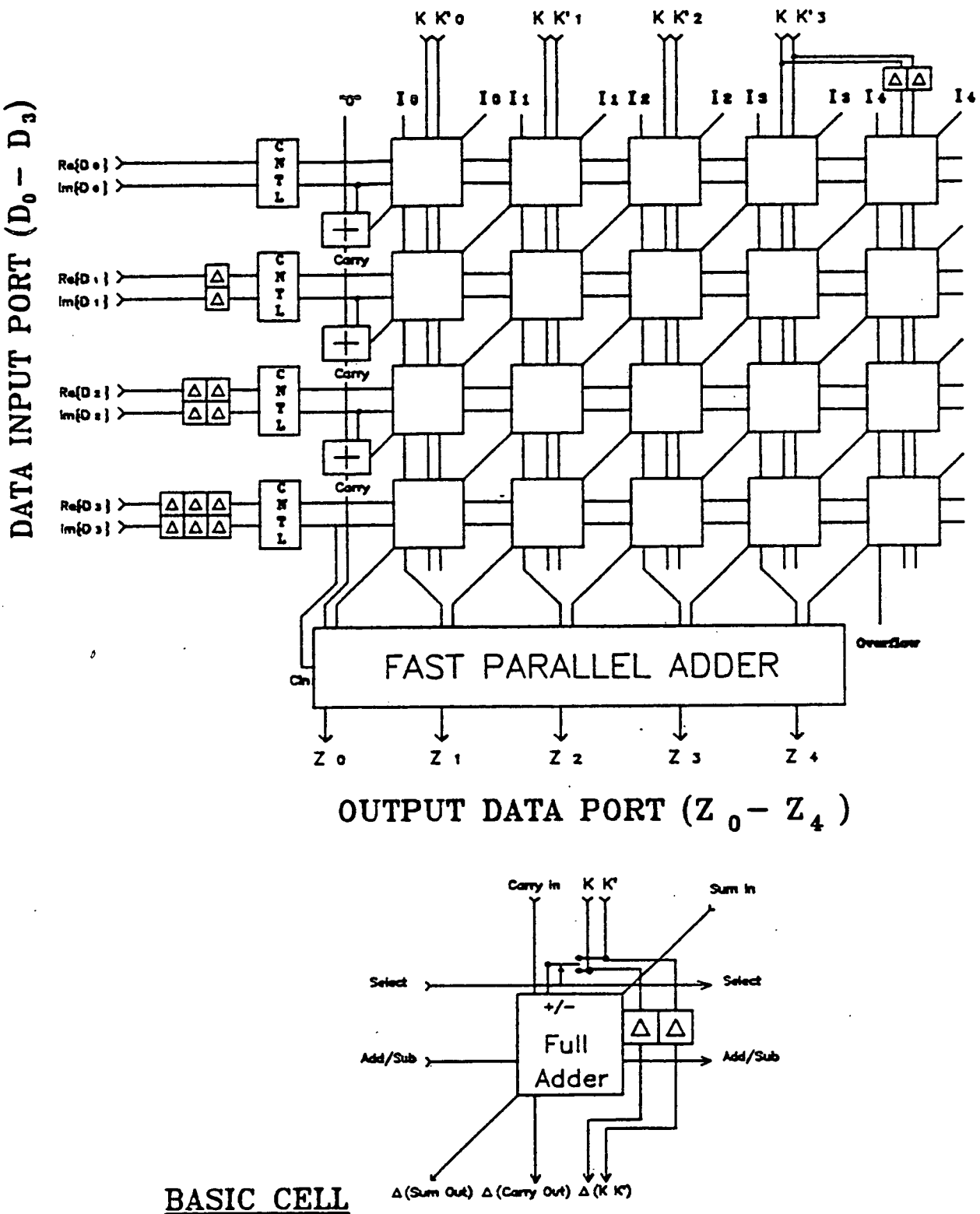
It is proposed to start by looking at lower levels of pipelining and then move towards higher levels which allow clocking rates that are essentially word length independent.

In all the structures considered here it will be useful to view the array of full adder cells which will form the skeleton of the distributed arithmetic algorithm implementation, as several rows of parallel adders. It would not be surprising to consider that the carries would therefore be fed horizontally within each row which represents a parallel adder, and that the sums would be fed down to the next row of full adder cells representing another parallel adder, with a possible shift in significance if required. Unfortunately if the horizontal carries are latched (ie non ripple



Figure(4.3.1.2.) A Totally Parallel Ripple-Through Complex Multiply Architecture.

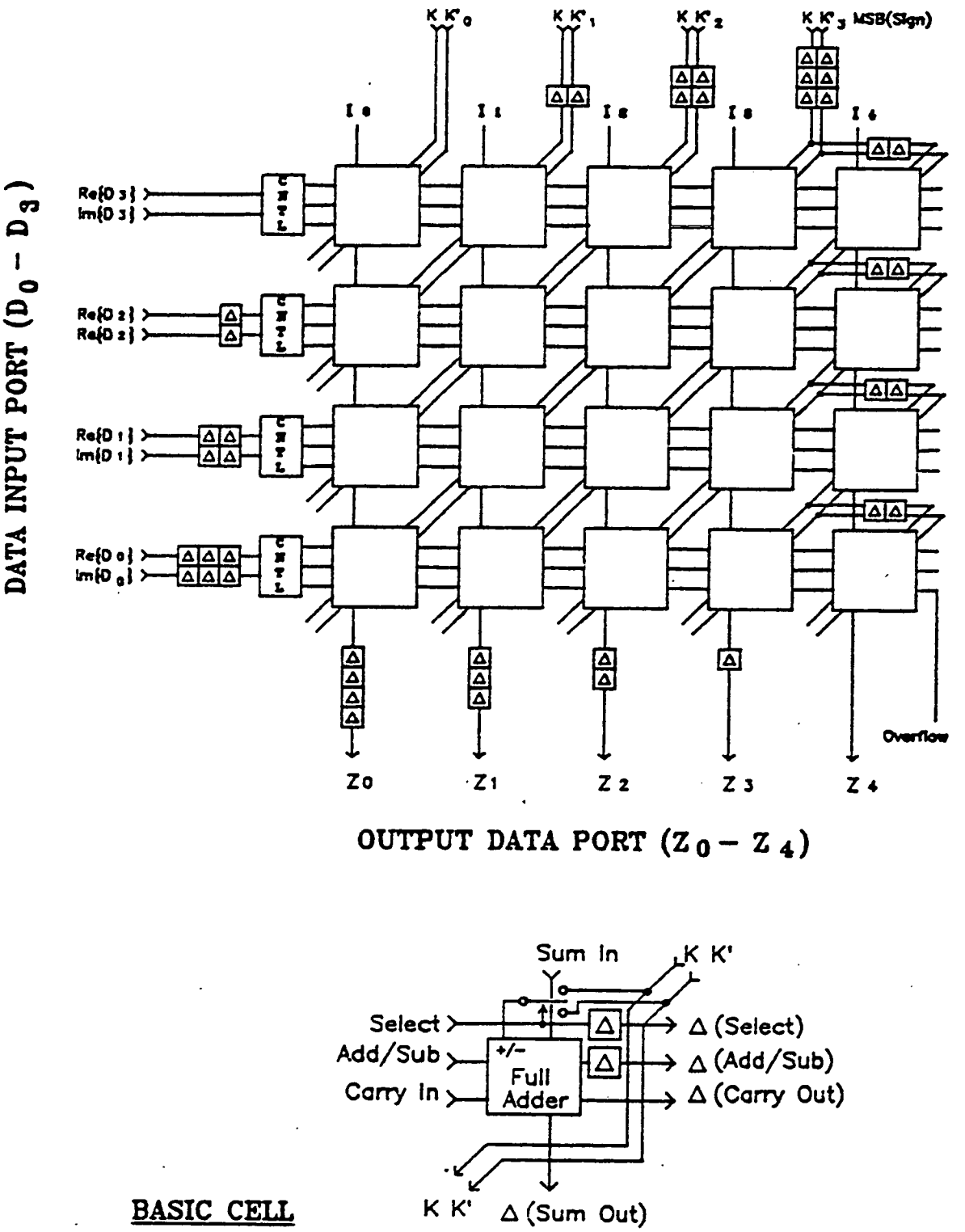
through) then it would be neccessary to place a skew on the input data port to ensure match up of data with carry formation. A quite different approach, however, would be to consider feeding the carries as well as the sums from one row to the row beneath, treating the carries as being sums of double significance (which is essentially what a carry from a full adder cell is). This then allows deferal of carry formation until the output of the array where sums and carries of equal significance appear. At this point, some form of fast adder would be needed to assimilate the sums and carries of equal significance. A complete structure which implements the distributed arithmetic complex multiply based on this carry deferal approach is shown in Figure (4.3.1.3) together with the basic cell that would be needed to implement the specific distributed arithmetic complex multiply algorithm. In addition to the full adder this involves a data-selector and some shift registers to delay K and K^*, which are the distributed arithmetic representations of the complex coefficient. The advantage of this structure is that it is not necessary to skew input (or output) data to (and from) the array, however, the fast add requirement does impose some degree of wordlength dependence on the speed performance. In this structure, the sums and carries are fed forward in such a way as to effect a scale down by a factor of two from one row to the next. This allows K and K^* to be fed down through the array vertically so that it is added in or subtracted at a different level of significance relative to the sums and carries at each row of cells. It will also be noted that this structure allows two bits of array initialisation per cell at the input. One bit could be used for input of a fixed rounding word and the other bit could be used for the initialisation required by the


DISTRIBUTED ARITHMETIC
COEFFICIENT INPUT PORT
DATA INPUT PORT (D0 – D3)
K K'0
K K'1
K K'2
K K'3
"0"
I0
I0 I1
I1 I2
I2 I3
I3 I4
I4
Re{D0}
Im{D0}
Re{D1}
Im{D1}
Re{D2}
Im{D2}
Re{D3}
Im{D3}
CNTL
Carry
FAST PARALLEL ADDER
Cin
Overflow
Z0
Z1
Z2
Z3
Z4
OUTPUT DATA PORT (Z0 – Z4)
Carry in
K K'
Sum in
Select
Select
+/–
Full
Adder
Add/Sub
Add/Sub
Δ(Sum Out)
Δ(Carry Out)
Δ(K K')
BASIC CELL

Figure(4.3.1.3.)

algorithm as described in (3.5.10) and (3.5.11). This type of structure is particularly compact however it is not highly suited to very large wordlengths because of the fast add requirement.

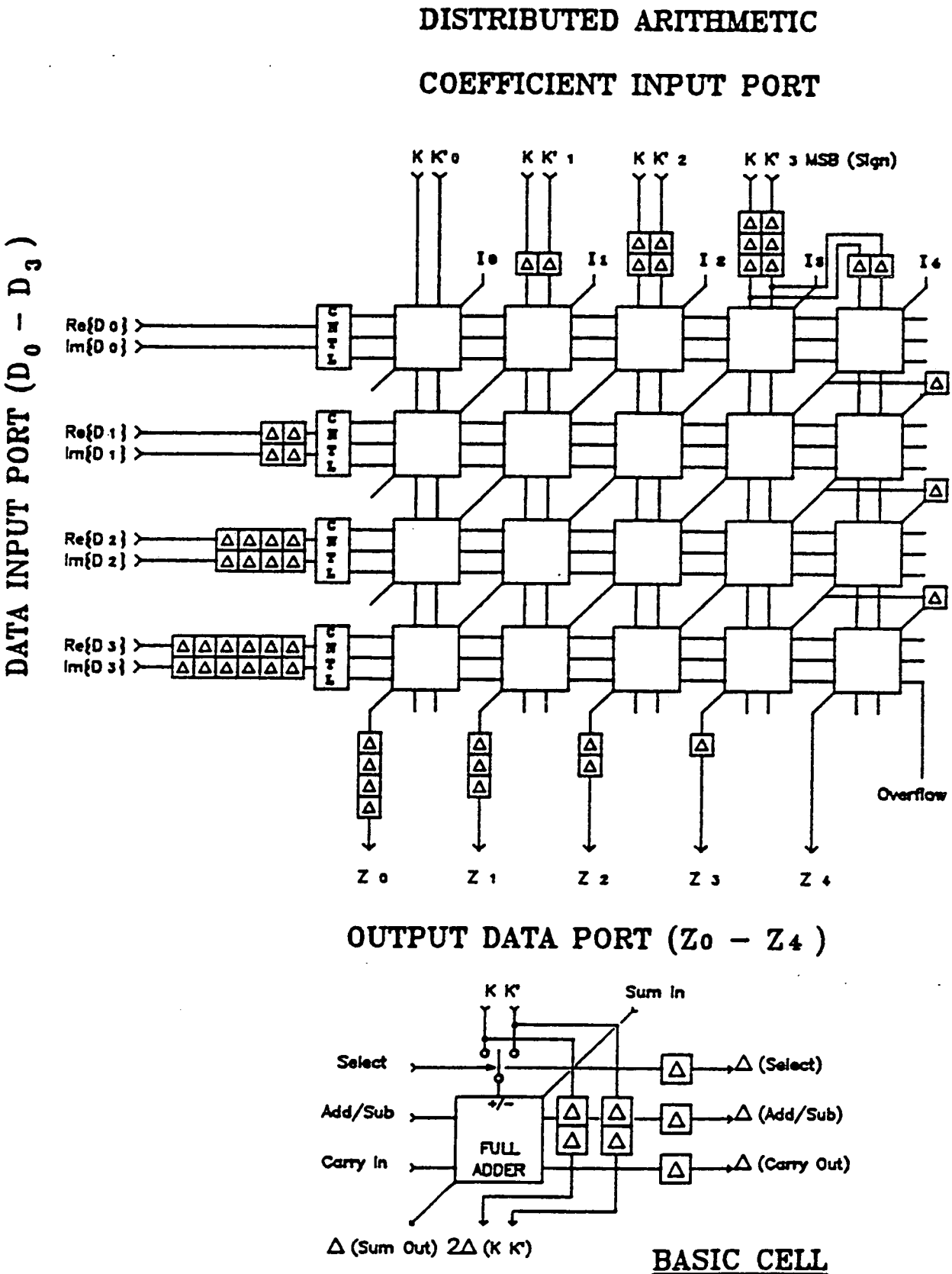
One way to overcome this wordlength dependence is, as just mentioned, to latch the carries horizontally in an array of full adders. This also entails latching the horizontal control data which comprises the data-select control and the add/subtract control. If, however, instead of scaling down the sums and carries at each row, K and K^* are scaled down instead then it is not neccessary for a delay to be placed on them at each cell. Thus the shift register count is not increased at all. It is, however, neccessary to skew all data entering the array and to perform a de-skew operation at the output. The speed of this type of structure, shown in Figure (4.3.1.4) will not suffer the same degree of wordlength dependence as the structure of Figure (4.3.1.3), which has a fast add requirement, however, K and K^* have to be fed through the array without being latched so the maximum load placed on the source buffer which supplies (MSB's of) K and K^* will increase linearly with wordlength. This of course does not imply a linear decrease in speed with wordlength because the buffers that supply K and K^* can be engineered to work optimally into a given load. Also the delay from other circuits such as the full adder will tend to have the dominating influence on the overall speed. To summarise, the structure of Figure (4.3.1.4) will exhibit some degree of wordlength dependance on overall speed performance but this can be kept fairly small. The top row of this structure has one bit of initialisation per cell thus allowing the initialisation requirements of the distributed


DISTRIBUTED ARITHMETIC
COEFFICIENT INPUT PORT
K K'0
K K'1
K K'2
K K'3 MSB(Sign)
I 0
I 1
I 2
I 3
I 4
DATA INPUT PORT (D0 - D3)
Re{D 3}
Im{D 3}
Re{D 2}
Re{D 2}
Re{D 1}
Im{D 1}
Re{D 0}
Im{D 0}
Overflow
Z0
Z1
Z2
Z3
Z4
OUTPUT DATA PORT (Z0 - Z4)
Sum In
K K'
Select
Add/Sub
Carry In
+/-
Full
Adder
Δ (Select)
Δ (Add/Sub)
Δ (Carry Out)
K K'
Δ (Sum Out)
BASIC CELL

Figure(4.3.1.4.)

arithmetic complex multiply algorithm to be met. If, however, it was required to add a small fixed number to minimise noise from the truncated K and K^* words a sparse row of half adders would be required to add this to the initialisation word. This is detailed in the following chapter which considers some specific implementations. It should be noted that the initialisation word needs to be skewed in this structure as with K and K^*. The initialisation word is derived from K or K^* so extra shift register delays are not required. The basic cell for this structure is fairly small thus resulting in a fair compromise between clocking rate and overall area.

For very large wordlengths, it is desirable to seek a structure which exhibits essentially no wordlength dependence with all cell communication being latched. One such structure is shown in Figure (4.3.1.5). Here, the carries are fed horizontally and the sums are fed down and shifted to reduce their significance in going from one row to the row beneath. This results in a vertical delay through the cell of two cycles which is the delay that must be applied to K and K^* which are fed vertically down through the cell. As the vertical delay is two cycles per cell, the delay on data entering the control input data port must increase by two cycles in moving from one row to the row beneath. That is to say, the equivalent of two series data skew operations must be performed. Figure (4.3.1.5) shows the basic cell that this structure entails. The shift register count at first sight appears to be rather large, however the vertical delay on K and K^* can be implemented with half the number of shift registers clocked at half the normal rate as described in the next


DISTRIBUTED ARITHMETIC
COEFFICIENT INPUT PORT
DATA INPUT PORT (D0 - D3)
K K' 0
K K' 1
K K' 2
K K' 3 MSB (Sign)
I0
I1
I2
I3
I4
Re{D 0}
Im{D 0}
Re{D 1}
Im{D 1}
Re{D 2}
Im{D 2}
Re{D 3}
Im{D 3}
CNTL
Overflow
Z 0
Z 1
Z 2
Z 3
Z 4
OUTPUT DATA PORT (Z0 - Z4)
K K'
Sum In
Select
Add/Sub
Carry In
+/-
FULL ADDER
Δ (Select)
Δ (Add/Sub)
Δ (Carry Out)
Δ (Sum Out) 2Δ (K K')
BASIC CELL

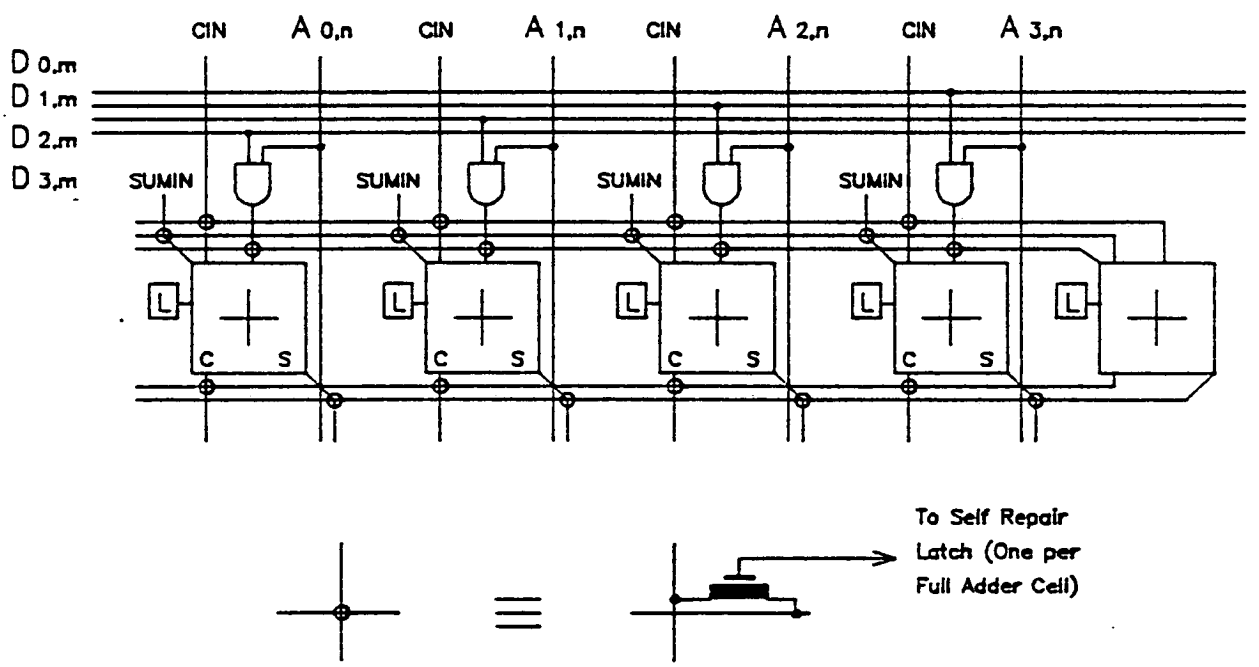
Figure(4.3.1.5.)

chapter which looks at a specific implementation. The structure of Figure (4.3.1.5) has a clocking rate which has no wordlength dependence other than any delays that might be incurred in transmitting the clocks to each cell. This is indeed a problem for very large wordlengths and demands the set up of a hierarchy of clock buffers to distribute the clock with a minimum of delay. The possibilty of abandoning the synchronous structure and using a self-timed approach would be one way of avoiding clock distribution problems, however, this would involve a large area penalty and thus require yield enhancement for even modest wordlengths.

It was mentioned in the previous chapter that the distributed arithmetic coefficients can be computed in situe to produce highly regular structures which are suitable for yield enhancement techniques. Figure (4.3.1.6) shows a basic cell with yield enhancement that could be used in a large distributed arithmetic array to compute a four term linear equation in parallel.

4.4. Summary

In this chapter a number of distributed arithmetic structures have been suggested for implementing the complex multiply as part of a high bandwidth datapath. The next Chapter is devoted to some specific high bandwidth arithmetic processors for FFT computation, which use some of these distributed arithmetic structures to streamline the computation of the complex multiply function which is the most intensive computational requirement of the FFT and DFT.



Note : Each adder in this block operates on data of same significance
Thus, shifts that are required, such as for sums, must be performed
from one such block of cells to another block of cells (sets of 4 in this example)

Figure(4.3.1.6.) Pseudo-Distributed Arithmetic High Level Cell showing Possible Yield Enhancement Mechanism. (This computes a four term linear equation).

References

1. I. R. Mactaggart, "A VLSI Radix-2 FFT Butterfly," MSc Project Report MSP6, (September 1981).

Chapter 5 - MOS - LSI/VLSI Distributed Arithmetic Structures

5.1. Introduction

In this chapter, some specific MOS - LSI and VLSI implementations of the architectures discussed in the previous chapter will be presented. The structures described here offer efficient high speed computation of the complex multiply function through the use of distributed arithmetic.

The first silicon device to be described (number EU201) is a totally parallel radix-2 FFT butterfly arithmetic processor, however, in a later design, greater versatility is obtained by allowing dynamic re-configuration of the datapath itself. Of the three chips described in this chapter the first two devices (nMOS) were fabricated and tested. The last design to be described was due for fabrication on a GEC 4 micron CMOS-SOS process but due to poor Applicon software to allow transfer of the design, this was delayed, making the fabrication and testing of the design impractical within a reasonable time frame. Test results if available may be added as an appendix to this thesis. The testing of the second nMOS device (EU219) is covered in some detail, as a number of working samples were obtained. The first device (EU201) which is similar to the second device could not unfortunately be tested in depth due to a limited number of samples being available and also a low process yield was in evidence. This may have been related to very high depletion thresholds (around -1.0 V) which were measured during probe testing of some test structures in the chip frame. The unfabricated SOS design is described and documented to allow

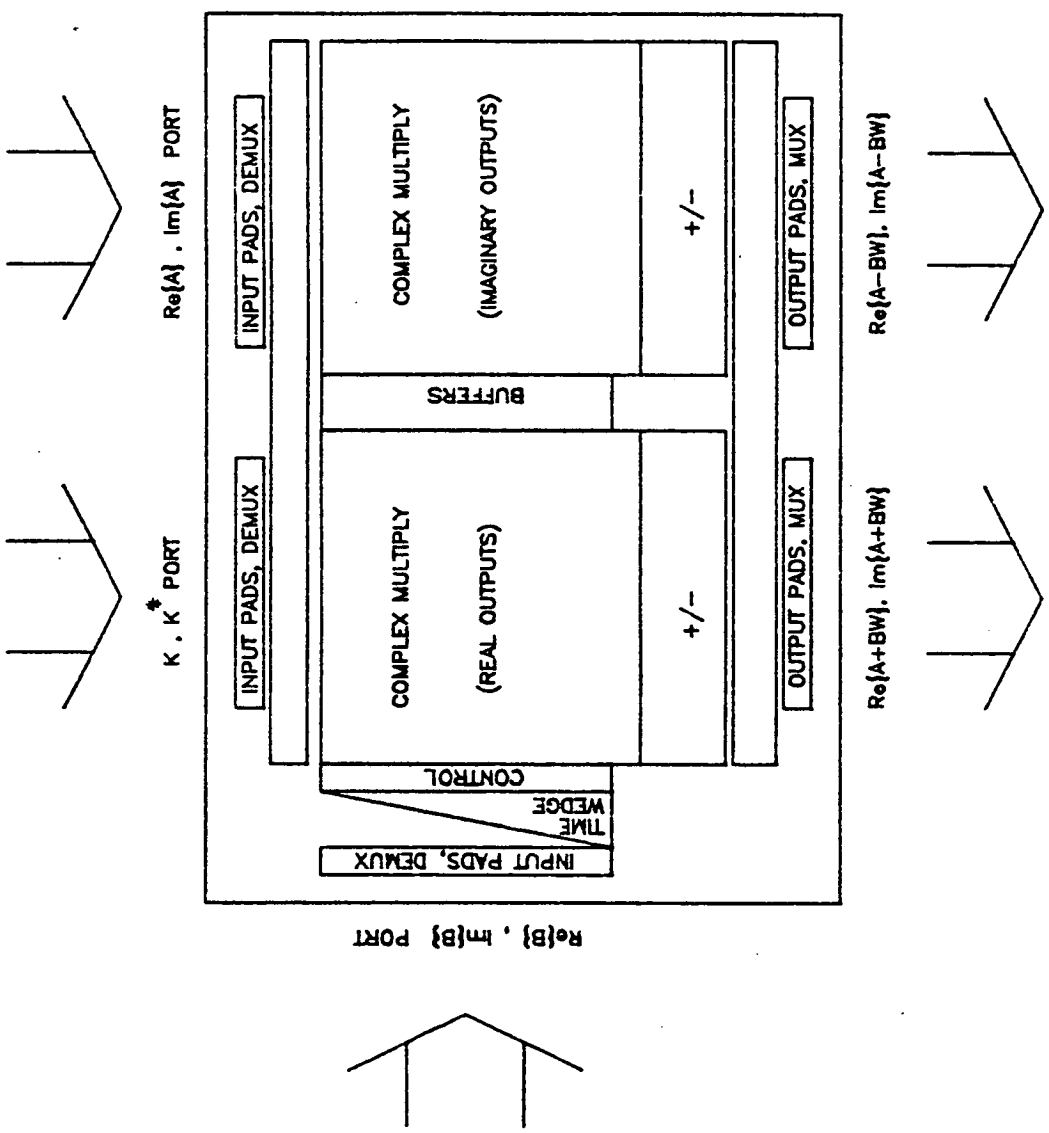
subsequent testing and application in a system.

5.2. A Totally Parallel 6 Bit Radix-2 FFT Butterfly

5.2.1. General

To compute the radix-2 FFT butterfly in parallel requires that the complex multiply be implemented in parallel. The radix-2 decimation-in-time butterfly function requires that two data input words (A,B) (complex) be modified by a coefficient (W) (complex) to form (A+BW) and (A-BW) (complex). The complex multiply distributed arithmetic algorithm described in Chapter 3 can be implemented as two distinct distributed arithmetic arrays, after Figure (4.2.4) one producing real data and the other producing imaginary data, or alternatively, a single array can be constructed after the totally parallel example of Figure (4.2.5). It was decided to pursue a structure after the former as this would allow a natural progression to a programmable device if required, with a larger word length - producing real or imaginary data but not both simultaneously. One such device is described in the next section (EU219).

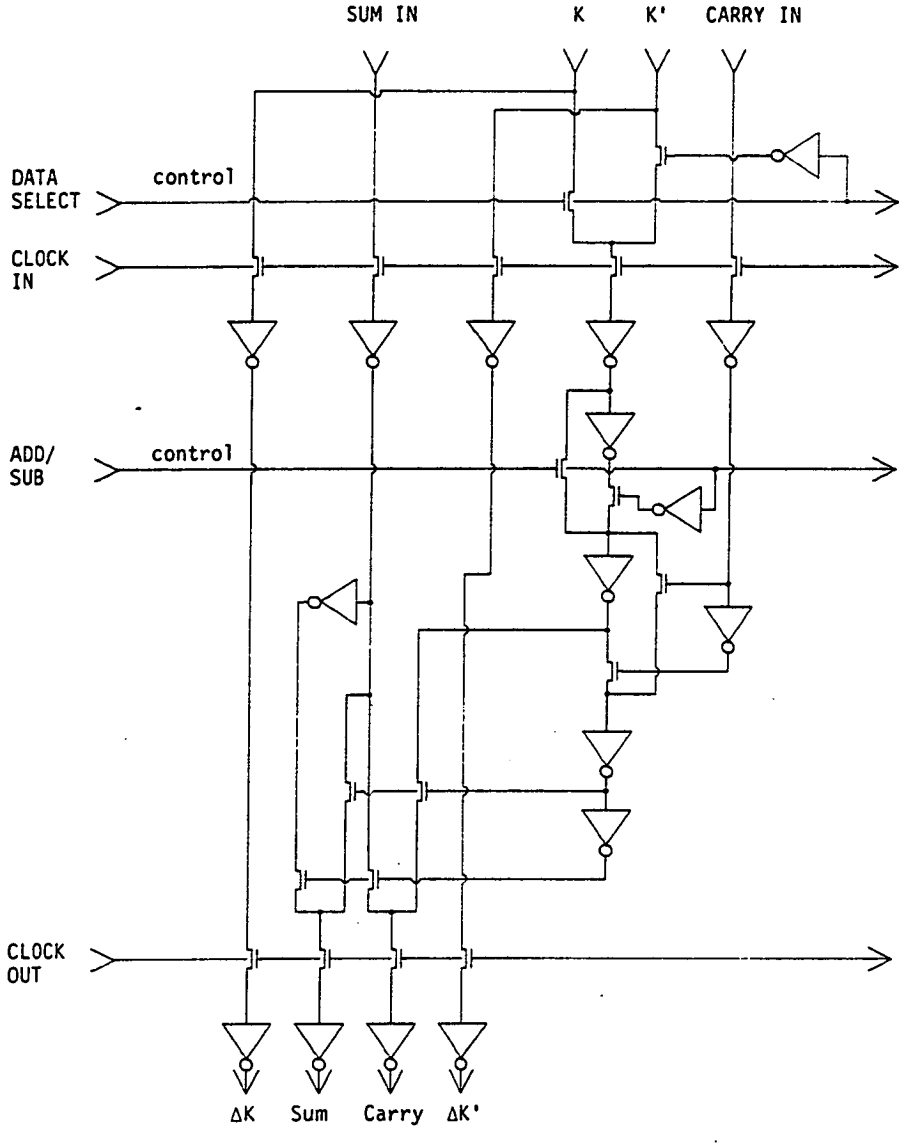
Having computed the complex product (BW) an additional adder and subtractor can be used to compute the required butterfly outputs (A+BW) and (A-BW). This resulted in the chip floorplan shown in Figure (5.2.1.1). It was decided to limit the wordlength of this prototype chip to 6 bits for both data and coefficient. This is too small a wordlength for most system applications but it was hoped to obtain some yield data which might indicate whiether longer wordlengths could be considered in the future. At this small wordlength most of the larger and more highly pipelined structures discussed in the previous chapter do not offer much greater



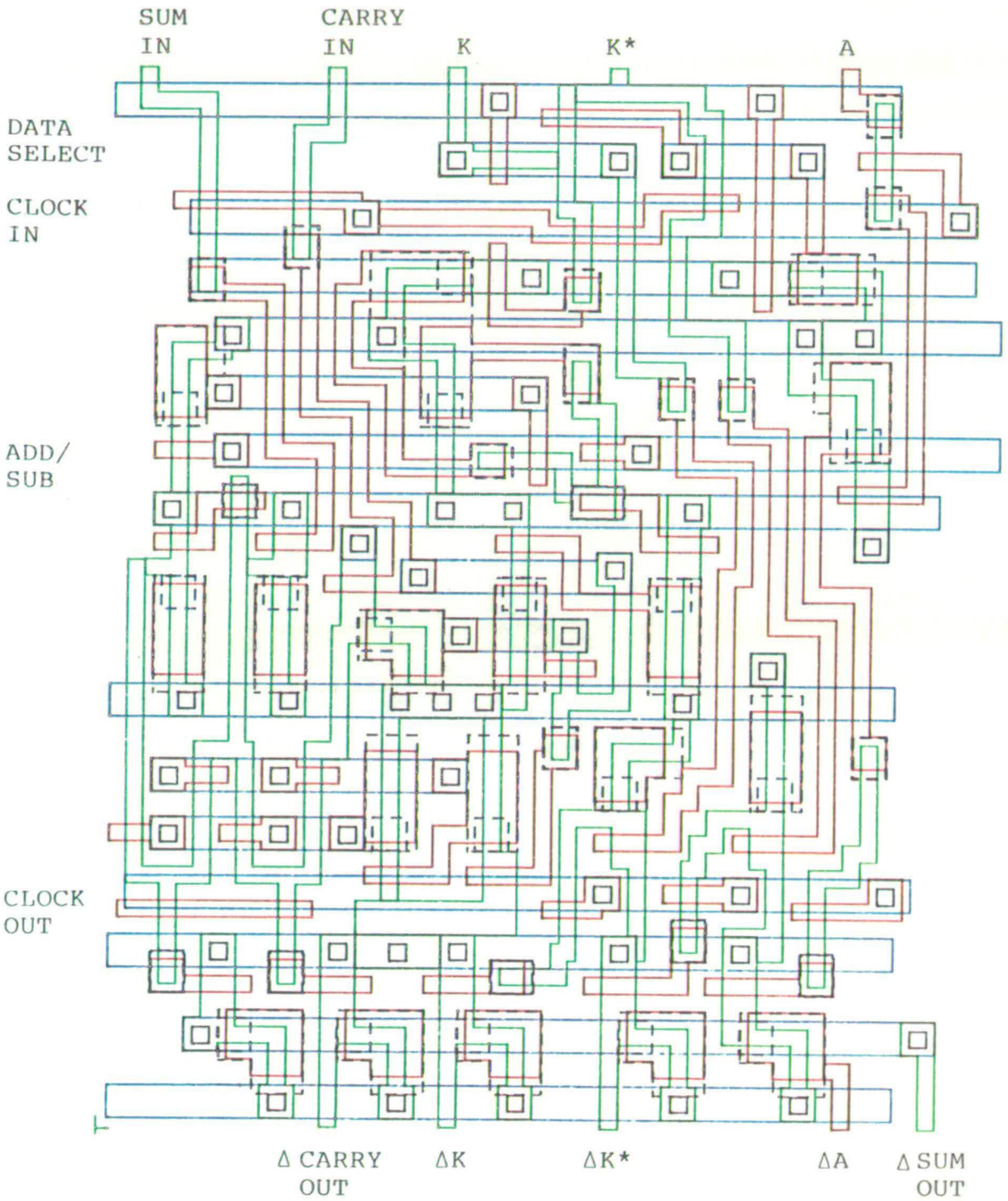
Figure(5.2.1.1.) Floorplan of Totally Parallel Radix-2 FFT Butterfly (EU201).

bandwidth and so the carry deferal architecture described in Figure (4.3.1.3) may be considered to be highly suitable, even though this structure requires a fast adder to assimilate the sums and carries at the output. Other than this fast add requirement, the distributed arithmetic complex multiply structure may be made up from an array of identical cells each of which fulfil the three main requirements of the distributed arithmetic complex multiply algorithm. The basic cell must be capable of performing a data-select and bit level full add as well as containing delays for the pipelined operation. The precise logic of the basic cell used, together with nMOS silicon layout is shown in Figures (5.2.1.2) and (5.2.1.3) respectively.

This cell permits selection of either K or K^* as required by the algorithm, and the bit level addition or subtraction of this by the full adder, which is an integral part of the accumulator. Finally, it presents K and K^* delayed for operation on by the cell in the row beneath. Most of the cell area is comprised of the full adder. This made use of inverter controlled data-select type exclusive-OR gates which offers a good compromise between area, speed, and power consumption, when compared with PLA, function block, and random logic nMOS implementations, as determined by Myers [1] The latches used were standard nMOS dynamic devices which offer a lower area and power consumption figures compared to static circuitry. This places a minimum safe clocking rate of around 20 KHz on the chip as a whole, however, the minimum clocking rate may be reduced further by increasing the clock logic "HIGH" to $V_{dd} + V_{th}$ where V_{th} is the threshold voltage of the enhancement devices. The output of the distributed arithmetic array contains unassimilated



Figure(5.2.1.2.) Logic used in Basic Cell of the Distributed Arithmetic Complex Multiplier.

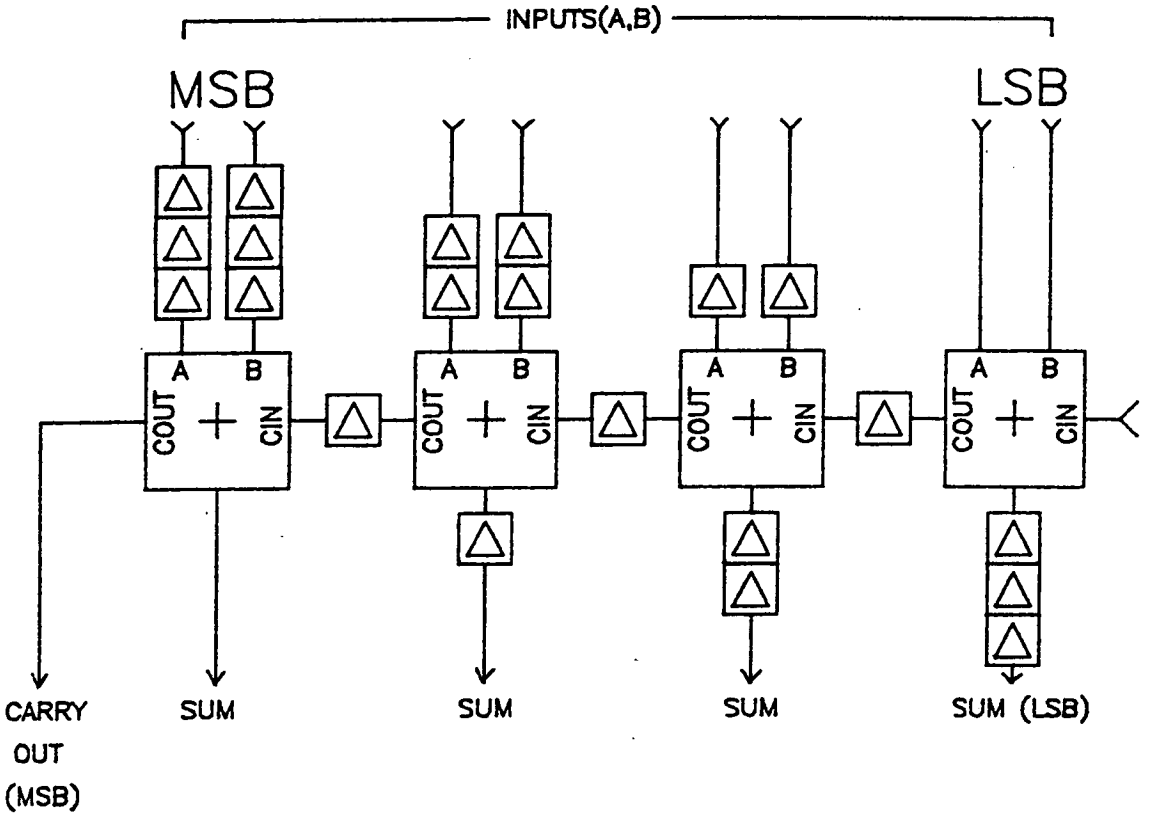


Figure(5.2.1.3.) The nMOS Layout of the Basic Cell.

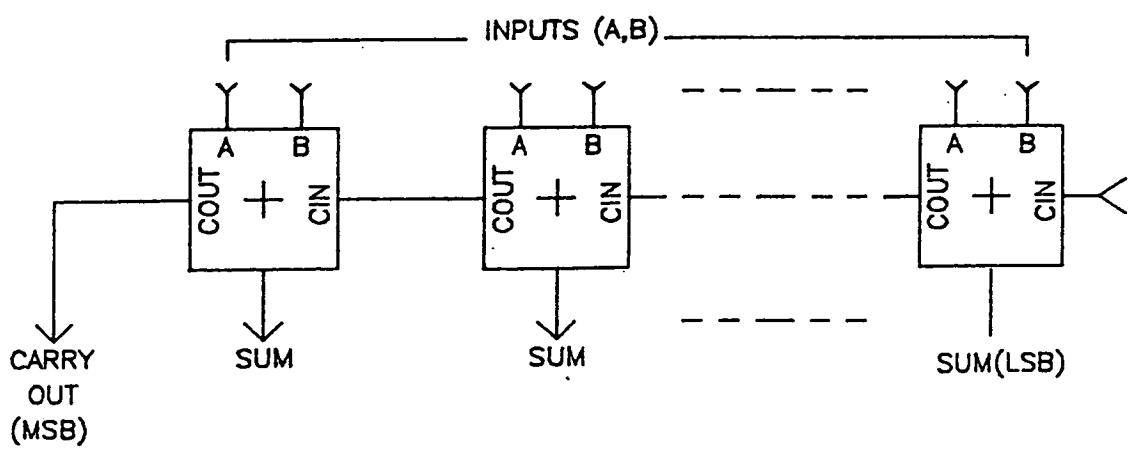
sums and carries which must be combined to produce the desired output. This may be accomplished with a fast adder.

5.2.2. Fast Adder for Data Assimilation at Array Output

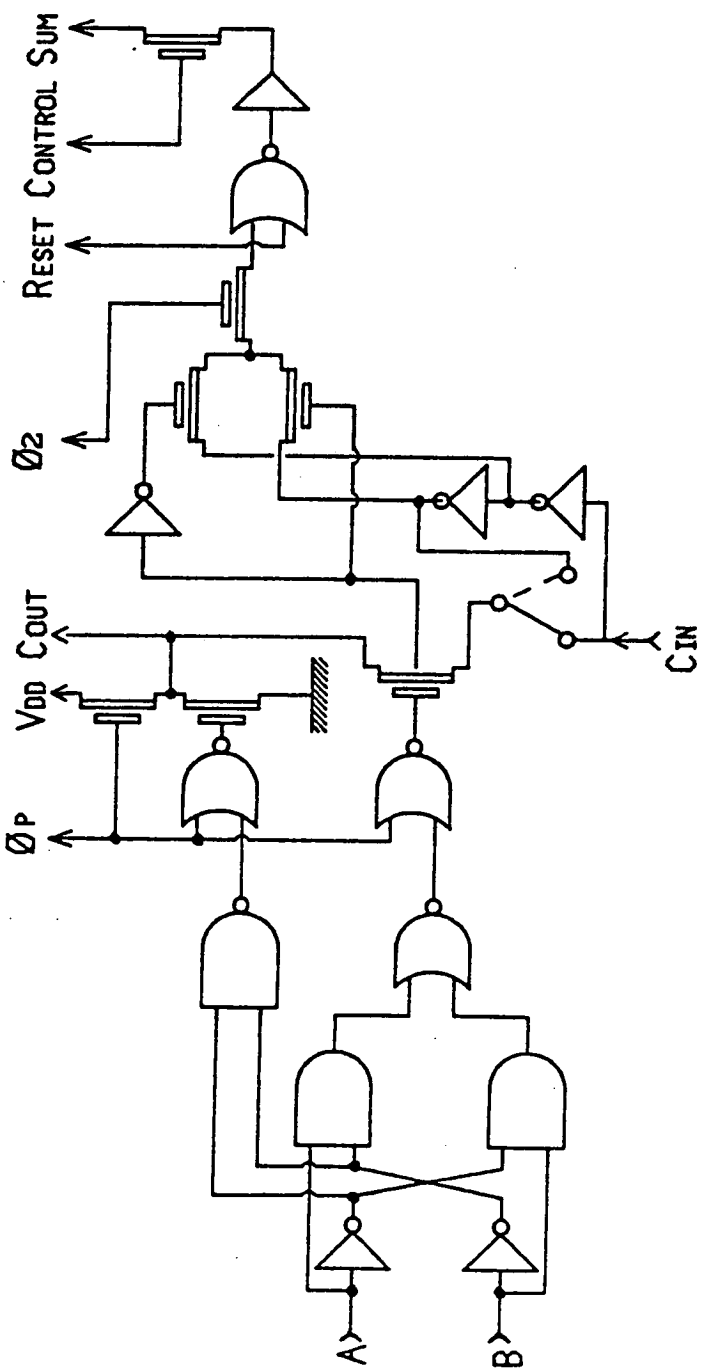
There are several techniques for achieving a fast add. It would be possible to use a high latency pipelined adder with skewed input and output data, using latched carries, as shown in Figure (5.2.2.1). This would not strictly be a fast adder, (with unity latency) but rather a high bandwidth adder (with greater than unity latency). Another approach would be to use some form of carry look ahead (CLA) technique which would offer true high speed, and could be operated with unity latency. In nMOS technology, however, it is possible to use an ordinary carry ripple-through (Figure (5.2.2.2)) adder at quite high speed if the carry chain is pre-charged to a logic "1", before forming the carries. This technique offers high speed for small to medium word lengths, but is superseeded by the CLA adder at larger word lengths (typically >12 bits). In a 6 or 8 bit device, the carry chain pre-charge technique, is likely to offer the most optimum approach then, since the ripple through adder is smaller than a carry look ahead adder. The logic used for this ripple-through adder with pre-charged carry-chain is shown in Figure (5.2.2.3) and the silicon layout used is shown in Figure (5.2.2.4) for two adders. This was in fact the final output stage of the chip. The adder uses random logic.



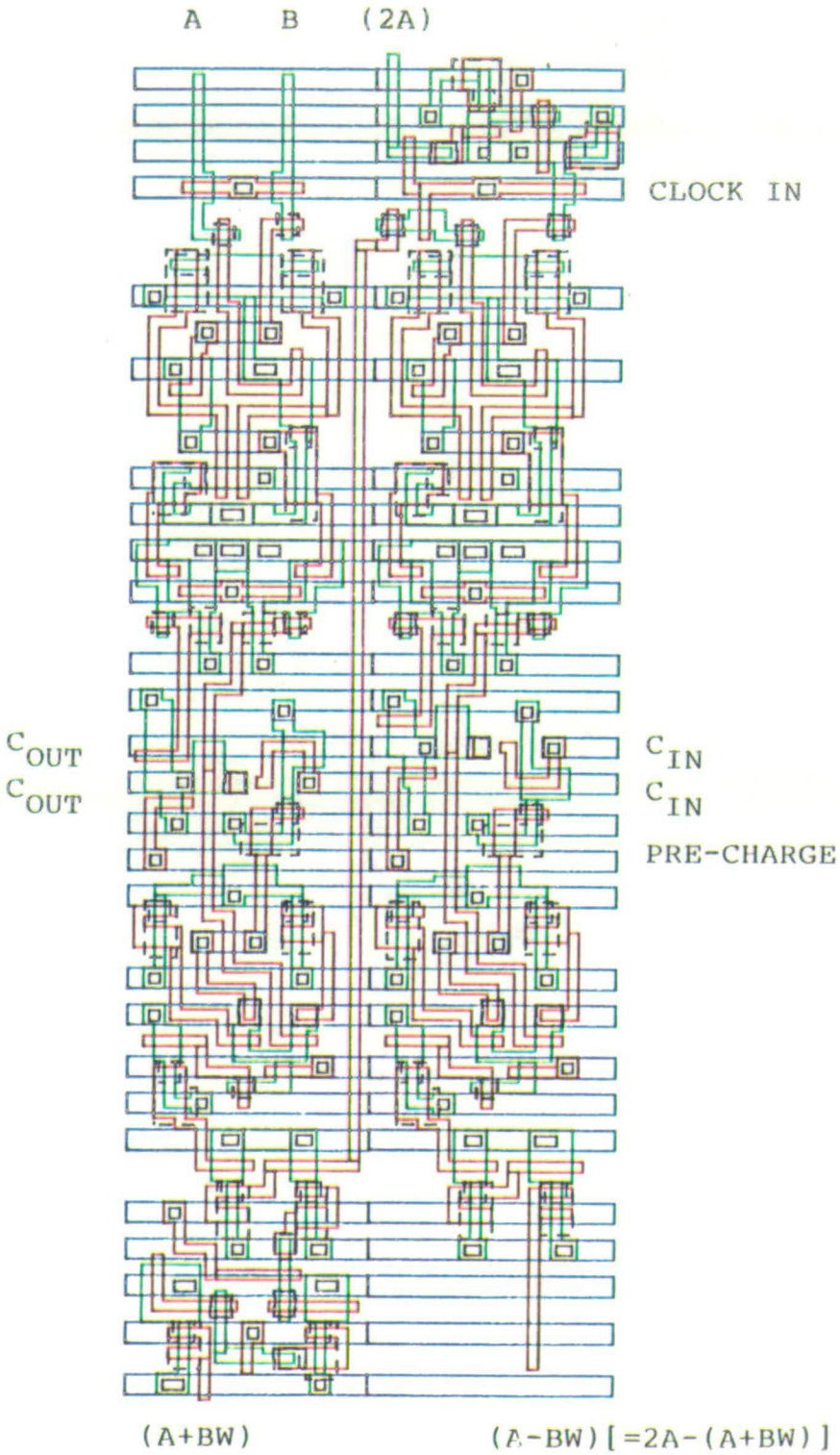
Figure(5.2.2.1.) Pipelined, High Data Throughput Adder



Figure(5.2.2.2.) Ripple-Through Adder



Figure(5.2.2.3.) Logic of Ripple-Through Adder using Pre-charged Carry Chain.



Figure(5.2.2.4.) Final Adder/Subtractor used to Generate Butterfly Outputs for EU201.

5.2.3. Clocking Scheme

It was decided to use a standard two-phase non-overlapping clock system as is quite common in nMOS technology. This clock could be conveniently used to control de-multiplexer circuitry and multiplexer circuitry to allow the fast transfer of both real and imaginary complex data within a complete clock cycle at each of the five input ports. The use of a non-overlapping clock eases the design problems of minimising clock skews and keeping control over internal clock rise times throughout the chip, in order to avoid race conditions in shift register circuitry.

5.2.4. Performance of EU201

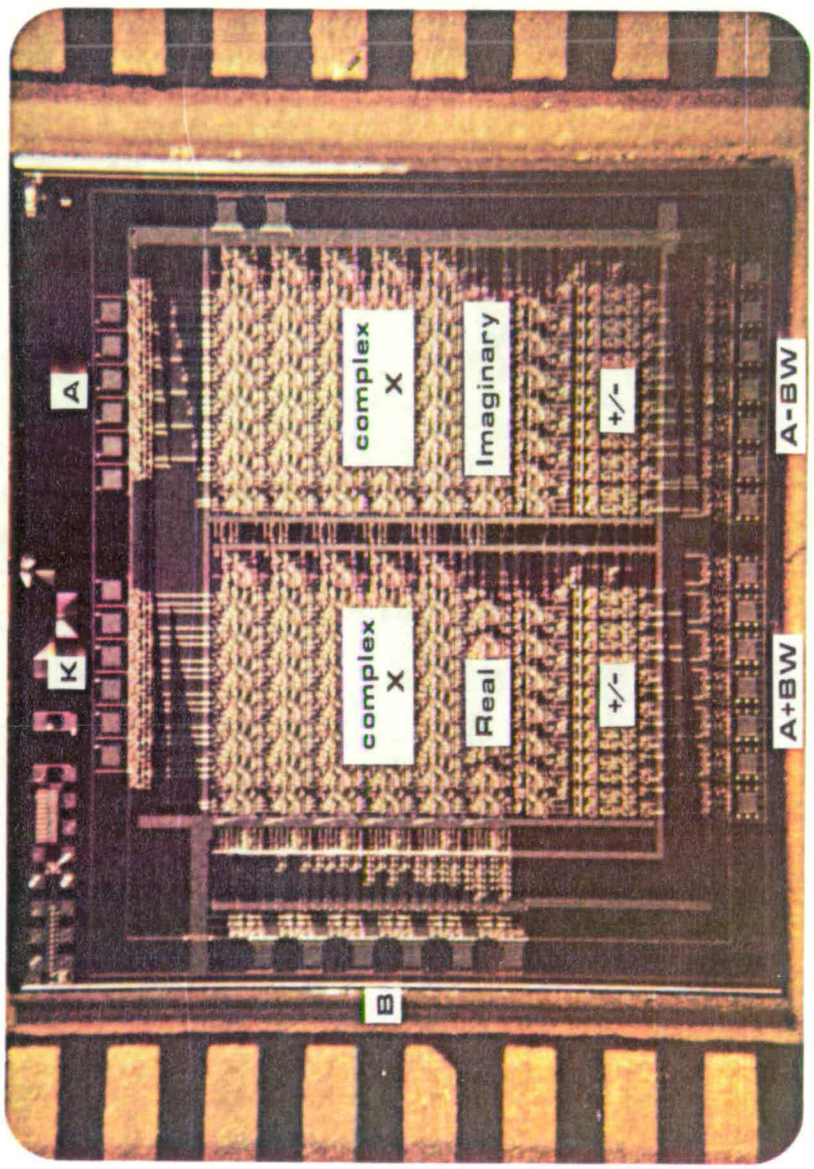
The chip EU201 was not probe tested but rather was bonded up and ten such devices were made available by the Edinburgh Microfabrication Facility (EMF) for testing. It was found that one of the chips produced entirely correct real results and another produced entirely correct imaginary results. The design was therefore verified, however, the yield was not high enough to produce a completely working sample out of the ten samples.

The depletion thresholds in EU201 were too high (-1.0 V typical), thus slowing the internal logic rise times. No maximum speed performance check was therefore made, however, no problems were encountered at 1MHz on those circuit parts which appeared to operate at the lower clock rates used initially. (At a clock rate of 4MHz, six EU201 chips incorporated into a pipelined FFT would compute a 64-point transform in only 48 microseconds). As the chip was not

available in sufficient quantities to get suitable yields (wafer shared with another device), in-depth testing of this device was not carried out. The device to be described next however which is based on the same basic distributed arithmetic array as EU201, was available in somewhat larger quantities and several working devices were obtained. Extensive testing of this device was therefore possible.

5.2.5. Summary of EU201 Butterfly Processor

The device just described, was designed to compute the radix-2 Decimation-in-time FFT butterfly function, completely in parallel. It did not require any external control therefore. A photograph of the fabricated device is shown in Figure (5.2.5.1). The ability to re-configure the datapath, however, could offer a great potential in extending the range of functions that could be computed. This approach could be extended to allow a single datapath to be programmed to compute either real or imaginary outputs of the complex multiply. This approach is useful, as the area of circuitry is halved, thus allowing larger word lengths to be implemented. The next chip to be described can be programmed to compute either real or imaginary outputs of the radix-2 butterfly, and offers a larger wordlength.



Figure(5.2.5.1.) Photograph of EU201 Totally Parallel Radix-2 FFT Butterfly, with Labels.

5.3. EU219 - A Programmable 8 Bit Version of EU201

5.3.1. General

The previous section described a totally parallel radix-2 butterfly which was based on a distributed arithmetic complex multiply algorithm. The array used to compute real data and the array used to compute imaginary data are identical, with fixed internal control determining whiether real or imaginary data is produced from each array. It is possible, therefore, to program a single distributed arithmetic array to produce either real or imaginary data. This partition would of course, hàlf the processing bandwidth per chip, but allows the possibility of larger word lengths. This section describes a single programmable distributed arithmetic array which can output either real or imaginary data every computation cycle. The device, (number EU219), operates on a more practical 8 bit data word, and can be interfaced directly with an identical device to recover (that is, double) processing bandwidths. This is achieved through the use of a high speed multiplexer and tri-state output pads on the chip, enabling the interleaving of real and imaginary output data in time-sequence.

5.3.2. High Speed Input and Output Port

It was decided, in order to avoid external data sorting, and to keep pin counts as low as possible, that a single 8 bit port would be used for DATA IN and a separate 8 bit port for DATA OUT. The coefficient would have a separate data input port also, as before. In the previous section, it was described how the distributed

arithmetic coefficients K and K^* were fed vertically down into the array from the top of the chip. The input to the butterfly (A) which is not rotated (not passed through the complex multiplier) was fed vertically down through the chip, being delayed for subsequent addition to +/- BW at the output. The other data input (B) to the butterfly which is rotated, was fed in horizontally from a separate input port where it was passed to the control logic for controlling the array. If A and B input ports were to be absorbed into a single data port, then awkward bus interconnections would be required to connect vertical and horizontal accesses to the array. It was decided that the problem could be solved by feeding in the coefficient (W) in horizontally to the control logic and feeding in data (A,B) at the top of the array. Unfortunately, if data B is fed in at the top of the array, then it must be presented in the form of K and K^* ([Re{B}+Im{B}]/2 and [Re{B}-Im{B}]/2), as was done for W in the previous case. W is now fed in as Re{W} and Im{W} at the control data port. An add and subtract must therefore be performed at the top input port on all Re{B} and Im{B} data. It was decided that this would result in a more regular floorplan and reduced chip area than would the bus interconnections required otherwise, and would also mean that all data to the chip was in the form of Real and Imaginary data only, thus simplifying the coding of coefficient data in ROM. The use of a single data port also meant the need for a high speed de-multiplexer to separate the four input data words to the butterfly (Re/Im, A and B). Steps must be taken to ensure that this does not limit the performance of the chip and does not cause problems with interfacing to the outside world. The single 8-bit output data port also demands the availability of a high speed multiplexer.

5.3.3. High Speed Multiplexer

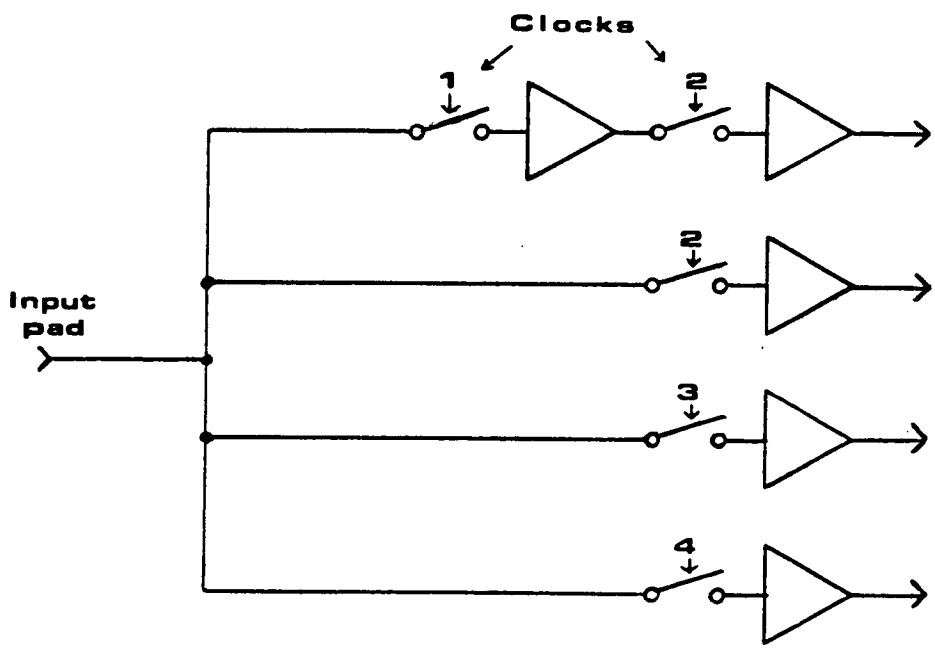
The chip required a 1:4 de-multiplexer at the 8 bit input port, and a 4:1 multiplexer at the output port, which would be used as a 2:1 multiplexer to allow real/imaginary data interleaving, with another device.

The standard de-multiplexer or multiplexer consists of two basic elements. These are a N to 2^N decoder, and a 2^N row of switches, only one of which can be placed "on" by the decoder at any one time. Finally the selected data must be latched. The decoder, however, would involve a minimum of 3 gate delays in nMOS technology. These are, input buffering (2), and logical NOR gate for implementing the AND decoder function (1). An additional output buffer would normally be required, however, further increasing the overall decoder delay time. An additional problem that occurs when decoders are used in a multiplexer and de-multiplexer is that when the decoder changes state, glitches inevitably result during that short time interval and can be quite severe. This is caused by the decoder logic gates receiving transient analogue data from poorly defined inputs which occurs during input transitions. This effect can be removed by masking the decoder output ("AND" function) so that the decoder output would be reset to zero during the time interval when the input logic to the decoder was changing. It was felt that the extra time penalty required for this decoder operation would be too great, so it was decided that a decoder would not be used at all, and instead the de-multiplexer would be directly controlled by a global four phase non-overlapping clock, some of the phases of which could also be put to use elsewhere in the chip for pre-charging

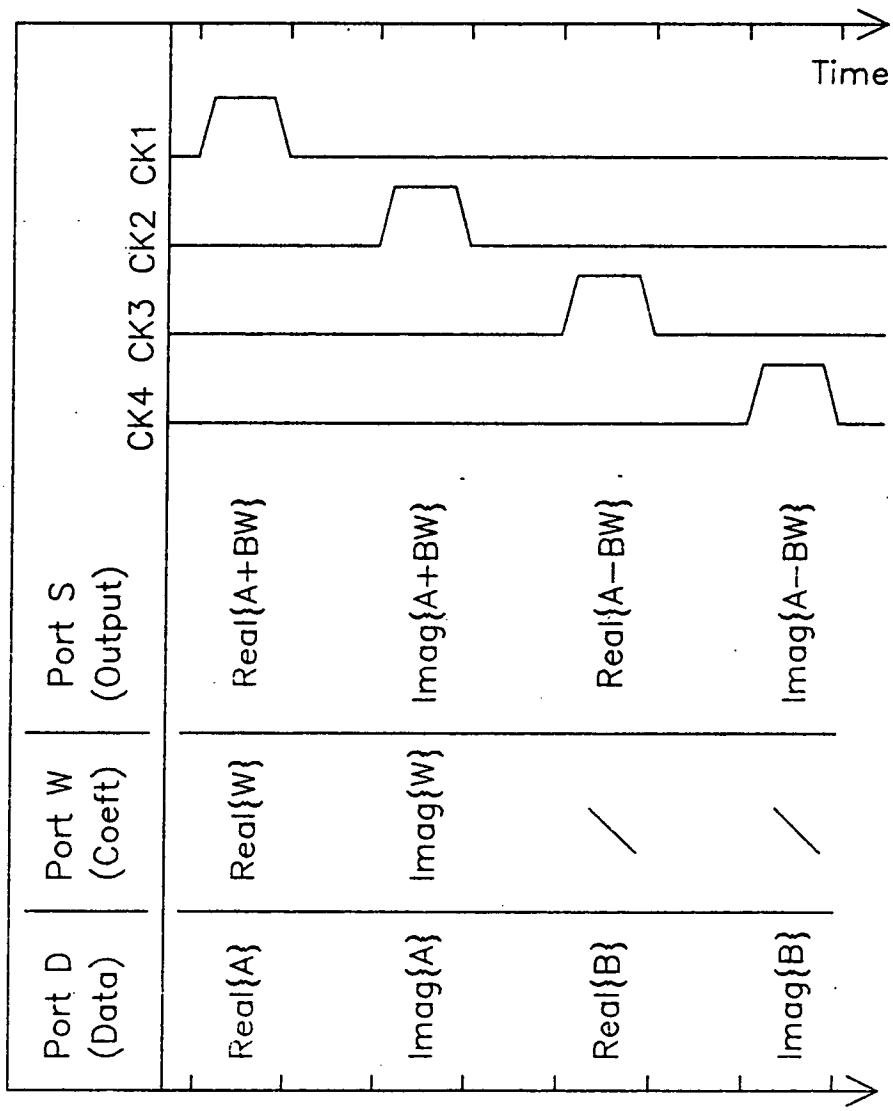
purposes and controlling latches in the array. The multiplexer would be operated by the same clock control lines. The actual logic used in these circuits was very simple. Dynamic logic was used to allow fastest possible operation, and only a single gate delay was involved in the multiplexer itself. Figure (5.3.3.1) shows the "logic" used.

5.3.4. Global Four Phase Clock

The four-phase clock was chosen, primarily for fast multiplexing and de-multiplexing of data, however, it also made available highly optimised timing signals required by those circuit elements which required pre-charging as is quite often useful in nMOS technology for speed enhancement. The fast adder, described in the previous section, used a pre-charged carry chain. The time required for carry pre-charge, is much less than is required for carry ripple through, however. This means that the pre-charge clock should be substantially shorter than the time between the pre-charge falling edge and the output latch falling edge (add-time). The use of the first phase (CK1) for pre-charge and the last phase (CK4) for output sum latching gives the adder more time for actual addition compared with the time that is available in a standard two phase non-overlapping clock system with CK1 being used for pre-charge. Figure (5.3.4.1) shows the four phase clock used, and those phases used by the fast adder for pre-charge and output latch. It can be seen that about 75% of the cycle is made available for add time.



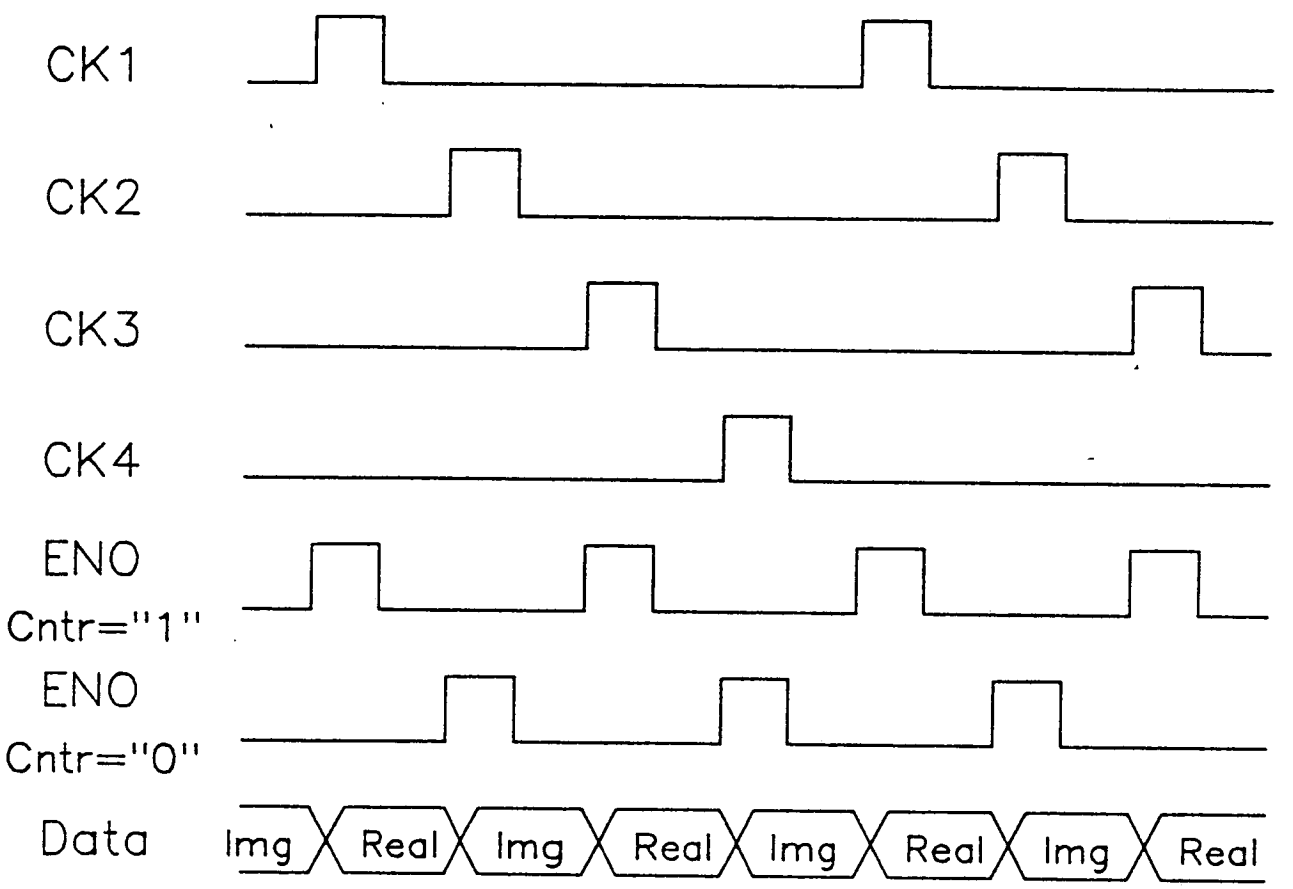
Figure(5.3.3.1.) Basic 1:4 De-Multiplexer Operating on CK1-CK4. Output is sampled on CK1.



Figure(5.3.4.1.) Four Phase Clock used by EU219. As well as providing control for high speed de-multiplexing and multiplexing, CK1 is used to pre-charge "soft" nodes.

5.3.5. Tri-State Logic and Timing Considerations

The global four-phase clock provides four time slots in which data can be latched on chip and sent to the outside world via the tri-state output pads. It was decided, that real data transfers would take place on the rising edge of clocks 1 and 3, and imaginary data transfers would take place on the rising edge of clocks 2 and 4. The reason for adopting this timing was that it would be possible to store data in memory on a complex word basis with one address per complex data word, comprising real and imaginary data segments, and then use an external multiplexer for transferring real and imaginary data to the chip in adjacent time slots. This would be necessary if the memory was not fast enough to afford a separate address for real and a separate address for imaginary data. If the device was to be programmed to output only real or imaginary data, then clearly it would be necessary to force the outputs of the chip into a high impedance state for two of the four cycles in which it has no data to contribute to the outside world. This would allow another chip to force valid data onto an external bus during those cycles. The logic used to generate the tri-state enable signal consists of a two input OR gate, which is connected to clocks 1 and 3 when real data is being output, or clocks 2 and 4 if imaginary data is being output. The timing for the tri-state enable logic is shown in Figure (5.3.5.1).

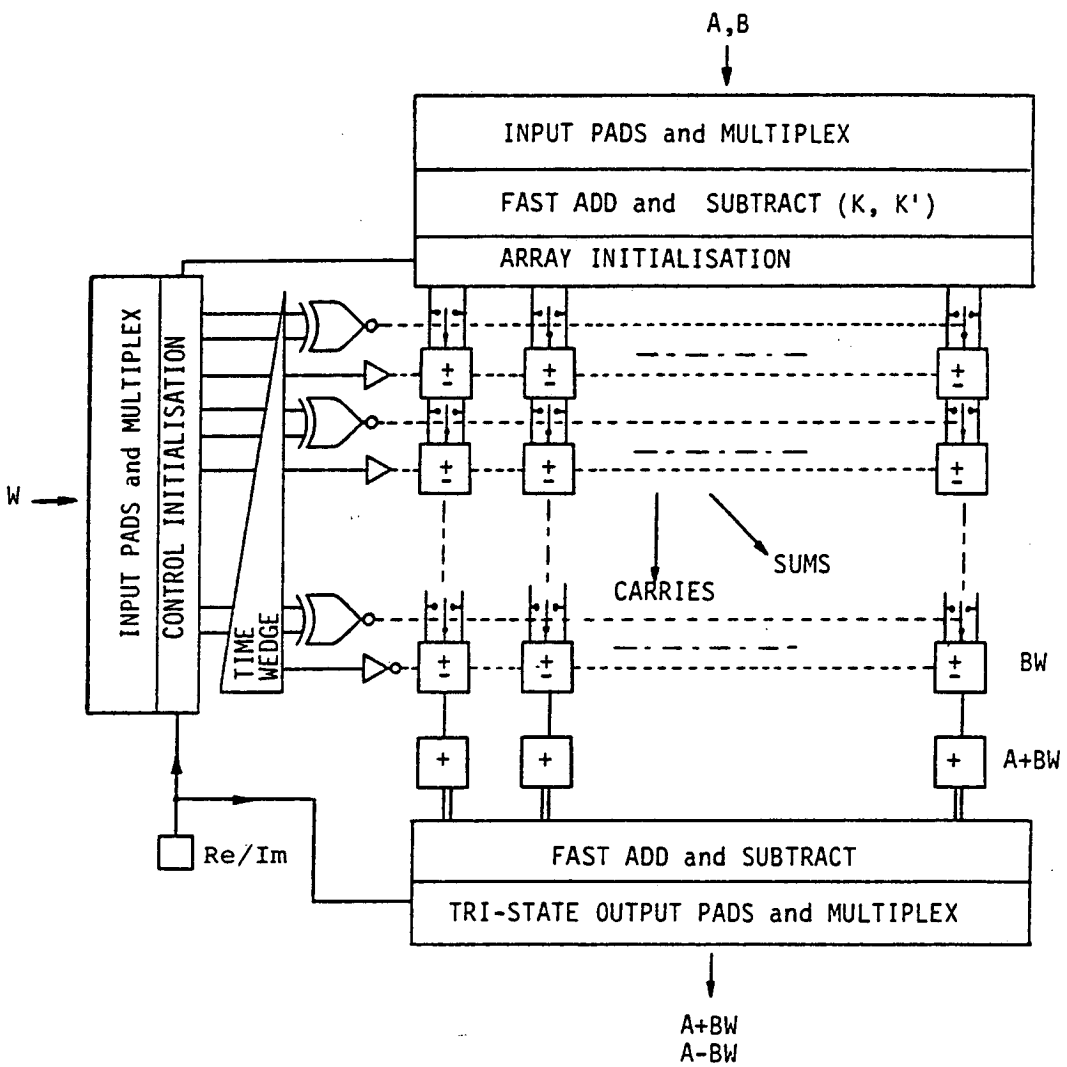


Figure(5.3.5.1.) Tri-State Enable Signals "ENO" as a function of CK1-4 and CNTR.

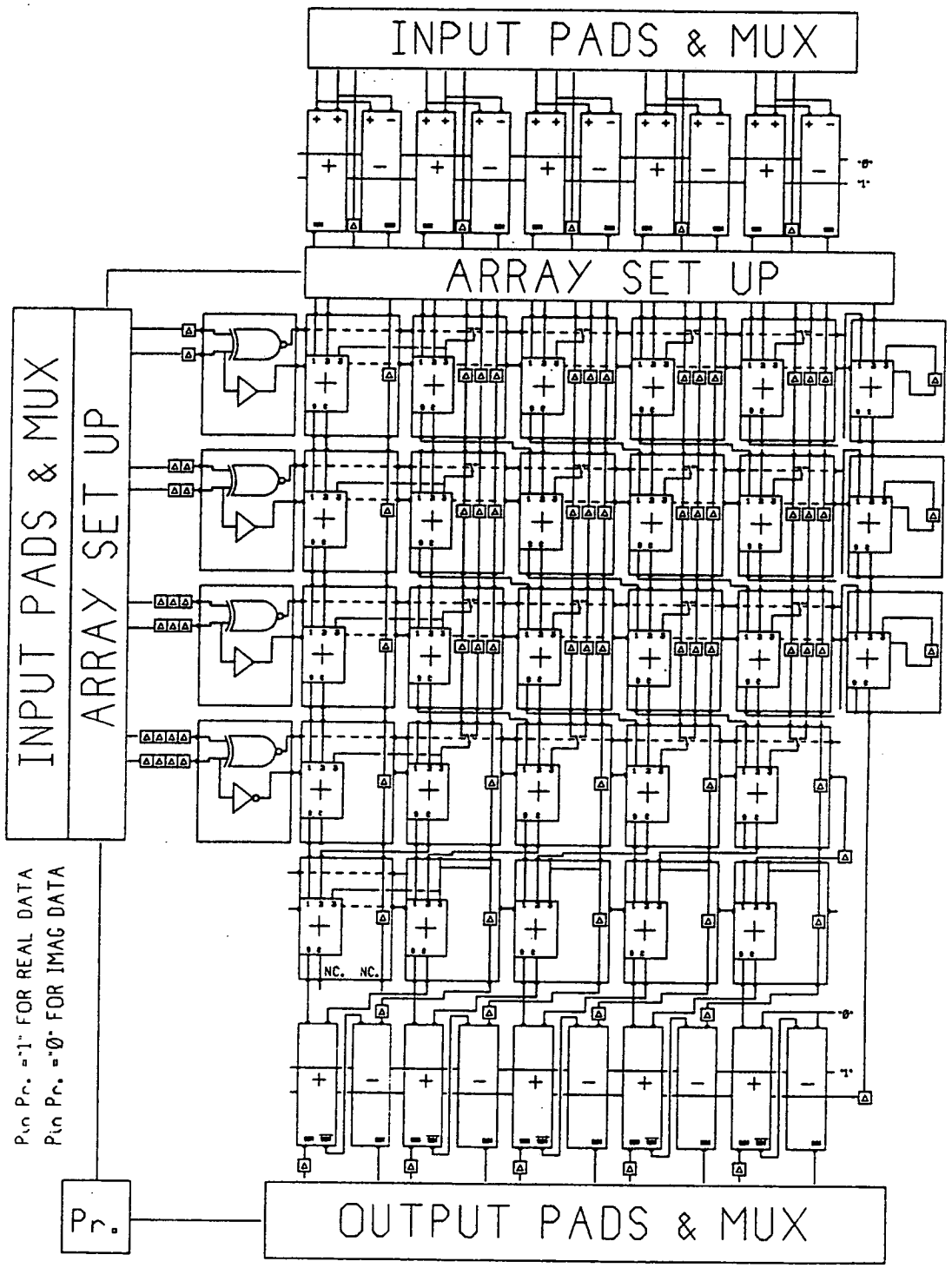
5.3.6. Formation of Butterfly Outputs

Depending on whether real or imaginary data was being computed, either Re(A) or Im(A) would be selected at the input port of EU219 and this would be shifted down through the distributed arithmetic complex multiply array to allow subsequent addition of this with either Re(BW) or Im(BW). This addition was accomplished using carry deferal as in the distributed arithmetic array. Thus Re or Im(A+BW) in unassimilated sums and carries were presented at the output. This was then assimilated using a fast manchester carry adder as previously described to form a complete word. The formation of Re or Im(A-BW) presented a problem because this would have meant feeding BW as well as A through the previous arithmetic stage. This would in practice have required that this stage be widened to allow these signals to pass through. It was therefore decided to form A-BW without feeding BW forward. This was done by shifting A one place left to form 2A. The subtraction of the already formed (A+BW) from 2A would produce the required (A-BW). Thus, (A-BW) was formed as 2A-(A+BW). It was checked that this approach would not cause overflow to occur in the datapath. A general floorplan of EU219 is shown in Figure (5.3.6.1) and a more detailed version is shown in Figure (5.3.6.2).

5.4. Digital and Analogue Testing of EU219



Figure(5.3.6.1.) General Floorplan of EU219



Figure(5.3.6.2.) Detailed Structure of EU219

5.4.1. General

Twenty wafers, containing around 100 devices per wafer, were fabricated by the Edinburgh Microfabrication Facility. The devices were all given an initial probe test to evaluate peripheral circuits and to give the distributed arithmetic array a sparse testing, intended to eliminate most of the eventual rejects. Around 5% of the devices passed this probe test and were bonded up. The bonded devices were then given a speed check (Analogue Performance) and a comprehensive digital check on a Tektronix DAS_9100 series logic analyser.

5.4.2. Probe Testing

The probe test was primitive, and involved the input of a fixed binary word to the data and coefficient ports, and changing the logic on a single pin of the chip which determines whiether real or imaginary data is to be produced. The value chosen was 1/(SQRT(2)) as this would produce a unity magnitude 45 degree vector at all input ports. Thus the 45 degree coefficient would rotate a 45 degree data word to produce a 90 degree result from the complex multiplier. The expected results of this test are shown in Table 1.

Inputs	Outputs			
Re/Im {A,B,W}	Re{A+BW}	Im{A+BW}	Re{A-BW}	Im{A-BW}
01011010	00101101	01101100	00101101	11101101

These precise outputs were obtained during the test in about 5% of the devices. In addition to monitoring the output data port of the chip, which involved most of the chip logic, the tri-state enable logic output was also monitored. This represented a small amount of peripheral circuitry and over 90% of the devices produced a correct result at this output pin. The probe test was designed only to eliminate a large number of chips from the bonding process and was not intended in any way to be interpreted as a final test. Indeed, this test does not properly exercise the input de-multiplexer, as the data is held static. The following tests were performed on the 5% of devices that were bonded up after the simple probe test.

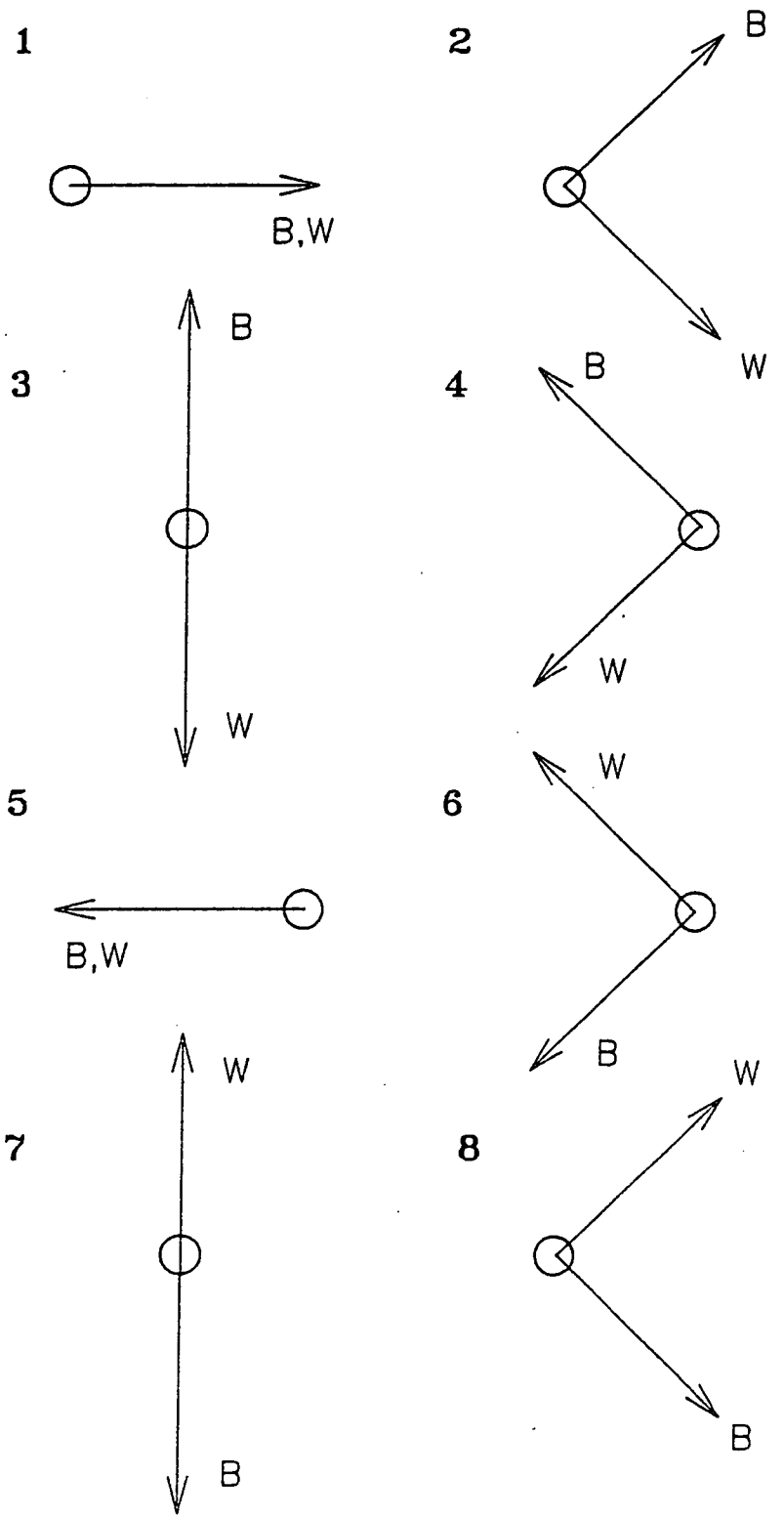
5.4.3. Testing EU219 on a Logic Analyser

A Tektronix DAS_9100 logic analyser permitted the testing of the devices under dynamic conditions, in which data was rapidly changing and the latches were being fully exercised to check for possible poor logic conditions such as might occur due to crosstalk between hard and "soft" nodes, for example, the latter being a feature of dynamic MOS circuits. The initial test involved a check that the two's complement circuitry was operating correctly with both data and coefficient being tried in all four quadrants. The vector rotate circuitry comprises over 90% of the chip transistor count, so it was decided that the next step should be to attempt to exercise this part of the device. It was decided therefore to multiply a unity magnitude data vector which was rotating anti-clockwise with a unity magnitude coefficient vector, rotating

clockwise at the same rate. This is the data that a complex multiplier in a DFT might be subjected to, for example, and TEST1 to be described represents the complex multiplier computing the results of an 8 point DFT, with the "time domain" in coherence with the rotating coefficient vectors. If the chip was operating correctly then it should be observed that the resulting vector (BW) was made fully stationary. An additional feature of this test is that it would allow the examination of any noise that might appear in a system based on the chip due to possible fluctuations in the LSB.

5.4.4. TEST1 for EU219 (Complex Multiplier Only)

In the specific test, the rotating vectors must both start from some arbitrary point on the axis. It was decided to start the rotations from a zero degree angle, and rotate in 45 degree increments. This made calculation of the real and imaginary components fairly trivial. Figure (5.4.4.1) shows the 8 different vectors that were used for "B" and "W". In the initial test, "A" was set to zero as this has nothing to do with testing the complex multiplier part of the chip. After completing this test "A" was set to a unity magnitude, zero angle vector (Real part = hex 7F), and TEST1 was re-run, to give the modified results shown in TEST2, which also exercises the input de-multiplexer more fully and the final butterfly output stages. TEST2 also serves to check for arithmetic overflows, by presenting input signals which should produce the largest output possible from the chip.



Figure(5.4.4.1.) Each of these Vector Multiplications should Result in the same Zero Angled Vector of Unity Magnitude.

5.4.5. TEST1 Results

The results of TEST1 indicated that the chip was operating correctly, with some fluctuations appearing on the LSB of the output. These fluctuations are believed to be normal and primarily a result of arithmetic rounding noise plus quantisation errors in representing the magnitude and phase of the input data vectors. The precise program and data used in TEST1 is shown in Figure (5.4.5.1), and the actual logic output obtained is shown in Figure (5.4.5.2).

5.4.6. TEST2 Results

This test was the same as TEST1, except that Real{A} was set to unity with Imag{A} set to zero as before. This corresponds to a unity magnitude, zero angle vector. The program and data used in this test are shown in Figure (5.4.6.1), with the test results shown in Figure (5.4.6.2). The results obtained were correct, with normal fluctuations in the LSB only, as in TEST1. This test was designed to check for overflow as well as providing another test vector for the output fast adders.

Subsequent tests were then performed to specifically look for possible interactions between data in adjacent time slots, such as might occur in race situations for example. No such situations were observed. Figure (5.4.6.3) shows the results of one of the tests used to evaluate the pipeline for these hazards.

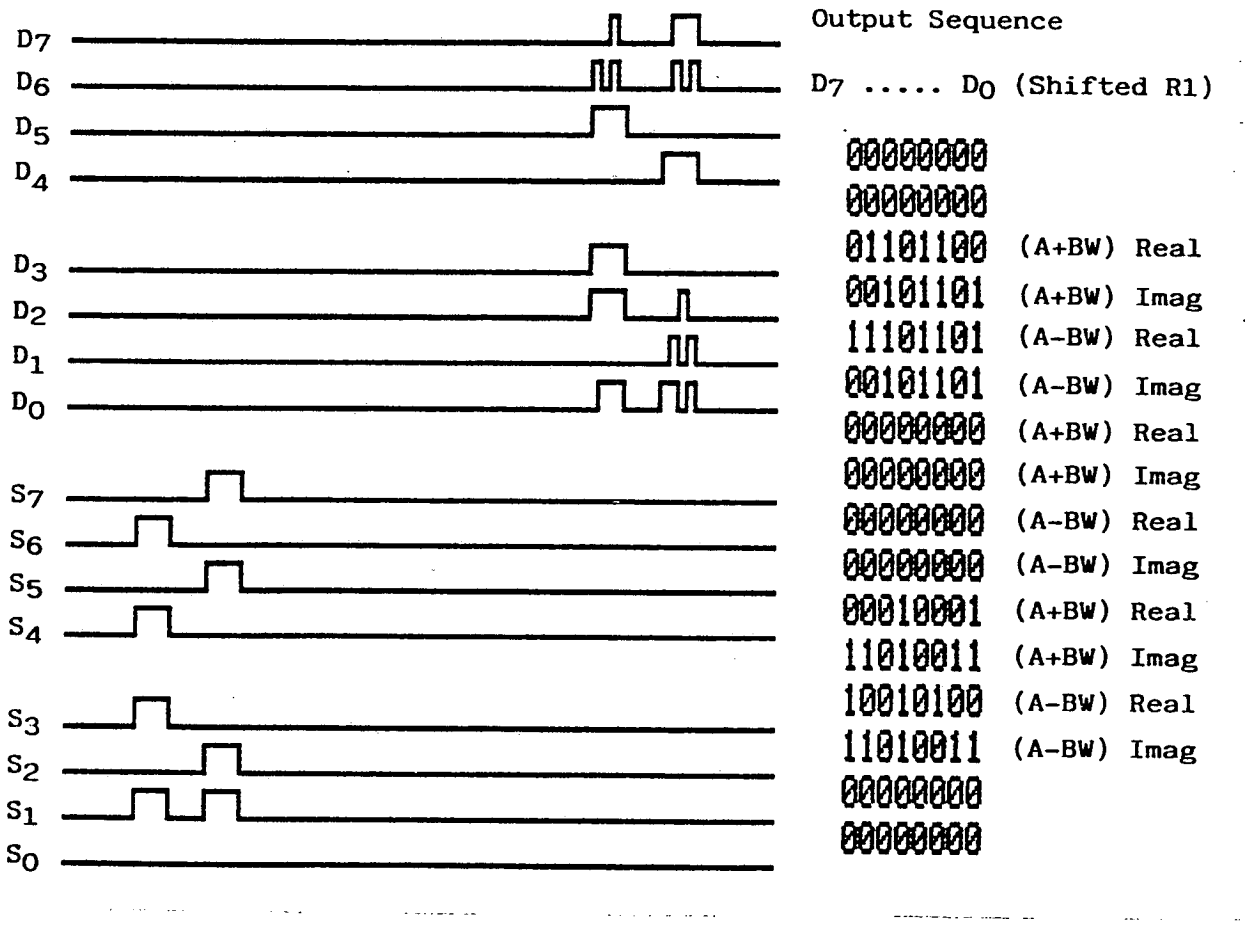


Figure (5.4.6.3.) Dynamic Operation of EU219 Pipeline at 4M Bytes/Second Data Rate showing device Latency.

TEST 1

PATTERN GENERATOR: PROGRAM INTERRUPT: CALL ON 1

CLOCK: 1uS MOS +1.40V PAUSE ON: 1 INHIBIT ON: 1

SEQ	LABEL	P0D4DC HEX	P0D4BA HEX	P0D1CB BIN	INSTRUCTIONS		STROBES
0		0000	0000	1111100000000000	REPEAT	20	
1	SRT	0000	0000	1000100001111111			
2		0000	0000	0000100001111111			
3		0000	0000	0100100000000000			
4		0000	0000	0000100000000000			
5		0000	7F00	0010100000000000			
6		0000	7F00	0000100000000000			
7		0000	0000	0001100000000000			
8		0000	0000	0000100000000000			
9		0000	0000	1000100001011010			
10		0000	0000	0000100001011010			
11		0000	0000	0100100010100110			
12		0000	0000	0000100010100110			
13		0000	5A00	0010100000000000			
14		0000	5A00	0000100000000000			
15		0000	5A00	0001100000000000			
16		0000	5A00	0000100000000000			
17		0000	0000	1000100000000000			
18		0000	0000	0000100000000000			
19		0000	0000	0100100010000001			
20		0000	0000	0000100010000001			
21		0000	0000	0010100000000000			
22		0000	0000	0000100000000000			
23		0000	7F00	0001100000000000			
24		0000	7F00	0000100000000000			
25		0000	0000	1000100010100110			
26		0000	0000	0000100010100110			
27		0000	0000	0100100010100110			
28		0000	0000	0000100010100110			
29		0000	A600	0010100000000000			
30		0000	A600	0000100000000000			
31		0000	5A00	0001100000000000			
32		0000	5A00	0000100000000000			

DATA PORT (8) | CLOCKS (4) | COEFFICIENT PORT(w) (8)

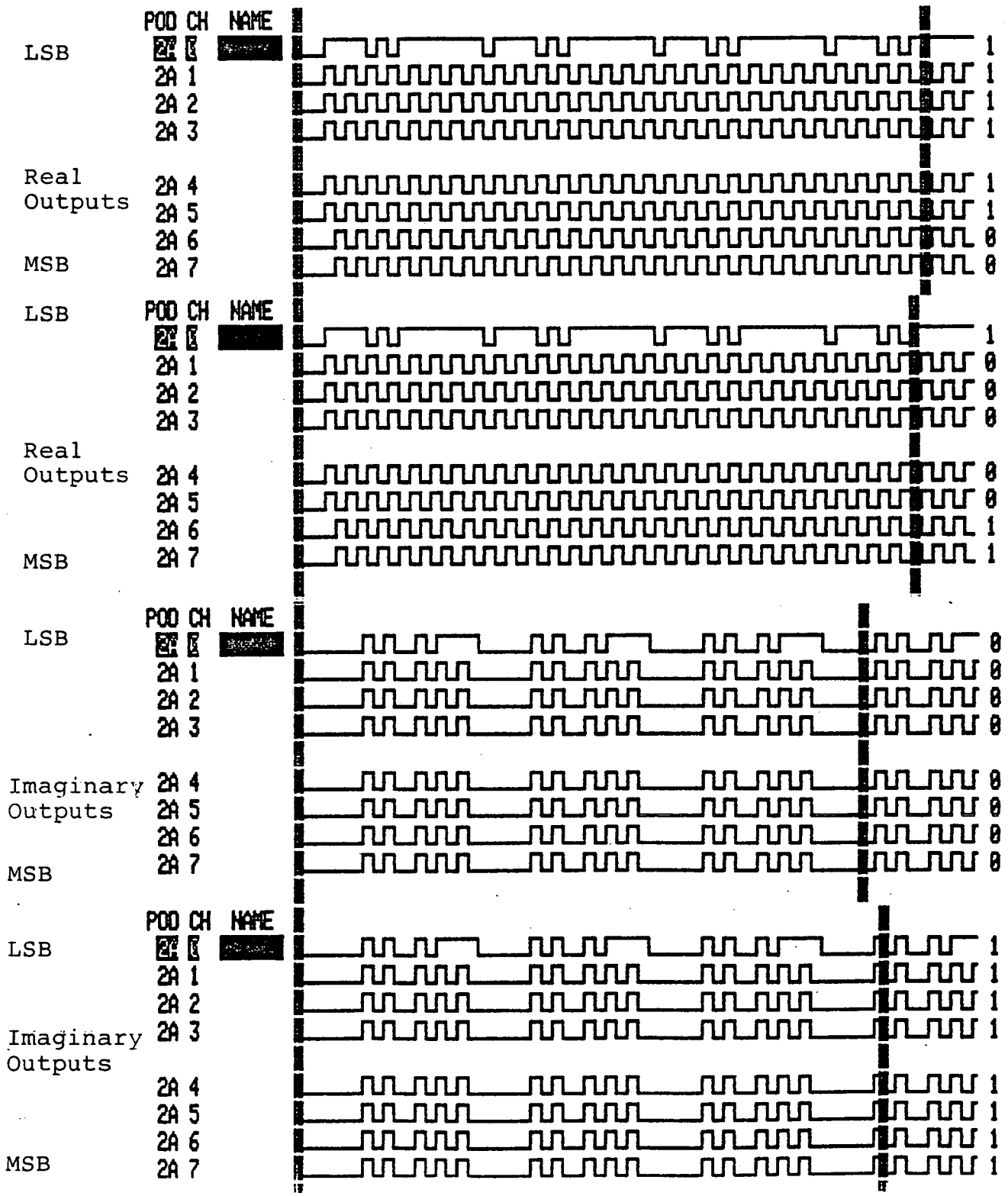
Figure(5.4.5.1.)

33	0000	0000	1000100010000001		
34	0000	0000	0000100010000001		
35	0000	0000	0100100000000000		
36	0000	0000	0000100000000000		
37	0000	8100	0010100000000000		
38	0000	8100	0000100000000000		
39	0000	0000	0001100000000000		
40	0000	0000	0000100000000000		
41	0000	0000	1000100010100110		
42	0000	0000	0000100010100110		
43	0000	0000	0100100001011010		
44	0000	0000	0000100001011010		
45	0000	A600	0010100000000000		
46	0000	A600	0000100000000000		
47	0000	A600	0001100000000000		
48	0000	A600	0000100000000000		
49	0000	0000	1000100000000000		
50	0000	0000	0000100000000000		
51	0000	0000	0100100001111111		
52	0000	0000	0000100001111111		
53	0000	0000	0010100000000000		
54	0000	0000	0000100000000000		
55	0000	8100	0001100000000000		
56	0000	8100	0000100000000000		
57	0000	0000	1000100001011010		
58	0000	0000	0000100001011010		
59	0000	0000	0100100001011010		
60	0000	0000	0000100001011010		
61	0000	5A00	0010100000000000		
62	0000	5A00	0000100000000000		
63	0000	A600	0001100000000000		
64	0000	A600	0000100000000000	GOTO	SRT

Figure(5.4.5.1.)

TEST 1

(Results)



Figure(5.4.5.2.) Results obtained from the DAS-9100

TEST 2

PATTERN GENERATOR: PROGRAM INTERRUPT: CALL ON

CLOCK: 1µS MOS +1.40V PAUSE ON: 1 INHIBIT ON: 1

SEQ	LABEL	POD4DC HEX	POD4BA HEX	POD1CB BIN	INSTRUCTIONS	STROBES
0		0000	0000	1111100000000000	REPEAT 20	
1	SRT	0000	7F00	1000100001111111		
2		0000	7F00	0000100001111111		
3		0000	0000	0100100000000000		
4		0000	0000	0000100000000000		
5		0000	7F00	0010100000000000		
6		0000	7F00	0000100000000000		
7		0000	0000	0001100000000000		
8		0000	0000	0000100000000000		
9		0000	7F00	1000100001011010		
10		0000	7F00	0000100001011010		
11		0000	0000	0100100010100110		
12		0000	0000	0000100010100110		
13		0000	5A00	0010100000000000		
14		0000	5A00	0000100000000000		
15		0000	5A00	0001100000000000		
16		0000	5A00	0000100000000000		
17		0000	7F00	1000100000000000		
18		0000	7F00	0000100000000000		
19		0000	0000	0100100010000001		
20		0000	0000	0000100010000001		
21		0000	0000	0010100000000000		
22		0000	0000	0000100000000000		
23		0000	7F00	0001100000000000		
24		0000	7F00	0000100000000000		
25		0000	7F00	1000100010100110		
26		0000	7F00	0000100010100110		
27		0000	0000	0100100010100110		
28		0000	0000	0000100010100110		
29		0000	A600	0010100000000000		
30		0000	A600	0000100000000000		
31		0000	5A00	0001100000000000		
32		0000	5A00	0000100000000000		

DATA PORT (8) — CLOCKS (4) — COEFFICIENT PORT(w) (8)

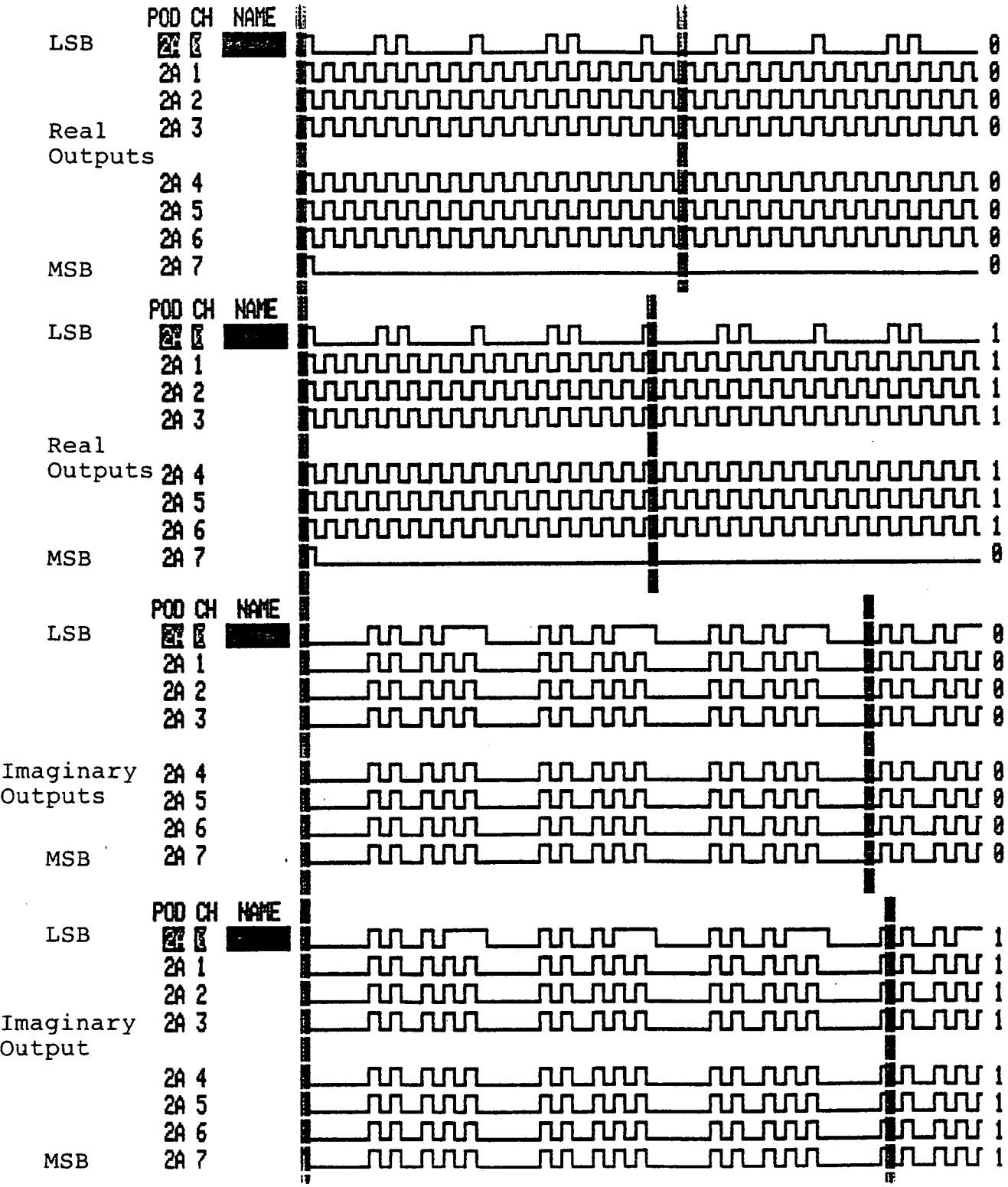
Figure(5.4.6.1.)

33	0000	7F00	1000100010000001		
34	0000	7F00	0000100010000001		
35	0000	0000	0100100000000000		
36	0000	0000	0000100000000000		
37	0000	8100	0010100000000000		
38	0000	8100	0000100000000000		
39	0000	0000	0001100000000000		
40	0000	0000	0000100000000000		
41	0000	7F00	1000100010100110		
42	0000	7F00	0000100010100110		
43	0000	0000	0100100001011010		
44	0000	0000	0000100001011010		
45	0000	A600	0010100000000000		
46	0000	A600	0000100000000000		
47	0000	A600	0001100000000000		
48	0000	A600	0000100000000000		
49	0000	7F00	1000100000000000		
50	0000	7F00	0000100000000000		
51	0000	0000	0100100001111111		
52	0000	0000	0000100001111111		
53	0000	0000	0010100000000000		
54	0000	0000	0000100000000000		
55	0000	8100	0001100000000000		
56	0000	8100	0000100000000000		
57	0000	7F00	1000100001011010		
58	0000	7F00	0000100001011010		
59	0000	0000	0100100001011010		
60	0000	0000	0000100001011010		
61	0000	5A00	0010100000000000		
62	0000	5A00	0000100000000000		
63	0000	A600	0001100000000000		
64	0000	A600	0000100000000000	GOTO	SRT

Figure(5.4.6.1.)

TEST 2

(Results)



Figure(5.4.6.2.) Results obtained from the DAS-9100

5.4.7. TEST3

The first and second tests involved vectors associated with an 8 point DFT. It was decided to double the number of test vectors and apply vectors associated with the W coefficients of a 16 point DFT in order to provide more comprehensive fault detection. This involved the input of angles which are multiples of 22.5 degrees. The data used in this test is shown in Figure (5.4.7.1). Figure (5.4.7.2) shows the results of the test. The correct response was obtained with fluctuations in the LSB evident. This is thought to be the result of rounding errors and is believed to be normal. In this test which was primarily intended to exercise the distributed arithmetic complex multiplier, the non-rotated input data (A) was set to zero.

5.4.8. TEST4

A fourth test was run in which the non-rotated input vector (A) was set to unity (Real {A} =1, Imaginary {A} =0). This test checks for correct carry and sum formation in the final arithmetic stage of the processor and also checks for overflow. This test which is very similar to TEST3 is not shown for this reason. The results of TEST4, however, are shown in Figure (5.4.8.1).

5.4.9. TEST5

It was felt that it would be useful to view the vector outputs from the chip directly on an oscilloscope as an analogue signal. The chip produces digital outputs only, and data is time division

TEST 3

PATTERN GENERATOR: PROGRAM INTERRUPT: CALL [redacted] ON [redacted]

CLOCK: [redacted] MOS [redacted] PAUSE ON: [redacted] INHIBIT ON: [redacted]

SEQ	LABEL	POD4DC HEX	POD4BA HEX	POD1CB BIN	INSTRUCTIONS		STROBES
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	REPEAT	[redacted]	[redacted]
1	SRT	0000	0000	1000100001111111			
2		0000	0000	0000100001111111			
3		0000	0000	0100100000000000			
4		0000	0000	0000100000000000			
5		0000	7F80	0010100000000000			
6		0000	7F80	0000100000000000			
7		0000	0000	0001100000000000			
8		0000	0000	0000100000000000			
9		0000	0000	1000100001110110			
10		0000	0000	0000100001110110			
11		0000	0000	0100100011001111			
12		0000	0000	0000100011001111			
13		0000	7600	0010100000000000			
14		0000	7600	0000100000000000			
15		0000	3100	0001100000000000			
16		0000	3100	0000100000000000			
17		0000	0000	1000100001011010			
18		0000	0000	0000100001011010			
19		0000	0000	0100100010100110			
20		0000	0000	0000100010100110			
21		0000	5A00	0010100000000000			
22		0000	5A00	0000100000000000			
23		0000	5A00	0001100000000000			
24		0000	5A00	0000100000000000			
25		0000	0000	1000100000110001			
26		0000	0000	0000100000110001			
27		0000	0000	0100100010001010			
28		0000	0000	0000100010001010			
29		0000	3100	0010100000000000			
30		0000	3100	0000100000000000			
31		0000	7600	0001100000000000			
32		0000	7600	0000100000000000			

DATA PORT (8) CLOCKS (4) COEFFICIENT PORT(w) (8)

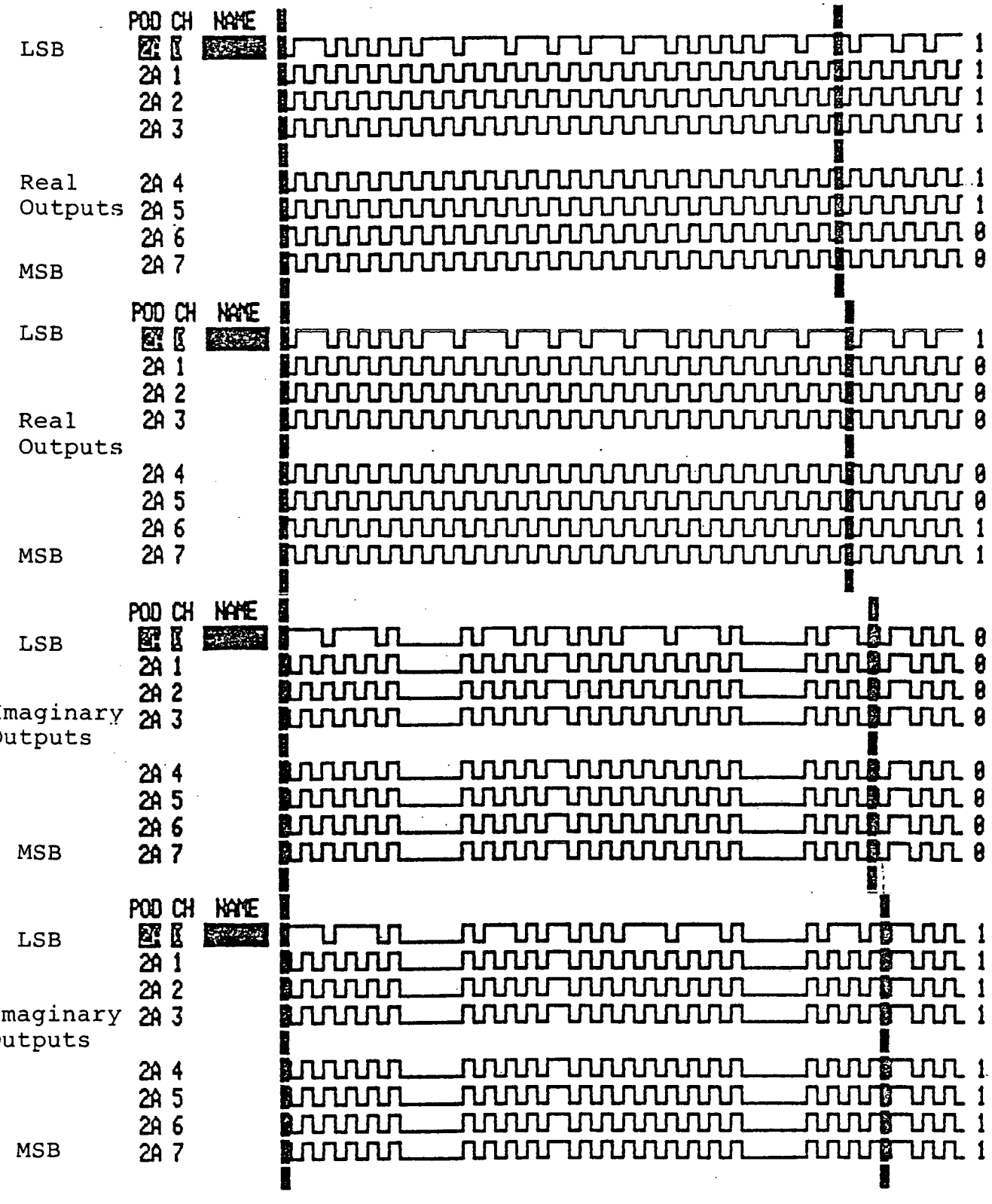
Figure(5.4.7.1.)

33	0000	0000	1000100000000000
34	0000	0000	0000100000000000
35	0000	0000	0100100010000001
36	0000	0000	0000100010000001
37	0000	0000	0010100000000000
38	0000	0000	0000100000000000
39	0000	7F00	0001100000000000
40	0000	7F00	0000100000000000
41	0000	0000	1000100011001111
42	0000	0000	0000100011001111
43	0000	0000	0100100010001010
44	0000	0000	0000100010001010
45	0000	CF00	0010100000000000
46	0000	CF00	0000100000000000
47	0000	7600	0001100000000000
48	0000	7600	0000100000000000
49	0000	0000	1000100010100110
50	0000	0000	0000100010100110
51	0000	0000	0100100010100110
52	0000	0000	0000100010100110
53	0000	A600	0010100000000000
54	0000	A600	0000100000000000
55	0000	5A00	0001100000000000
56	0000	5A00	0000100000000000
57	0000	0000	1000100010001010
58	0000	0000	0000100010001010
59	0000	0000	0100100011001111
60	0000	0000	0000100011001111
61	0000	8A00	0010100000000000
62	0000	8A00	0000100000000000
63	0000	3100	0001100000000000
64	0000	3100	0000100000000000
65	0000	0000	1000100010000001
66	0000	0000	0000100010000001
67	0000	0000	0100100000000000
68	0000	0000	0000100000000000
69	0000	8100	0010100000000000
70	0000	8100	0000100000000000
71	0000	0000	0001100000000000
72	0000	0000	0000100000000000
73	0000	0000	1000100010001010
74	0000	0000	0000100010001010
75	0000	0000	0100100000110001
76	0000	0000	0000100000110001
77	0000	8A00	0010100000000000
78	0000	8A00	0000100000000000
79	0000	CF00	0001100000000000
80	0000	CF00	0000100000000000

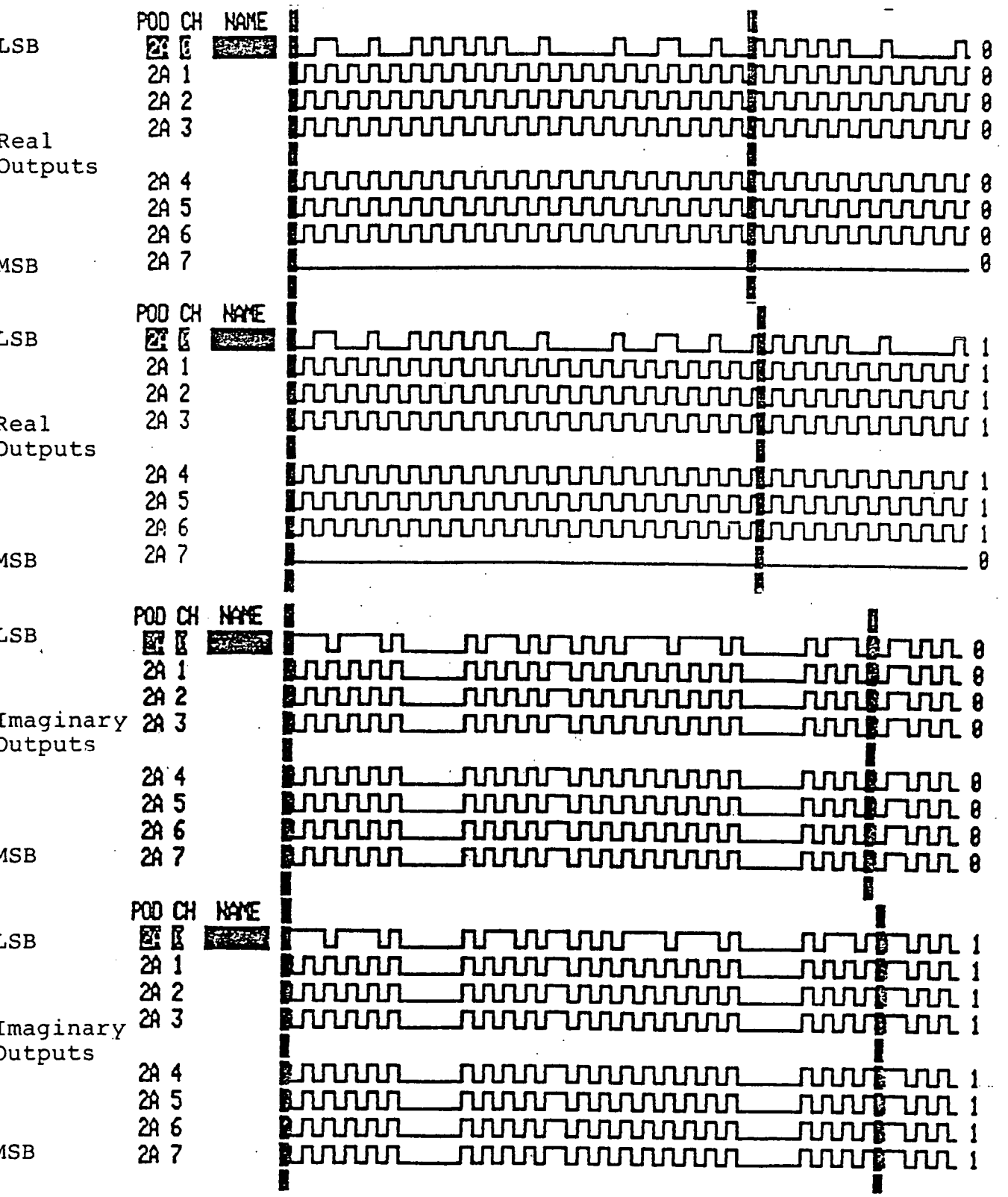
Figure(5.4.7.1.)

81	0000	0000	1000100010100110		
82	0020	0000	0000100010100110		
83	0000	0000	0100100001011010		
84	0000	0000	0000100001011010		
85	0000	A500	0010100000000000		
86	0000	A500	0000100000000000		
87	0000	A500	0001100000000000		
88	0000	A500	0000100000000000		
89	0000	0000	1000100011010000		
90	0000	0000	0000100011010000		
91	0000	0000	0100100001110101		
92	0000	0000	0000100001110101		
93	0000	CF00	0010100000000000		
94	0000	CF00	0000100000000000		
95	0000	8A00	0001100000000000		
96	0000	8A00	0000100000000000		
97	0000	0000	1000100000000000		
98	0000	0000	0000100000000000		
99	0000	0000	0100100001111111		
100	0000	0000	0000100001111111		
101	0000	0000	0010100000000000		
102	0000	0000	0000100000000000		
103	0000	8100	0001100000000000		
104	0000	8100	0000100000000000		
105	0000	0000	1000100000110000		
106	0000	0000	0000100000110000		
107	0000	0000	0100100001110101		
108	0000	0000	0000100001110101		
109	0000	3100	0010100000000000		
110	0000	3100	0000100000000000		
111	0000	8A00	0001100000000000		
112	0000	8A00	0000100000000000		
113	0000	0000	1000100001011010		
114	0000	0000	0000100001011010		
115	0000	0000	0100100001011010		
116	0000	0000	0000100001011010		
117	0000	5A00	0010100000000000		
118	0000	5A00	0000100000000000		
119	0000	A600	0001100000000000		
120	0000	A600	0000100000000000		
121	0000	0000	1000100001110110		
122	0000	0000	0000100001110110		
123	0000	0000	0100100000110001		
124	0000	0000	0000100000110001		
125	0000	7600	0010100000000000		
126	0000	7600	0000100000000000		
127	0000	CF00	0001100000000000		
128	0000	CF00	0000100000000000	GOTO	SRT

Figure(5.4.7.1.)



Figure(5.4.7.2.) Results obtained from the DAS_9100.



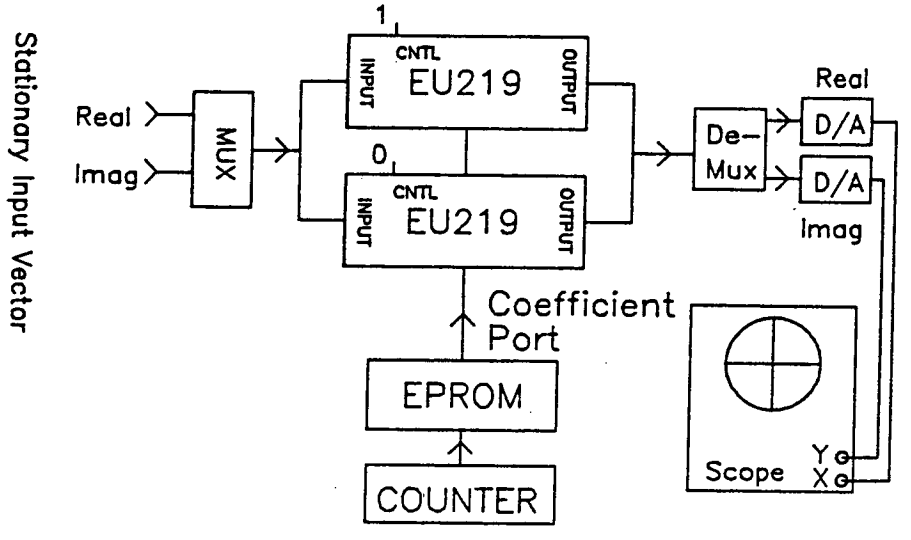
Figure(5.4.8.1.) Results obtained from the DAS_9100.

multiplexed so to do this requires an external de-multiplexer (to separate real and imaginary outputs), two digital to analogue converters (a single fast one could be used if preferred) and a two-channel oscilloscope with facilities for x and y inputs.

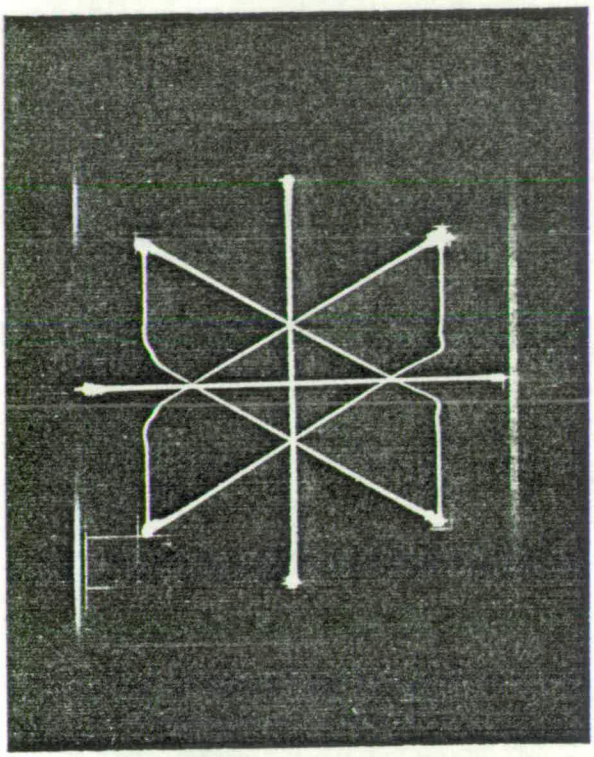
The coefficients (0 - 180 degrees) of a 1024 point FFT were loaded into a 2716 eprom with real and imaginary data being held in even and odd address locations respectively. The eprom could thus be made to output data corresponding to the coefficients of a 1024 point FFT or any size smaller than this which is an integer power of two. The data input to EU219 was supplied with a stationary vector this time so that instead of observing the chip cancelling out two equal and opposite rotations as was done in tests (1-4), it would be possible to observe the chip rotate a stationary vector by the vectors held in eprom. Figure (5.4.9.1) shows the test set up and Figure (5.4.9.2) show some photographs taken from the oscilloscope. The top half (0 - 180 degrees) of the circle represents the (A+BW) (A held constant) butterfly output. The bottom half (180 - 360 degrees) of the circle represents the (A-BW) (A held constant) butterfly output. The results obtained were somewhat encouraging, as there were no observable errors in all 1024 ((8 + 8) bit) vectors that were input to the chip. This test was carried out with several stationary input data vectors and no anomolies were observed.

5.4.10. Analogue Performance of EU219

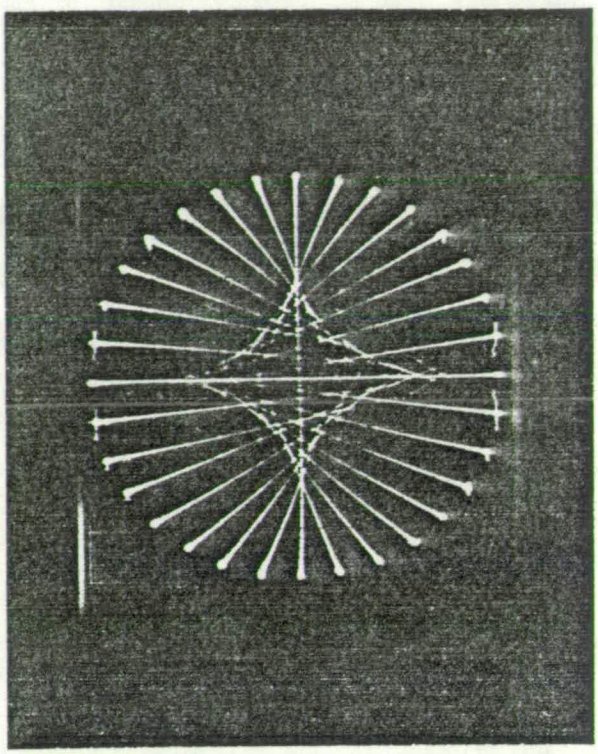
The speed of the device was measured using a high speed clock that was based on high speed Schottky TTL monostables. This was turned up in speed until the chip started to produce logic errors at



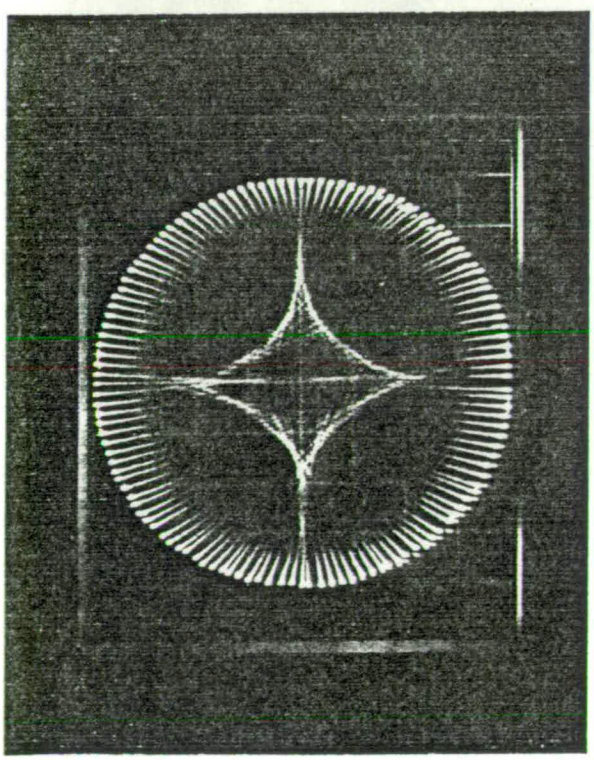
Figure(5.4.9.1.) The use of D/A's to monitor EU219's Output Port on an Oscilloscope.



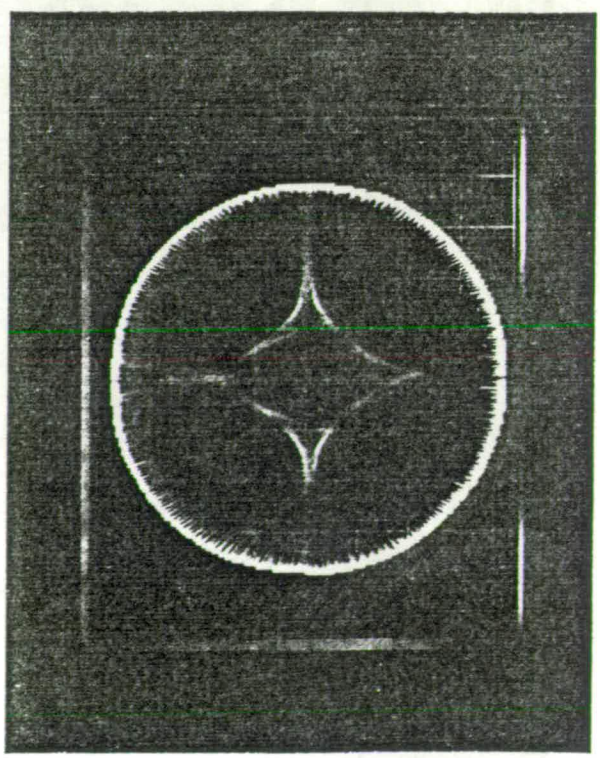
Rotations of $\frac{\pi}{4}$



Rotations of $\frac{\pi}{16}$



Rotations of $\frac{\pi}{64}$



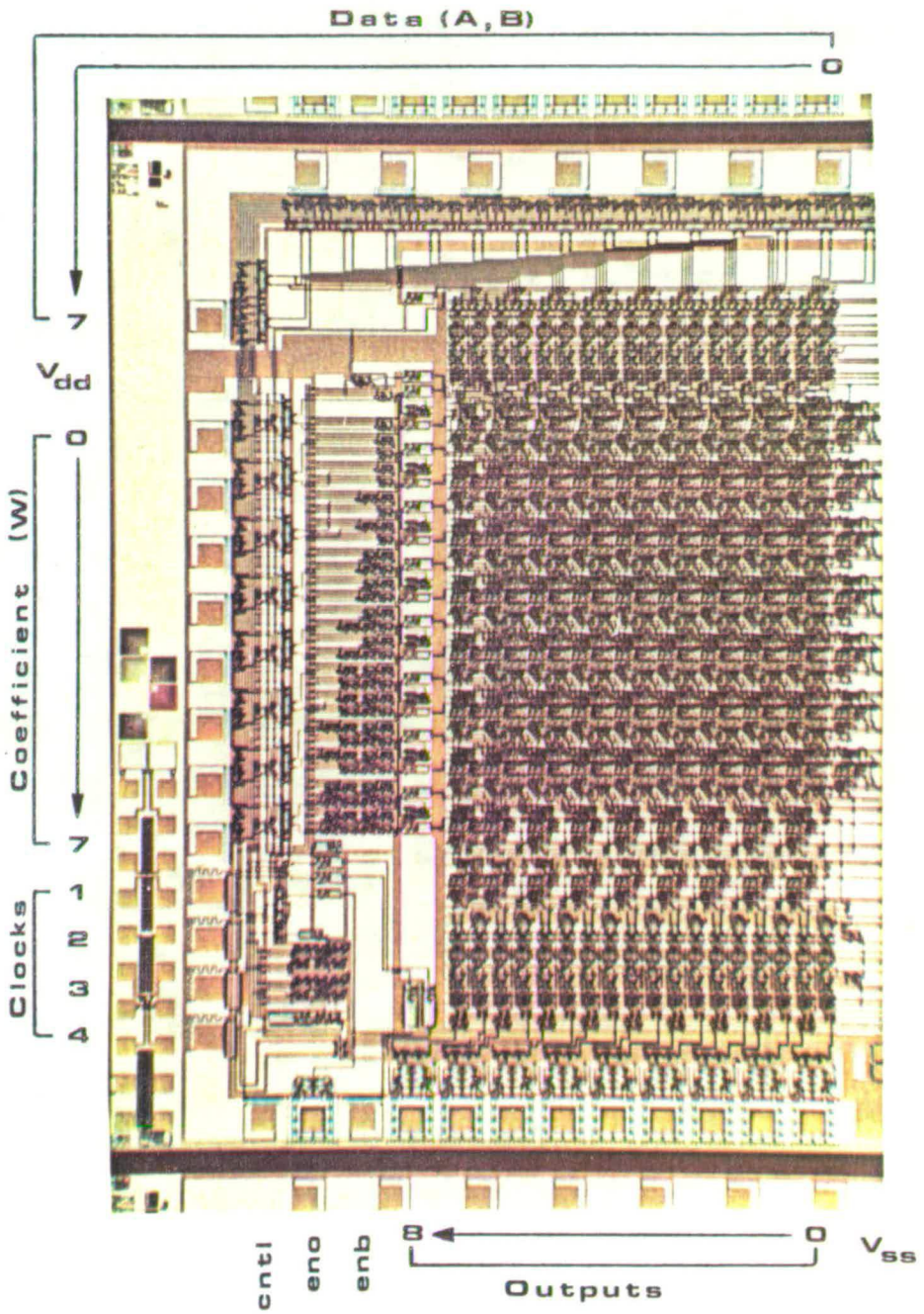
Rotations of $\frac{\pi}{512}$

Figure(5.4.9.2.)

the output on a set of data which was worst case for the fast adder. It was found that the devices clocked at up to 4MHz if the clocks were allowed to rise to 7.0V, however, at 5V, the maximum speed obtained was only 2MHz. This unusually high dependence on the clock voltage was traced to poor clock rise times internal to the chip. Although the device was fast enough for serious use. it was felt that if the clock input pads were modified, a useful increase in the maximum clocking rate should result. Consequently, this modification was made, and a near identical device, number EU341 was fabricated later. Other than the fact that EU341 is slightly faster than EU219, they can otherwise be regarded as the same devices.

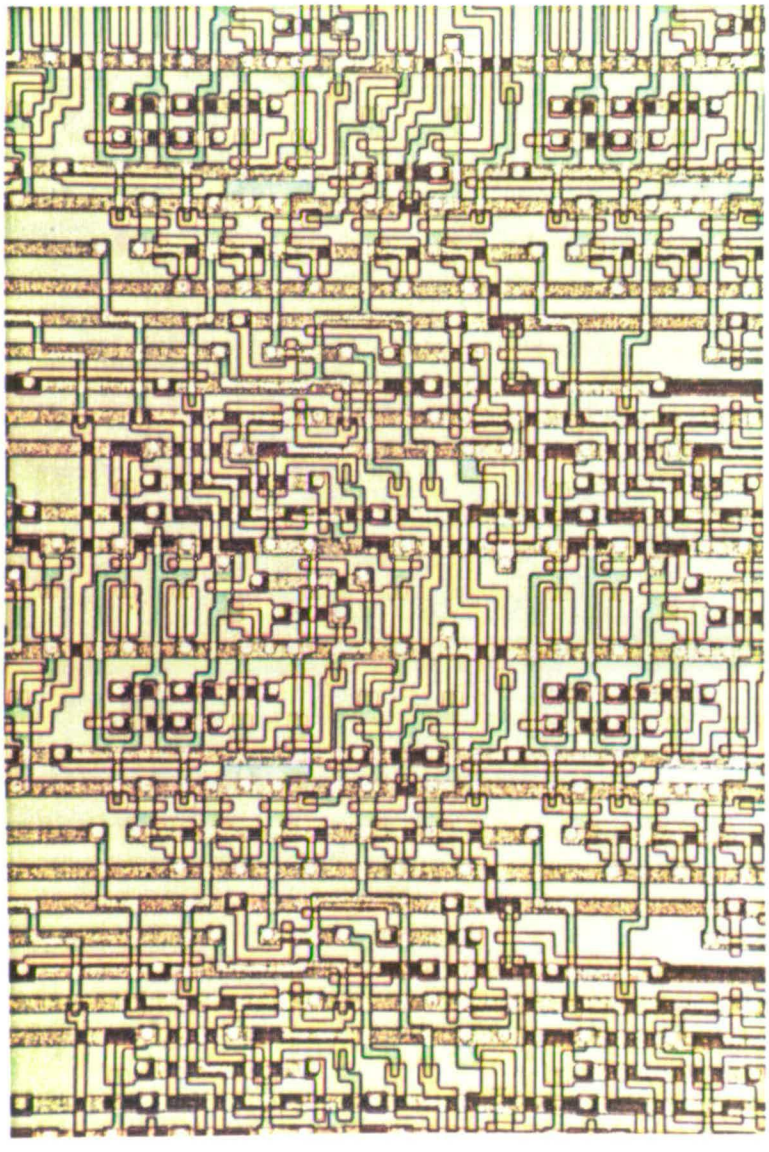
5.4.11. Summary of the EU219 Butterfly Processor

This butterfly arithmetic processor is similar to EU201, but differs in that the chip can be programmed to compute either real or imaginary outputs from the same data. Thus two identical chips can be combined, with the use of tri-state outputs, to function as one completely parallel butterfly device. This partition allows longer wordlengths to be realised and EU219 is an 8 bit device unlike the 6 bits of EU201. Figure (5.4.11.1) shows a chip photograph with pin data information. Figure (5.4.11.2) shows a close up of the distributed arithmetic complex multiplier array showing a number of the basic cells connected together. Finally, Figure (5.4.11.3) shows the pin numbers used when the device is bonded up in a 40 pin dil. package. A useful modification to these designs would be to allow a greater degree of configuration of the datapath to make the device more versatile.

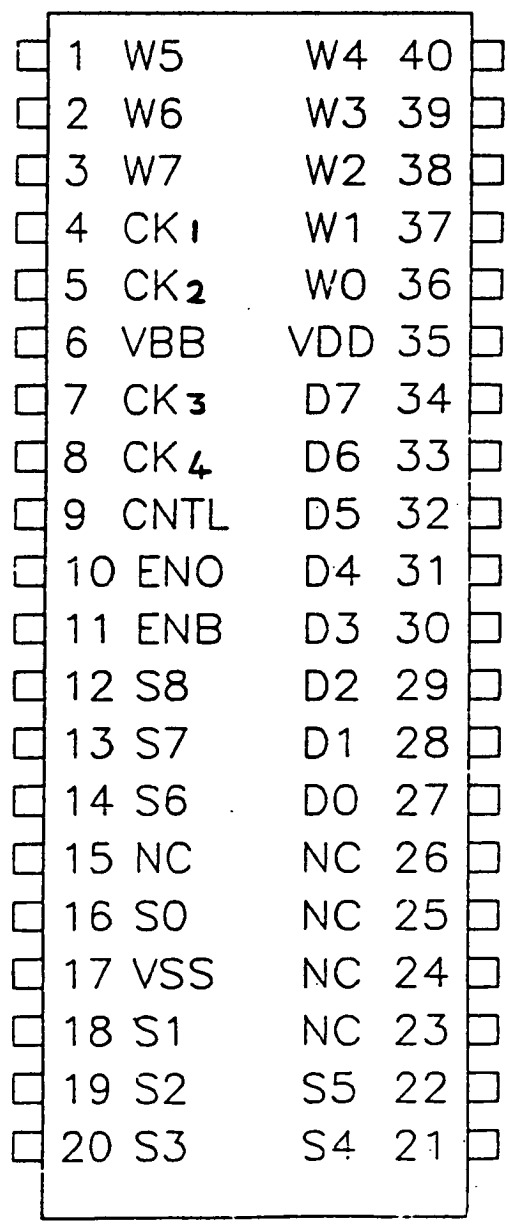


EU219/341

Figure(5.4.11.1.) Photograph and Pad Identification for EU219 (and EU341).



Figure(5.4.11.2.) Close up Photograph of Several Basic Cells in the Complex Multiplier of EU219.



Figure(5.4.11.3.) Pin Identification for EU219 and EU341

The next device to be described is a 16 bit CMOS-SOS datapath. This may be dynamically re-configured under external control and can perform time domain windowing functions in addition to the basic FFT butterfly. It may also be used to compute the DFT at high bandwidths if desired.

5.5. A 16 Bit CMOS-SOS Arithmetic Processor (Z686-SOS)

5.5.1. General

In the previous sections, two similar nMOS devices were described which made use of distributed arithmetic techniques to achieve a highly efficient silicon implementation of the FFT butterfly. These chips, however, do not employ a sufficiently high enough wordlength for general use and were fabricated on a low to medium performance process (6 micron nMOS). Both of these chips employed ripple-through adders internally, thus implying a reduced performance with increased wordlength.

This section considers a 16 bit distributed arithmetic CMOS-SOS datapath chip based on the highly pipelined architecture described earlier in Figure (4.2.8) to allow very high bandwidth computation of the complex multiply at longer wordlengths.

The SOS device is, like both of the previous nMOS devices, aimed primarily at computing the Radix-2 Decimation-in-Time FFT butterfly, however, unlike the previous devices, by passing all data through the complex multiply hardware, it is also possible to perform time domain windowing with no bandwidth penalty. Further, the inclusion of a small control port for datapath control allows (static or dynamic) configuration of the datapath from externally applied control signals thus increasing the versatility of the device.

5.6. CMOS SOS Processor Architecture

All data entering the SOS datapath passes through a hardware complex multiplier which is based on the architecture of Figure (4.2.8). The advantages of this approach are, in addition to the possibility of time domain windowing just mentioned, that when performing the butterfly function, the precise magnitude of the W coefficient vectors are non-critical, since both data inputs (A,B) to the butterfly will be scaled by the same amount. In a practical system this would lead to some improvement (reduction) in butterfly arithmetic noise, particularly as in fixed point fractional 2's complement notation it is not possible to represent unity precisely. (The largest number that can be represented with this notation, is (Unity - 1 LSB) which is 0111111111111111 for N=16 bits.)

In addition, this approach allows the complex multiplier coefficient word to be input directly as KandK* which would be stored in fast ROM, thus lowering still further, the power consumption required to compute the complex multiply.

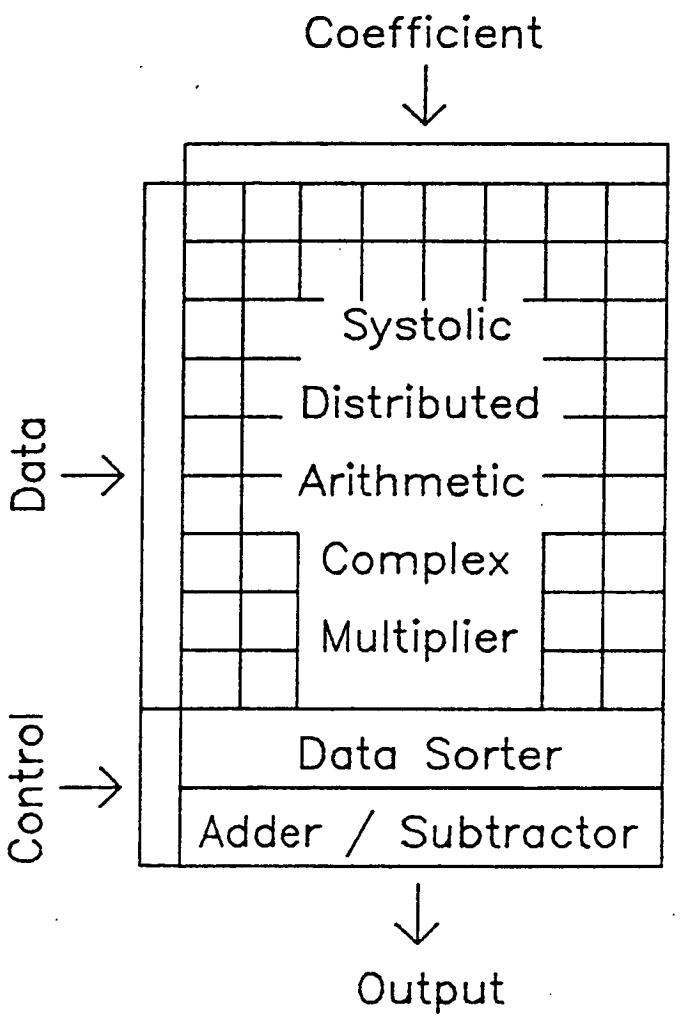
The SOS datapath was designed in three main sections :

1) A distributed arithmetic, systolic array, complex multiplier.

2) A data sorter to sort the data derived from 1) above.

3) An adder/subtractor to accumulate the results from 2) above.

By providing some external control pins to the data-sorter and adder/subtractor the datapath could be dynamically configured to increase its versatility, as mentioned earlier.



Figure(5.6.1.) Floorplan of the CMOS-SOS Arithmetic Processor Z686-SOS.

This lead to a general floorplan, of the type shown in Figure (5.6.1). The above sections will now be described in more detail.

5.6.1. Systolic Array Complex Multiplier using D.A.

Based on the architecture of Figure (4.2.8), the design of this complex multiplier was undertaken with the following requirements :

1/ High Data Throughput

2/ High Arithmetic Precision

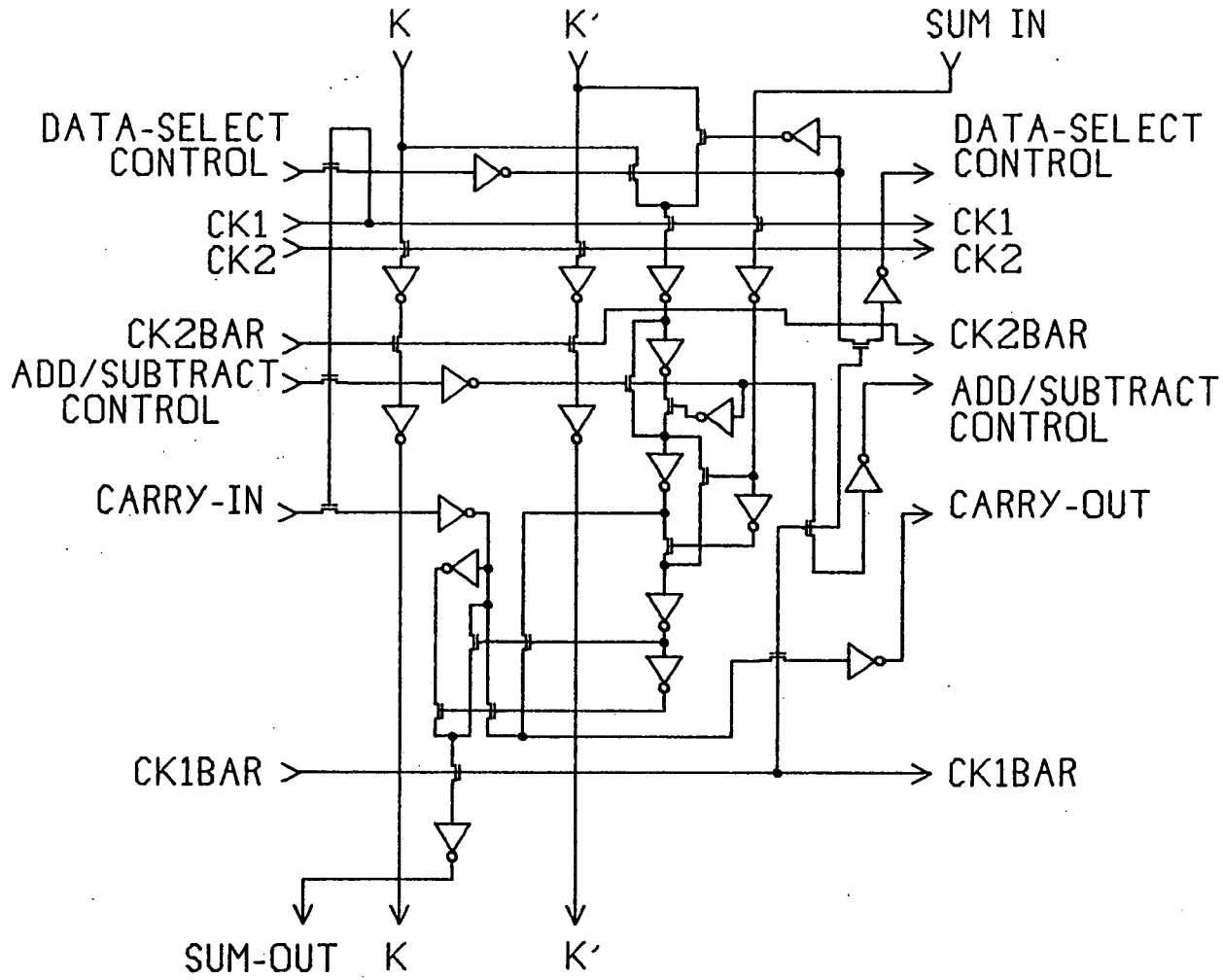
3/ Low Rounding and Arithmetic Noise

4/ Low Power Consumption

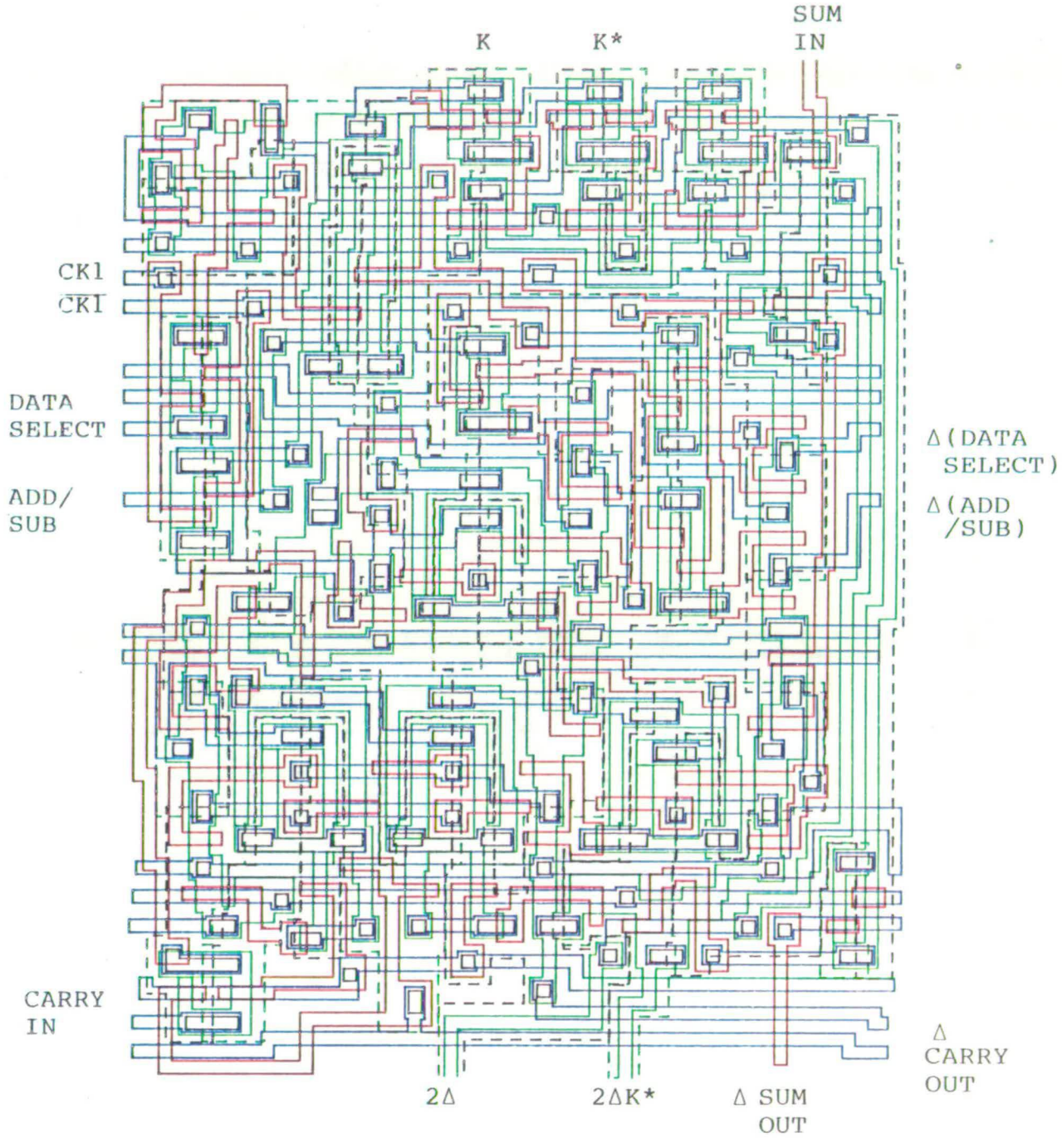
These requirements influenced the design approach in several ways.

5.6.1.1. High Data Throughput

As previously mentioned, high throughputs with large word-lengths can be achieved by using systolic architectures such as Figure (4.2.8) which involves a two-dimensional array of bit-level cells, communicating with their nearest neighbours each clock cycle. The performance of this structure is not highly dependent on word-lengths, with only the latency being determined by the word-length employed. Figure (5.6.1.1.1) shows the logic used in the basic cell of the complex multiplier and Figure (5.6.1.1.2) shows the corresponding SOS layout. In signal processing schemes such as the FFT where there is no essential recursion outside of the arithmetic



Figure(5.6.1.1.1.)



Figure(5.6.1.1.2.) SOS Layout
of Basic Cell

unit, the presence of latency need not affect the system bandwidth.

5.6.1.2. High Arithmetic Precision

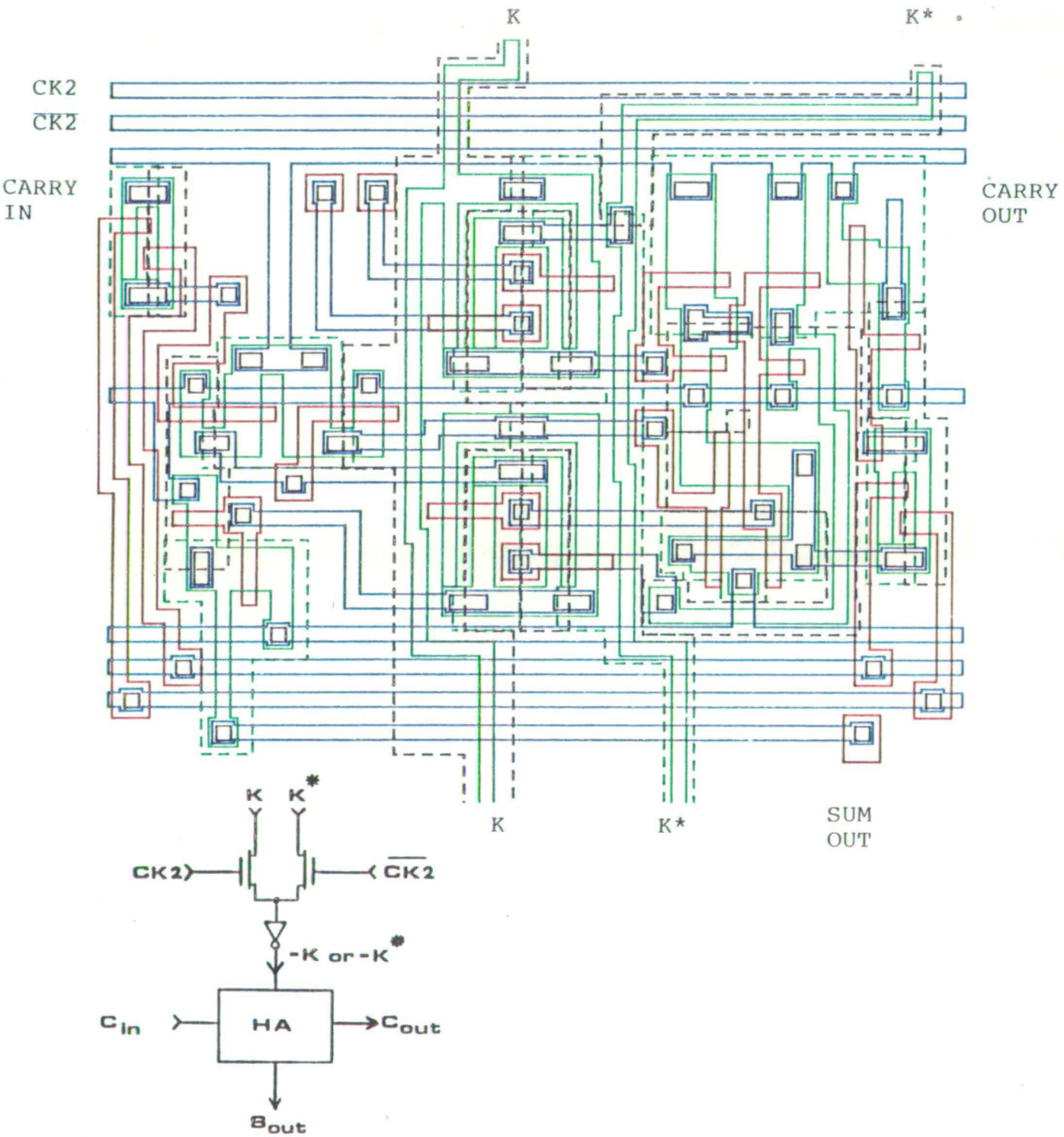
The precision of the butterfly arithmetic unit directly affects the precision of the transform as a whole. Clearly, however, in many systems it is possible that the analogue to digital (A/D) conversion may be the primary limitation, in terms of linearity and accuracy. It is, however, useful to have a reserve of arithmetic precision, beyond that of the A/D, as this allows signal growth in the system to take place, thus avoiding the introduction of rounding noise into the transform. A 16 bit arithmetic precision would allow this mode of operation in conjunction with an 8 bit flash converter, for example, but would also allow meaningful transform results using anything up to a 16 bit A/D converter. It was decided that 16 bits would, in general, be precise enough to cover most real-time digital signal processing applications.

5.6.1.3. Low Rounding and Arithmetic Noise

Intuitively, it would be expected that the DFT or FFT would be rather sensitive to wordlengths used and also to rounding and arithmetic noise, since frequency bins with ideally zero or very small contents may be formed by the cancellation effect of a number of very large vectors at various angles. Poor magnitude or phase resolution at any stage in the computation would therefore result in large percentage errors in frequency bins which ideally should have very small outputs.

One source of arithmetic noise is the noise introduced by truncation of the lowest significance sums at each row in the complex multiplier array where the current partial sum is scaled down 1 place, that is to say, by a numerical factor of 2 (The lowest sum is discarded and the 2nd LSB now becomes the LSB to the next stage). The discarding of these sums introduces a small error which may be called 'ERRTR' into the result. The magnitude of 'ERRTR' depends on the number and significance of logical 1's that were present in the discarded sums. It possible, however, to calculate the range of this error in numerical terms, from zero, to some number which might be called 'MAX' (corresponding to all the truncated sums being a logical '1'). As the result will always be too small by 'ERRTR' the addition of a fixed number, equal to 'MAX'/2, will ensure that instead of a maximum error of - ('MAX') being possible, the maximum error possible would be +/- ('MAX'/2). The precise value of 'MAX'/2 was calculated and later verified by logic simulations. The inclusion of the fixed number 'MAX'/2 was found to be essential in producing a true 16 bit result. It turned out that the value of 'MAX'/2 was equivalent to one-half LSB at the output port, or perhaps more meaningful, equal to the most significant DISCARDED sum.

The distributed arithmetic algorithm also required some initialisation which was a function of data as described in equations (3.5.10 and 11). This was also at a low significance like the rounding word 'MAX'/2. A special cell was therefore constructed to perform both the addition of the rounding word ('MAX'/2) to the array and the data dependent initialisation word. This cell had to perform a half-add function, and is shown in Figure (5.6.1.3.1).



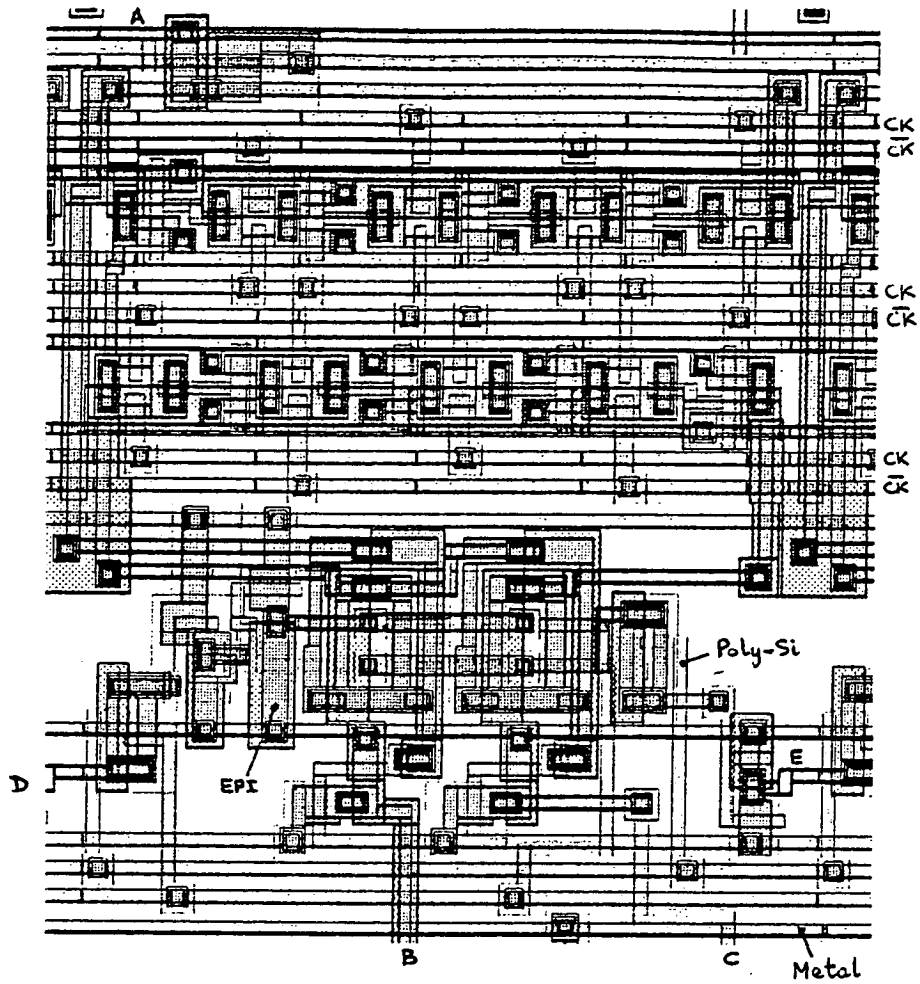
Figure(5.6.1.3.1.) Silicon Layout of Initialisation Cell.

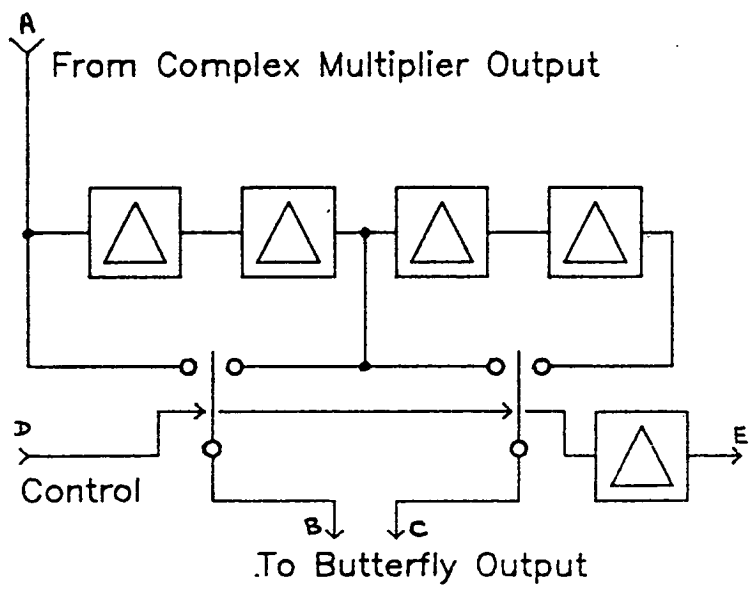
5.6.1.4. Low Power Consumption

Low power consumption is possible with the use of SOS technology, where stray capacitances are lower than in bulk CMOS. The complex multiply distributed arithmetic algorithm also offers low power consumptions due to a high computational efficiency. The storage of the W coefficient in the form of K and K^*, means that they do not need to be actively computed on chip. Thus, an add and subtract that would otherwise be required, need not be computed on chip.

5.6.2. Data Sorter

The complex multiplier, described above, produced a constant stream of alternate real and imaginary data. This data, must first be sorted before being passed to the adder/subtractor butterfly output stage. The circuit chosen to do this was a 4 stage shift register delay line which was tapped at three different points, separated by 2 delays each. Thus, as data flows continuously, either real data only or imaginary data only could be selected and fed to the adder/subtractor to perform the butterfly. A block diagram of the data sorter is shown in Figure (5.6.2.1), along with the actual silicon layout. An important feature of the sorter used was that the data that it had to operate on was skewed, as it was coming from the complex multiplier. This meant that the control for the sorter had to be delayed at each stage, hence the use of an extra shift register to accomplish this delay requirement. Data leaving the sorter is also skewed, allowing the final adder/subtractor stage to operate in the same pipelined configuration as the rest of the chip before finally de-skewing data to present to the outside world.





Figure(5.6.2.1.) Data Sorter Used to separate Real and Imaginary Data from Complex Multiplier.

5.6.3. Final Adder/Subtractor

Having sorted data from the complex multiplier (as defined by external control signals), information is then passed to the final adder/subtractor. This allows the radix-2 butterfly function to be performed, if desired. Control to this unit allows one of the inputs to be reset so that the chip can function as a stand alone complex multiplier. Also, as all data passes through the complex multiplier, the chip can perform a 2-point non-trivial DFT. Although a 2-point DFT in itself is of no use, the fact that the chip is capable of performing the first accumulation operation of the DFT as well as vector rotation, the bandwidth of data leaving the chip is halved, thus allowing a slower external accumulator to complete the job of accumulating each frequency domain bin of the DFT.

In fact, the job of the final adder/subtractor was so similar to the basic cell used in the rest of the chip that this was actually used with only a trivial modification (to allow one of its inputs to be reset).

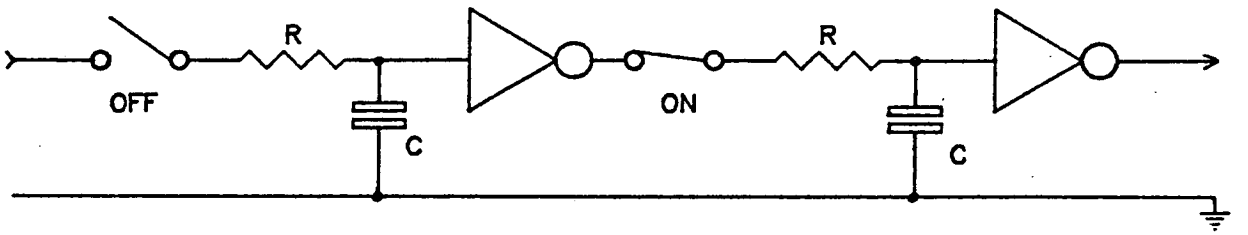
5.6.4. CMOS Design Considerations

In CMOS design, the basic inverter comprises an n-channel pull down and a p-channel pull up so that large currents can only flow under dynamic conditions. Whilst this entails near zero static power consumption, it is possible for a CMOS design to consume a comparable power to nMOS at high clocking rates. The input capacitance of the CMOS inverter is approximately doubled when compared to the nMOS inverter, because both the n channel and p channel device gates are

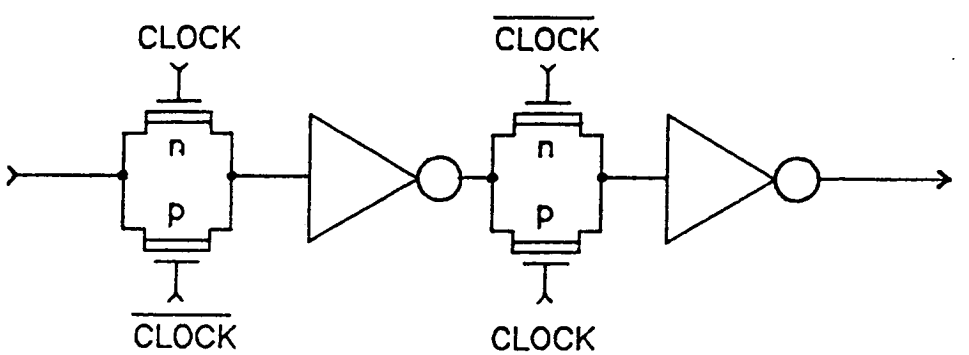
connected together. This usually means that the dynamic power consumption in a given CMOS circuit would rise more steeply with frequency than would an nMOS circuit. One problem that must be tackled in some CMOS designs is one of transients on power lines, which can occur if a number of logic stages, attached to a single power line, all change state at around the same time. This can be overcome by using thick metal conductors in such regions and by arranging that as few stages as possible will change state all at the same time.

Another notable difference between nMOS and CMOS design approaches is in connection with shift register design. In nMOS, it is common to use two phase non-overlapping clocks, as a means of avoiding race condition problems mainly in shift register elements, where the ripple through time is potentially very short. Figure (5.6.4.1) shows a simple but useful model of the dynamic nMOS shift register, which indicates that the race hazard is proportional to 1/RC, where R is the lowest ("ON") resistance of the single (n-channel) transistor transmission gate and C is the effective input capacitance of the inverter. As R and C, both tend to be fairly small in nMOS technology, the use of non-overlapping clocks is mandatory, unless the circuit is carefully designed with a large RC product and highly localised single phase to two phase conversion, to avoid clock skews. In general, however, the use of a two phase non-overlapping clock does not entail any substantial overhead, and is highly preferable to the lengthy analogue simulations, which would otherwise be necessary.

In CMOS design, however, it is fairly common to use clock and clock bar signals which may potentially have some degree of overlap

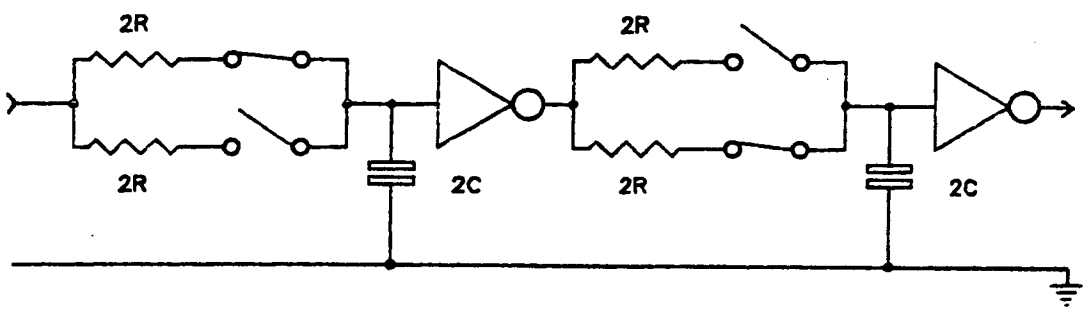


Figure(5.6.4.1.) Simple Model of Dynamic nMOS Shift Register.



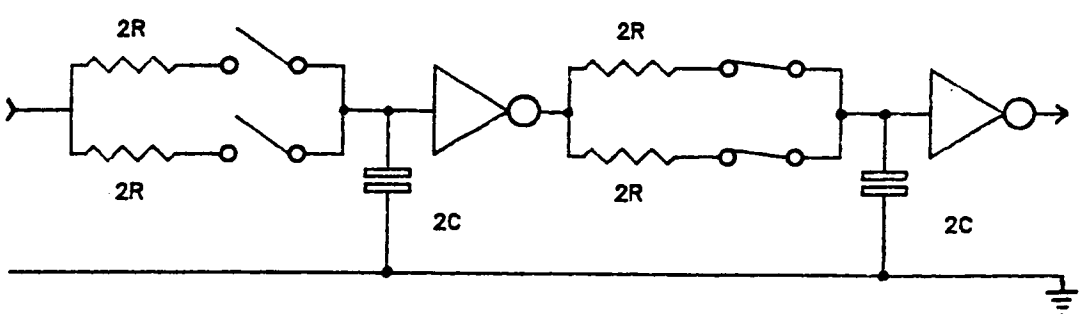
Figure(5.6.4.2.) Simple CMOS Dynamic Shift Register has a Potential Race-Hazard.

present on them. These signals are applied, alternately to the n and p-channel devices in the transmission gates in adjacent shift register cells. The chip to be described was intended to clock at very high data-rates, and consequently, the use of dynamic circuitry was considered. The simplest CMOS dynamic shift register consists of a transmission gate feeding a single inverter with charge storage at its input, as shown in Figure (5.6.4.2). As with nMOS, however, this type of circuit, presents a considerable race hazard as it requires clocks with very low overlap indeed. The amount of overlap that is tolerable, depends on the delay associated with the inverter, and transmission gate. Figure (5.6.4.3) shows the equivalent CMOS model of the dynamic shift register, to the nMOS model shown in Figure (5.6.4.1). It should be noted that some important differences exist in calculating the race hazard. Since, in the CMOS inverter, the input is connected to both gates of the n and p-channel devices, it can said that the input capacitance of the inverter will be at least double the input capacitance of the nMOS inverter. In practice, the p-channel pull-up will be given a wider channel than the n-channel pull-down whose mobility is higher, and thus the real increase in the input capacitance would be slightly greater than doubled. It can be concluded therefore that the CMOS inverter input capacitance is 2C instead of C as is the case with the nMOS device. Turning to the transmission gate, it can be seen that the equivalent circuit of the "ON" transmission gate consists of two parallel resistors, corresponding to the n and p channel devices both being hard ON. Assuming that the designer wishes this CMOS shift register to have a transmission gate of equivalent ON resistance to the nMOS design, then each of these resistors, must be given a value of 2R, to give



Shift Register Model, with Clock Overlapp
(Race Hazard Proportional to 1/4RC)

Figure(5.6.4.3.)



Shift Register Model, with Clocks Fully Settled

Figure(5.6.4.4.)

the transmission gate an "ON" resistance of R. In the special case of clock overlap, then, as shown in Figure (5.6.4.3), only half of the transmission gate is ON when it should be OFF. This means that the resistance to be taken into account in calculating the race hazard is 2R instead of R, (with the nMOS design). The overall race hazard is therefore proportional to 1/4RC. Perhaps, a more accurate comparison would have been to say that since the inverter input capacitance is 2C instead of C, the transmission gate ON resistance should be designed to be (1/2)R instead of R, in order to result in a circuit of the same speed capability. In such a case, the race hazard of the CMOS design would be proportional to 1/2RC compared to 1/RC for the nMOS case. The main point is that it is much easier to design a clock overlap tolerant CMOS dynamic shift register which is race free, than it is, to design an nMOS dynamic shift register with the same attributes. Figure (5.6.4.4) shows the CMOS shift register model with zero clock overlap.

When, the basic cell for the CMOS chip was being designed, it was decided to simulate the dynamic shift register using the SPICE program. This revealed, that the initial aspect ratios used in the shift register were unacceptable, as it was excessively intolerant to non-ideal clocks. The model of the shift register shown in Figure (5.6.4.4) indicated that if the shift register was redesigned with a transmission gate of much higher ON resistance and the inverter input gate capacitance was increased, then the subsequent RC delay introduced, would make the shift register much more tolerant to clock non-ideality. This involved increasing the channel lengths and reducing channel widths of the transmission gates, and increasing

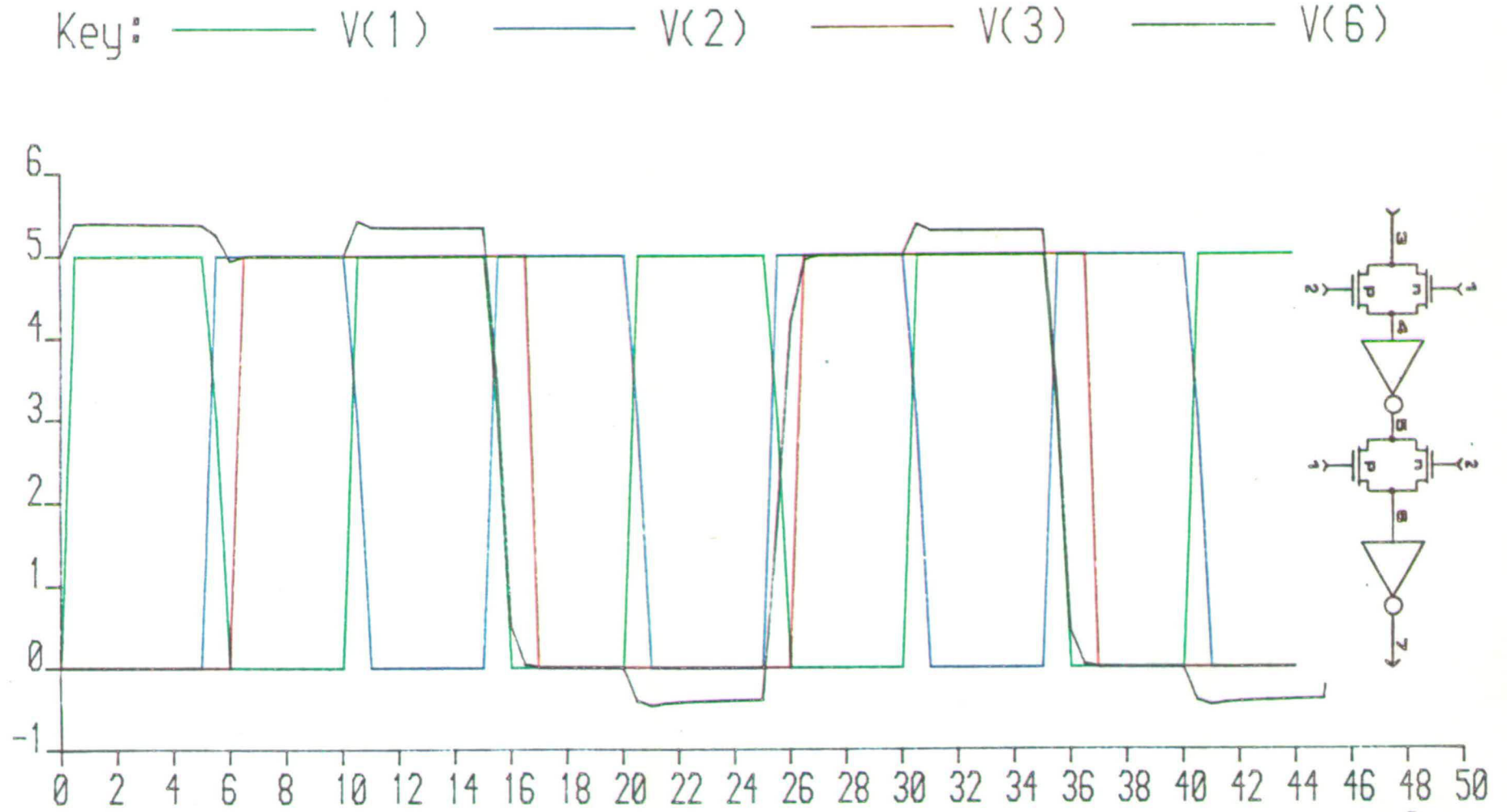
channel lengths of the inverters. Some results of analogue SPICE simulations are presented for this modified shift register design, with clocks ranging from near ideal, to clocks that are worse than would actually be seen by the shift register. These are shown in Figure (5.6.4.5).

The clock lines were distributed throughout the chip using very low resistance paths, to minimise skew and maintain rise-times. In addition, the clocked array of cells were fed from two sides (not one side) of the array, thus halving clock line series resistances caused by underpasses. The chip size was likely to be quite large, and so in common with many large chips, multiple bonding pads were made available for both power and clocks, distributed evenly throughout the chip.

5.6.5. Clocking Scheme

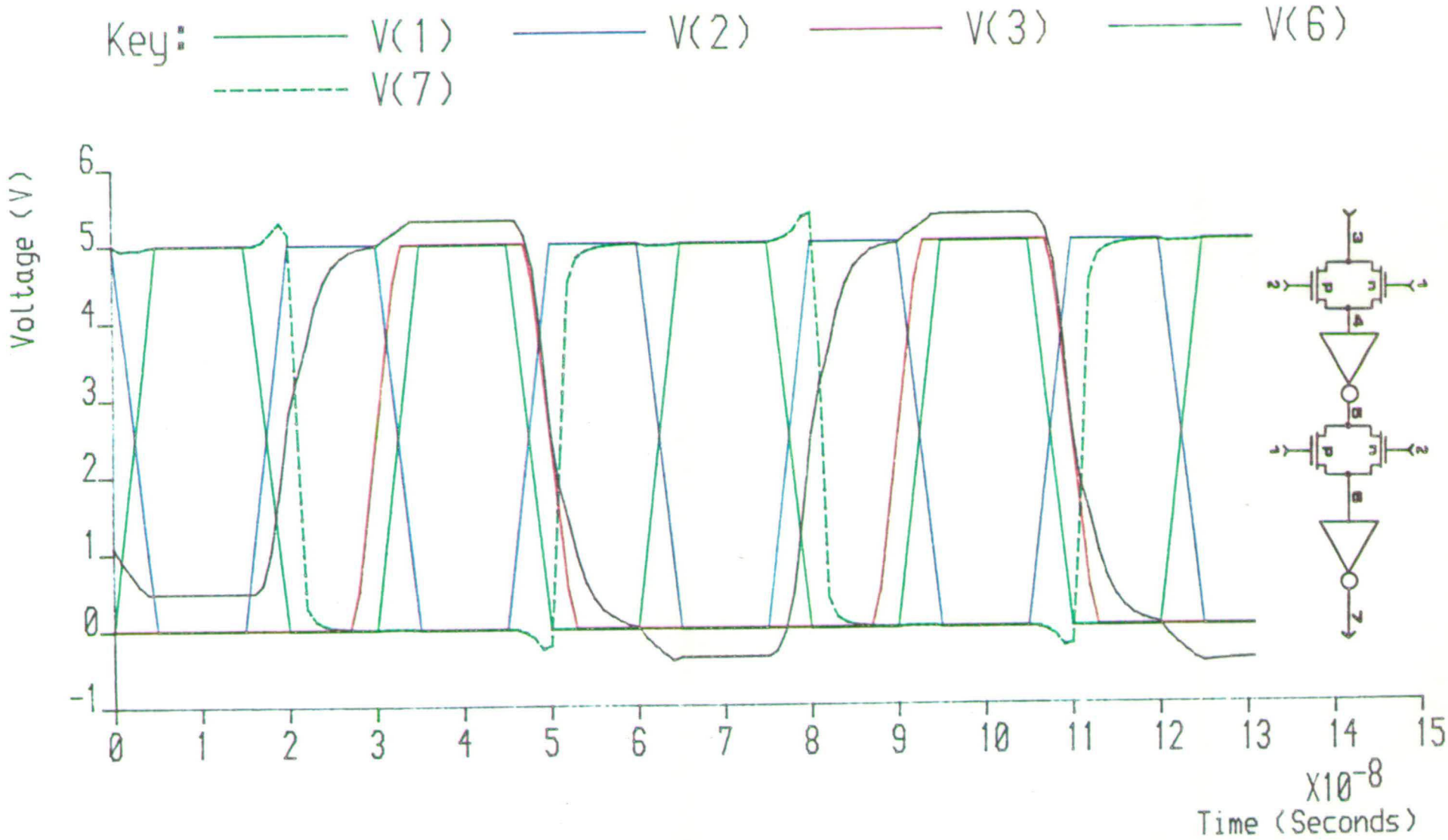
The standard four-phase clocking scheme, involving 'CLOCKIN' and 'CLOCKOUT' together with their complements is the safest and most desirable approach where area considerations are not critical. In this design, however, a substantial area reduction, estimated at around 20%, would result if a two-phase clocking scheme was used involving a single clock (CK1) and its complement. It was this consideration that resulted in the decision to adopt a two phase clocking scheme.

The chip was intended for operating on complex data, with real and imaginary data being multiplexed on the same data port. It was decided that the availability of another main clock (CK2), operating



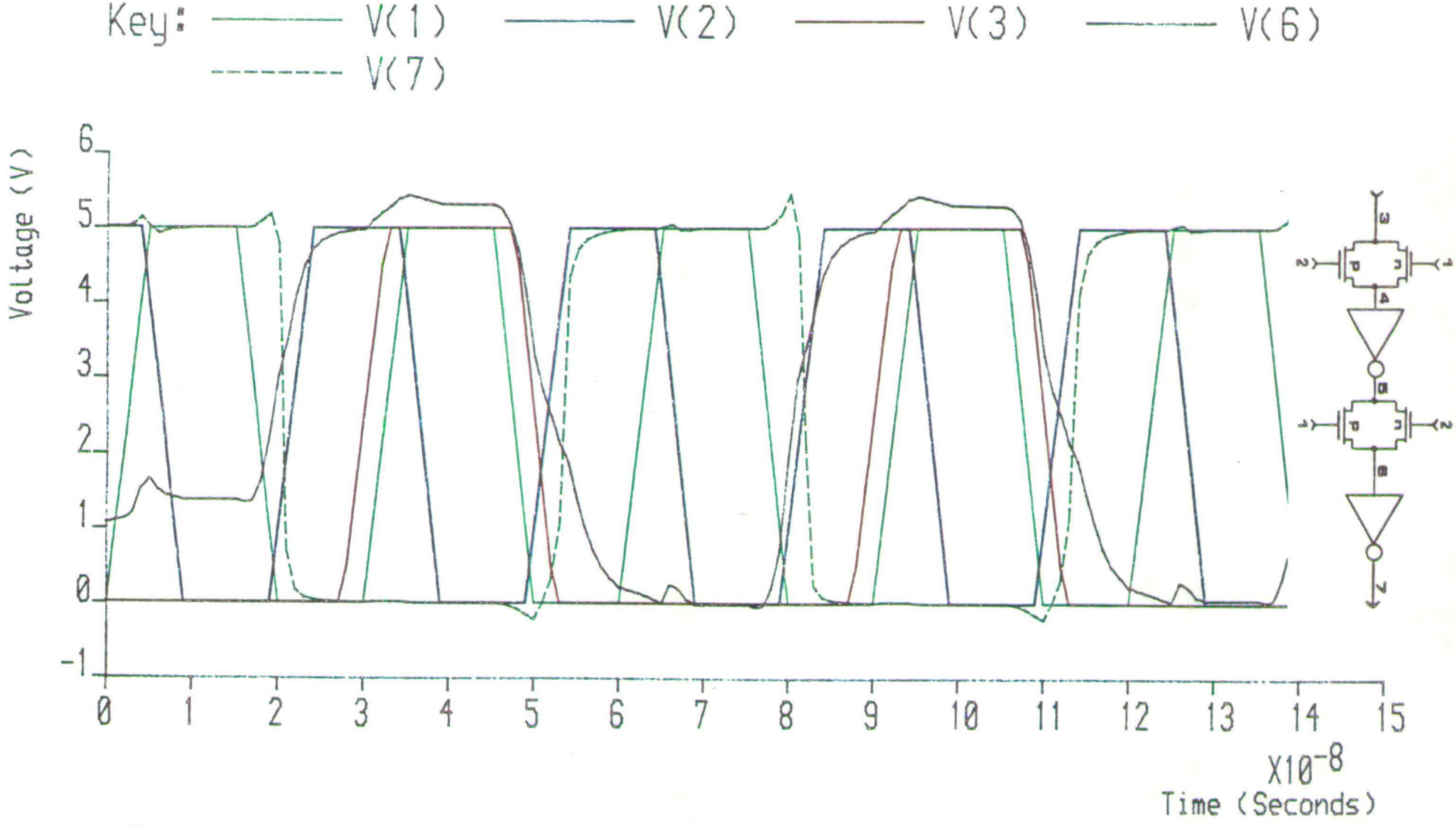
DYNAMIC SOS SR. WITH IDEAL CLOCKS

Figure(5.6.4.5.)



Dynamic SOS Shift Register-5nS Rise, 0nS Skew

Figure(5.6.4.5.)

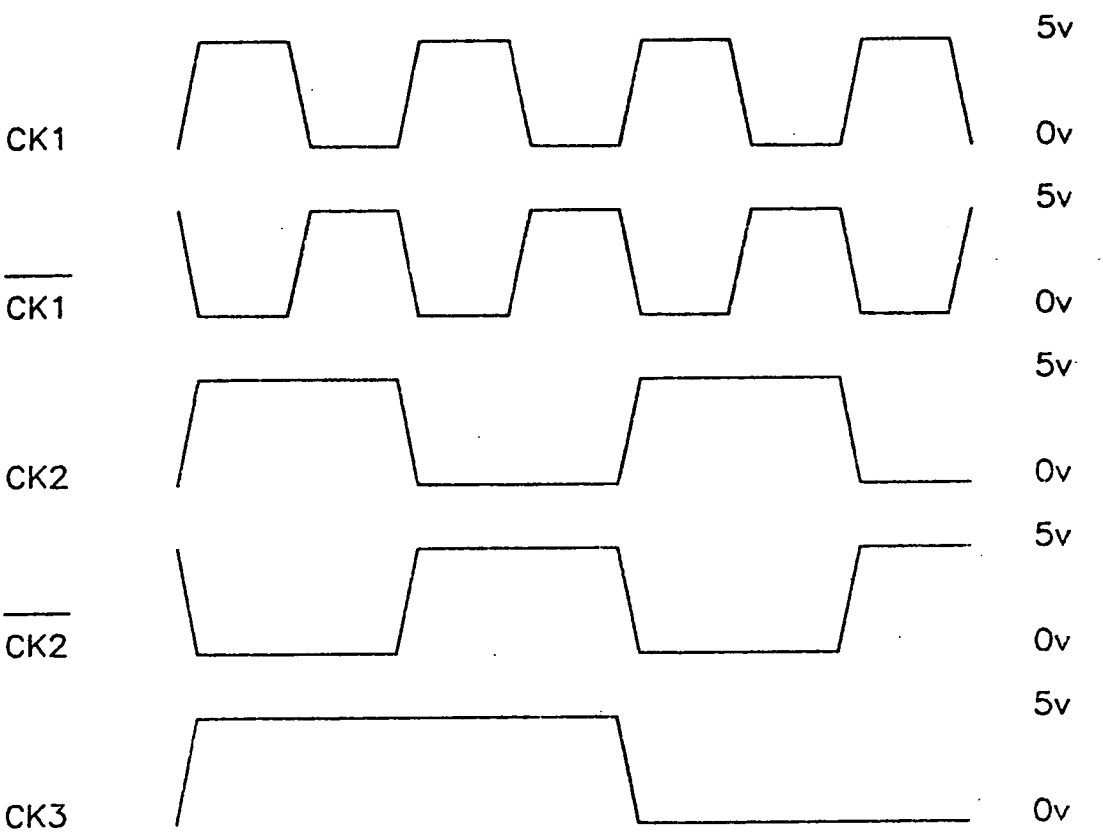

Key: V(1) V(2) V(3) V(6)
V(7)
Voltage (V)
6
5
4
3
2
1
0
-1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
X10-8
Time (Seconds)
Dynamic SOS Shift Register-5nS Rise, 4nS Skew

Figure(5.6.4.5.)

at half the micro-cycle frequency, would be useful in the internal de-multiplexing of real and imaginary data. This clock could also be used as a control in the basic cell to clock the shift registers that carry K and K^* through the array. This is possible because they are both required to compute real and imaginary data at each cell in the array and thus only need to be fed forward once every two cycles and not every cycle, resulting in the chip computing real data and imaginary data on adjacent micro-cycles. This, however, is not a restriction, meaning only that it is not possible to use two chips, one programmed to produce real data, and the other programmed to produce imaginary data, as was the case with the 8-bit nMOS device (as a means of doubling system bandwidth). It is believed to be more efficient and cost effective to perform the computation in this manner where real and imaginary data is computed on alternate cycles, thus reducing power consumptions further. In addition to the half frequency clock, which determines whiether real or imaginary data is being input to the datapath, an extra quarter frequency clock (CK3) is needed for the butterfly itself. This controls the data sorter, and the add/subtract in the butterfly output stage and corresponds to the rate at which the chip can perform windowing and/or butterfly functions. Figure (5.6.5.1) shows the clocks required to operate the chip.

5.6.6. Timing Requirements.

It was decided that clock signals would be derived externally from this chip, as the quality of the clock signals could then be controlled, if necessary. As the chip uses a two-phase clocking

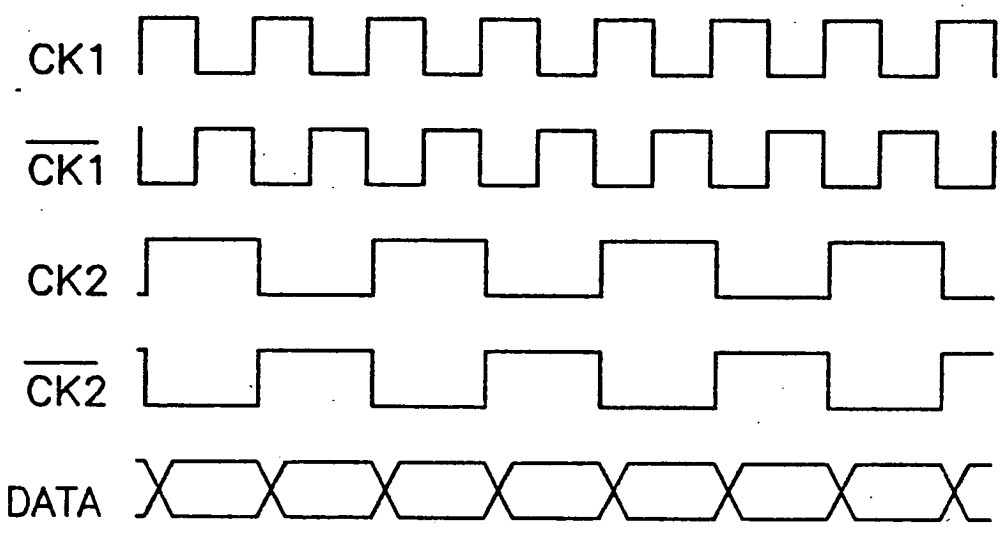


Figure(5.6.5.1.) Clocks Required for Butterfly Operation.

scheme, all clocks, with the exception of CK3 (ie CK1 and 2) must be provided with high integrity complements. If possible, a small (RC) analogue delay should be inserted into the leading transition to ensure that the clocks have a minimum overlap. Rise times should be kept as small as possible, preferably under 5ns, which can achieved by using bipolar devices for pull-up and down. It is intended that the chip should be clocked at or near to its maximum clocking frequency, as the error rate due to background alpha-radiation is minimised in this situation. There are no strict requirements for CK3, which does not require a complemented signal, other than that its falling edge must not come before the rising edge of CK1. A similar requirement exists for CK2 and its complement, whose falling edges must not come before the rising edge of CK1, or put another way, the rising edges of CK2/CK2BAR must come on or after (preferably just immediately after) the rising edge of CK1.

5.6.7. Latching of Input Data and Coefficients

The inputs to the chip are latched on CK1BAR. Thus, data must have settled before the falling edge of CK1BAR. If data is to be fed in to the latch without being corrupted, it must not alter state until after CK1BAR has returned to zero. The precise timing of data transitions relative to CK1 and CK1BAR is non-critical other than this requirement. One simple way of ensuring that this occurs is to synchronise inputs data transitions with CK2/CK2BAR rising and falling edges. It was mentioned that this clock should be very slightly delayed with respect to CK1/CK1BAR to ensure that its rising/falling edges would never come before the rising edge of CK1 (falling edge



Figure(5.6.7.1.) Timing of Data Input and Output relative to CK1 and CK2.

of CK1BAR). A typical timing diagram of the chip receiving data relative to the clocks is shown in Figure (5.6.7.1). This also shows the output of data from the CMOS-SOS processor device. Data transitions at the output takes place on the rising edge of CK1. This is to be consistent with sampling data on the rising edge of CK1BAR as is done at the input ports of the chip.

5.6.8. Control Signals Required by Pipeline

Control data is sampled on the rising edge of CK1BAR as with all other data entering the device. Thus, timing for the input of control data is the same as for signal and coefficient data.

The CMOS-SOS device has a total of seven control pins, CNT1-7, four of which (CNT4-7) are aids to the testing of the device and three of which (CNT1-3) allow the pipeline to compute the functions described below.

If no connections are made to these control pins, in common with all other data pins, a logic zero default will be assumed by the chip.

1) The complex multiply, +/- A x W_1 : CNT1="1" for complex multiply only, CNT2="1" for "+", CNT2="0" for "-", CNT3=(don't care, however, if CNT3 is a "0", then data will appear with extra two CK1 delays but will be the same)

2) A two point DFT, +/- A x W_1 +/- B x W_2 : CNT1="0", CNT2=CNT3=CK3 (shown in Figure (5.6.5.1)). This function allows both butterfly operation (decimation-in-time) and data

windowing. It can also be used to compute the DFT using a slower, external accumulator.

The control signals, CNT4-7 are intended primarily for testing the device. The function of these pins are to control two ROMS, each storing two 16-bit words which may be input to the data port and the coefficient port. Two of the pins, CNT4 and 5 select either the ROM or the input pads for the two input ports. The remaining two pins, CNT6 and 7 select one of two words which have been stored in the small ROM. These words have been chosen to produce predictable results, consistent with exercising the internal logic. This allows a test of a similar level to the probe test of EU219, which was not a final test, but was designed to eliminate obviously faulty devices. This allows a simple probe test to be conducted with only four data pins instead of 32. The functions of CNT4-7 are detailed below.

1) CNT4 : Select Data ROM "1", or input pads "0".

2) CNT5 : Select Coefficient ROM "1", or input pads "0".

3) CNT6 : Select 1010010101111111 "0", or 0101101010000001 "1" at DATA input port.

4) CNT7 : Select 0000000000000000 "0", or 0011111111111111 "1" at COEFFICIENT port.

Note : 0101101010000001 = 1/SQRT(2) = - (1010010101111111)

5.7. Logic Simulation

The chip logic was simulated at a switch level at a depth of 8 bits and later, 4 bits (to speed up the process). This was to verify correct operation of the pipeline, rounding and initialisation of the distributed arithmetic array. A fixed binary count was applied to the data port and a +1 and -1 coefficient vector was applied to the coefficient port. The simulation results, shown in Figure (5.7.1) indicated no observable rounding errors and even weighting for both positive and negative number ranges. The logic simulator used was known as SLS which was designed at Edinburgh University and was run on a VAX750 mini-computer.

5.8. Additional Cell Level Details of the Z686-SOS Processor

The logic used in the control cell for the distributed arithmetic array in the SOS device is shown in Figure (5.8.1). Figure (5.8.2) shows the corresponding SOS layout. This cell produces alternate real and imaginary control data under the control of CK2.

The de-multiplexing of real and imaginary data may be accomplished under control of CK2. Figure (5.8.3) shows the logic used for de-multiplexing both data and coefficient input ports. Real data is sampled on CK2 and imaginary data is sampled on the complement of CK2. The SOS layout for the de-multiplexer is shown in Figure (5.8.4).

The complete chip includes input and output pads, shift register delays and two very small ROMS for testing purposes. Figure (5.8.5) shows the regular shift register layout and Figure (5.8.6)

	Results					initialisation complete								Data			
t	R4	R3	R2	R1	R0	C1T	C1B	C2T	C2B	K_3	K_2	K_1	K_0	D3	D2	D1	D0
0	1	0	0	0	0	1	0	1	0	K	1	1	0	0	0	0	0
2	1	1	0	0	0	1	0	0	1	K*	1	1	0	0	0	0	0
4	0	1	1	0	0	1	0	1	0	K	0	1	0	0	0	0	0
6	1	0	0	1	0	1	0	0	1	K*	0	1	0	0	0	0	0
8	1	1	0	1	1	1	0	1	0	etc	1	1	0	0	0	0	1
10	0	1	1	1	1	1	0	0	1		1	1	0	0	0	0	1
12	1	0	0	1	1	1	0	1	0		0	1	0	0	0	0	1
14	0	0	0	1	1	1	0	0	1		0	1	0	0	0	0	1
16	0	0	1	0	1	1	0	1	0		1	1	0	0	0	1	0
18	0	0	0	1	1	1	0	0	1		1	1	0	0	0	1	0
20	0	0	0	1	1	1	0	1	0		0	1	0	0	0	1	0
22	0	0	0	0	0	1	0	0	1		0	1	0	0	0	1	0
(Start >) 24	0	0	0	0	0	1	0	1	0		1	1	0	0	0	1	1
26	0	0	0	0	0	1	0	0	1		1	1	0	0	0	1	1
28	0	0	0	0	0	1	0	1	0		0	1	0	0	0	1	1
30	0	0	0	0	0	1	0	0	1		0	1	0	0	0	1	1
32	0	0	0	0	0	1	0	1	0		1	1	0	0	0	1	0
34	1	1	1	1	1	1	0	0	1		1	1	0	0	0	1	0
36	1	1	1	1	1	1	0	1	0		0	1	0	0	0	1	0
38	0	0	0	0	1	1	0	0	1		0	1	0	0	0	1	0
40	0	0	0	0	1	1	0	1	0		1	1	0	0	0	1	1
42	1	1	1	1	0	1	0	0	1		1	1	0	0	0	1	1
44	1	1	1	1	0	1	0	1	0		0	1	0	0	0	1	1
46	0	0	0	1	0	1	0	0	1		0	1	0	0	0	1	1
48	0	0	0	1	0	1	0	1	0		1	1	0	0	0	1	1
50	1	1	1	0	1	1	0	0	1		1	1	0	0	0	1	1
52	1	1	1	0	1	1	0	1	0		0	1	0	0	0	1	1
54	0	0	0	1	1	1	0	0	1		0	1	0	0	0	1	1
56	0	0	0	1	1	1	0	1	0		1	1	0	0	0	1	1
58	1	1	1	0	0	1	0	0	1		1	1	0	0	0	1	1
60	1	1	1	0	0	1	0	1	0		0	1	0	0	0	1	1
62	0	0	1	0	0	1	0	0	1		0	1	0	0	0	1	1
64	0	0	1	0	0	1	0	1	0		1	1	0	0	1	0	0
66	1	1	0	1	1	1	0	0	1		1	1	0	0	1	0	0
68	1	1	0	1	1	1	0	1	0		0	1	0	0	1	0	0
70	0	0	1	0	1	1	0	0	1		0	1	0	0	1	0	0
72	0	0	1	0	1	1	0	1	0		1	1	0	0	1	0	0
74	1	1	0	1	0	1	0	0	1		1	1	0	0	1	0	0
76	1	1	0	1	0	1	0	1	0		0	1	0	0	1	0	0
78	0	0	1	1	0	1	0	0	1		0	1	0	0	1	0	0
80	0	0	1	1	0	1	0	1	0		1	1	0	0	1	0	1
82	1	1	0	0	1	1	0	0	1		1	1	0	0	1	0	1
84	1	1	0	0	1	1	0	1	0		0	1	0	0	1	0	1
86	0	0	1	1	1	1	0	0	1		0	1	0	0	1	0	1

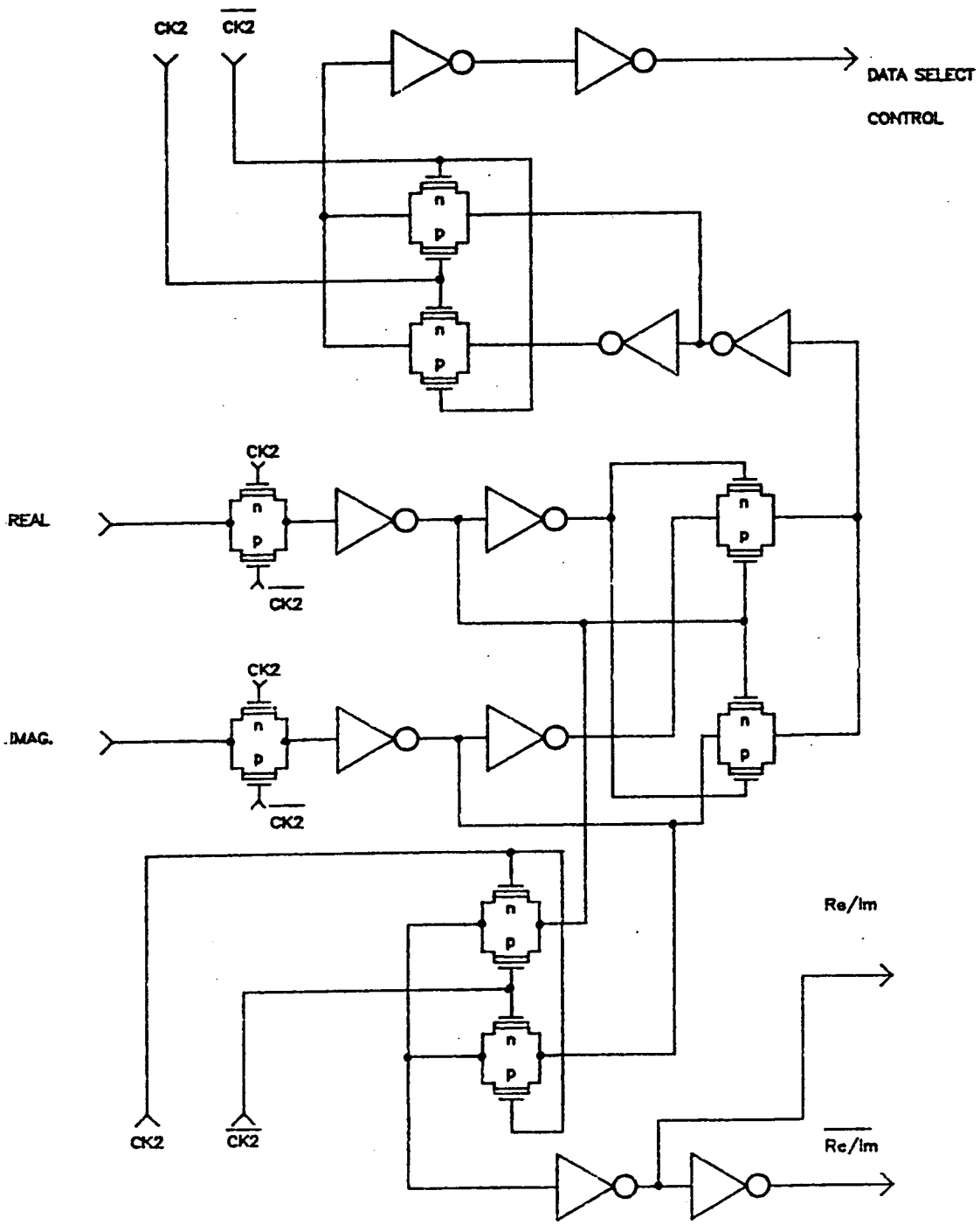
Figure(5.7.1.)

88	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	1	1
90	0	1	0	0	0	1	0	0	1	1	1	0	0	1	0	1	1
92	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	1	1
94	1	1	0	0	0	1	0	0	1	0	1	0	0	1	0	1	1
96	1	1	0	0	0	1	0	1	0	1	1	0	0	1	1	0	0
98	0	0	1	1	1	1	0	0	1	1	1	0	0	1	1	0	0
100	0	0	1	1	1	1	0	1	0	0	1	0	0	1	1	0	0
102	1	1	0	0	1	1	0	0	1	0	1	0	0	1	1	0	0
104	1	1	0	0	1	1	0	1	0	1	1	0	0	1	1	0	1
106	0	0	1	1	0	1	0	0	1	1	1	0	0	1	1	0	1
108	0	0	1	1	0	1	0	1	0	0	1	0	0	1	1	0	1
110	1	1	0	1	0	1	0	0	1	0	1	0	0	1	1	0	1
112	1	1	0	1	0	1	0	1	0	1	1	0	0	1	1	1	0
114	0	0	1	0	1	1	0	0	1	1	1	0	0	1	1	1	0
116	0	0	1	0	1	1	0	1	0	0	1	0	0	1	1	1	0
118	1	1	0	1	1	1	0	0	1	0	1	0	0	1	1	1	0
120	1	1	0	1	1	1	0	1	0	1	1	0	0	1	1	1	1
122	0	0	1	0	0	1	0	0	1	1	1	0	0	1	1	1	1
124	0	0	1	0	0	1	0	1	0	0	1	0	0	1	1	1	1
126	1	1	1	0	0	1	0	0	1	0	1	0	0	1	1	1	1
128	1	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0	0
130	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0
132	0	0	0	1	1	1	0	1	0	0	1	0	0	0	0	0	0
134	1	1	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0
136	1	1	1	0	1	1	0	1	0	1	1	0	0	0	0	0	1
138	0	0	0	1	0	1	0	0	1	1	1	0	0	0	0	0	1
140	0	0	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1
142	1	1	1	1	0	1	0	0	1	0	1	0	0	0	0	0	1
144	1	1	1	1	0	1	0	1	0	1	1	0	0	0	0	1	0
146	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	1	0
148	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	1	0
150	1	1	1	1	1	1	0	0	1	0	1	0	0	0	0	1	0
152	1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	1	1
154	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	1	1
156	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	1
158	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1
160	0	0	0	0	0	1	0	1	0	1	1	0	0	0	1	0	0
162	1	1	1	1	1	1	0	0	1	1	1	0	0	0	1	0	0
164	1	1	1	1	1	1	0	1	0	0	1	0	0	0	1	0	0
166	0	0	0	0	1	1	0	0	1	0	1	0	0	0	1	0	0
168	0	0	0	0	1	1	0	1	0	1	1	0	0	0	1	0	1
170	1	1	1	1	0	1	0	0	1	1	1	0	0	0	1	0	1
172	1	1	1	1	0	1	0	1	0	0	1	0	0	0	1	0	1

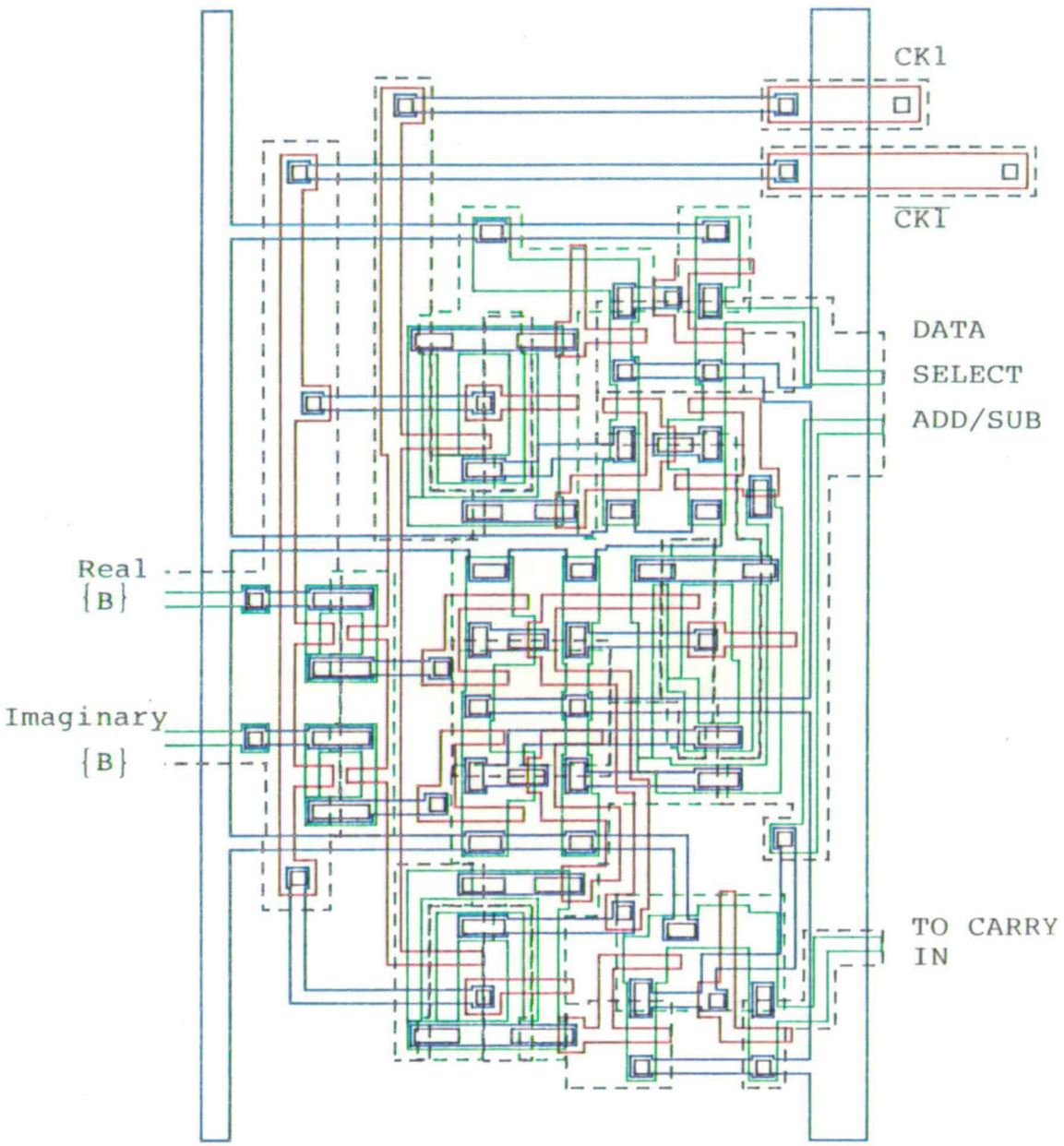
Figure(5.7.1.)

174	0	0	0	1	0	1	0	0	1	0	1	0	0	0	1	0	1
176	0	0	0	1	0	1	0	1	0	1	1	0	0	0	1	1	0
178	1	1	1	0	1	1	0	0	1	1	1	0	0	0	1	1	0
180	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	1	0
182	0	0	0	1	1	1	0	0	1	0	1	0	0	0	1	1	0
184	0	0	0	1	1	1	0	1	0	1	1	0	0	0	1	1	1
186	1	1	1	0	0	1	0	0	1	1	1	0	0	0	1	1	1
188	1	1	1	0	0	1	0	1	0	0	1	0	0	0	1	1	1
190	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	1	1
192	0	0	1	0	0	1	0	1	0	1	1	0	0	1	0	0	0
194	1	1	0	1	1	1	0	0	1	1	1	0	0	1	0	0	0
196	1	1	0	1	1	1	0	1	0	0	1	0	0	1	0	0	0
198	0	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0
200	0	0	1	0	1	1	0	1	0	1	1	0	0	1	0	0	1
202	1	1	0	1	0	1	0	0	1	1	1	0	0	1	0	0	1
204	1	1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	1
206	0	0	1	1	0	1	0	0	1	0	1	0	0	1	0	0	1
208	0	0	1	1	0	1	0	1	0	1	1	0	0	1	0	1	0
210	1	1	0	0	1	1	0	0	1	1	1	0	0	1	0	1	0
212	1	1	0	0	1	1	0	1	0	0	1	0	0	1	0	1	0
214	0	0	1	1	1	1	0	0	1	0	1	0	0	1	0	1	0
216	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	1	1
218	0	1	0	0	0	1	0	0	1	1	1	0	0	1	0	1	1
220	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	1	1
222	1	1	0	0	0	1	0	0	1	0	1	0	0	1	0	1	1
224	1	1	0	0	0	1	0	1	0	1	1	0	0	1	1	0	0
226	0	0	1	1	1	1	0	0	1	1	1	0	0	1	1	0	0
228	0	0	1	1	1	1	0	1	0	0	1	0	0	1	1	0	0
230	1	1	0	0	1	1	0	0	1	0	1	0	0	1	1	0	0
232	1	1	0	0	1	1	0	1	0	1	1	0	0	1	1	0	1
234	0	0	1	1	0	1	0	0	1	1	1	0	0	1	1	0	1
236	0	0	1	1	0	1	0	1	0	0	1	0	0	1	1	0	1
238	1	1	0	1	0	1	0	0	1	0	1	0	0	1	1	0	1
240	1	1	0	1	0	1	0	1	0	1	1	0	0	1	1	1	0
242	0	0	1	0	1	1	0	0	1	1	1	0	0	1	1	1	0
244	0	0	1	0	1	1	0	1	0	0	1	0	0	1	1	1	0
246	1	1	0	1	1	1	0	0	1	0	1	0	0	1	1	1	0
248	1	1	0	1	1	1	0	1	0	1	1	0	0	1	1	1	1
250	0	0	1	0	0	1	0	0	1	1	1	0	0	1	1	1	1
252	0	0	1	0	0	1	0	1	0	0	1	0	0	1	1	1	1
254	1	1	1	0	0	1	0	0	1	0	1	0	0	1	1	1	1

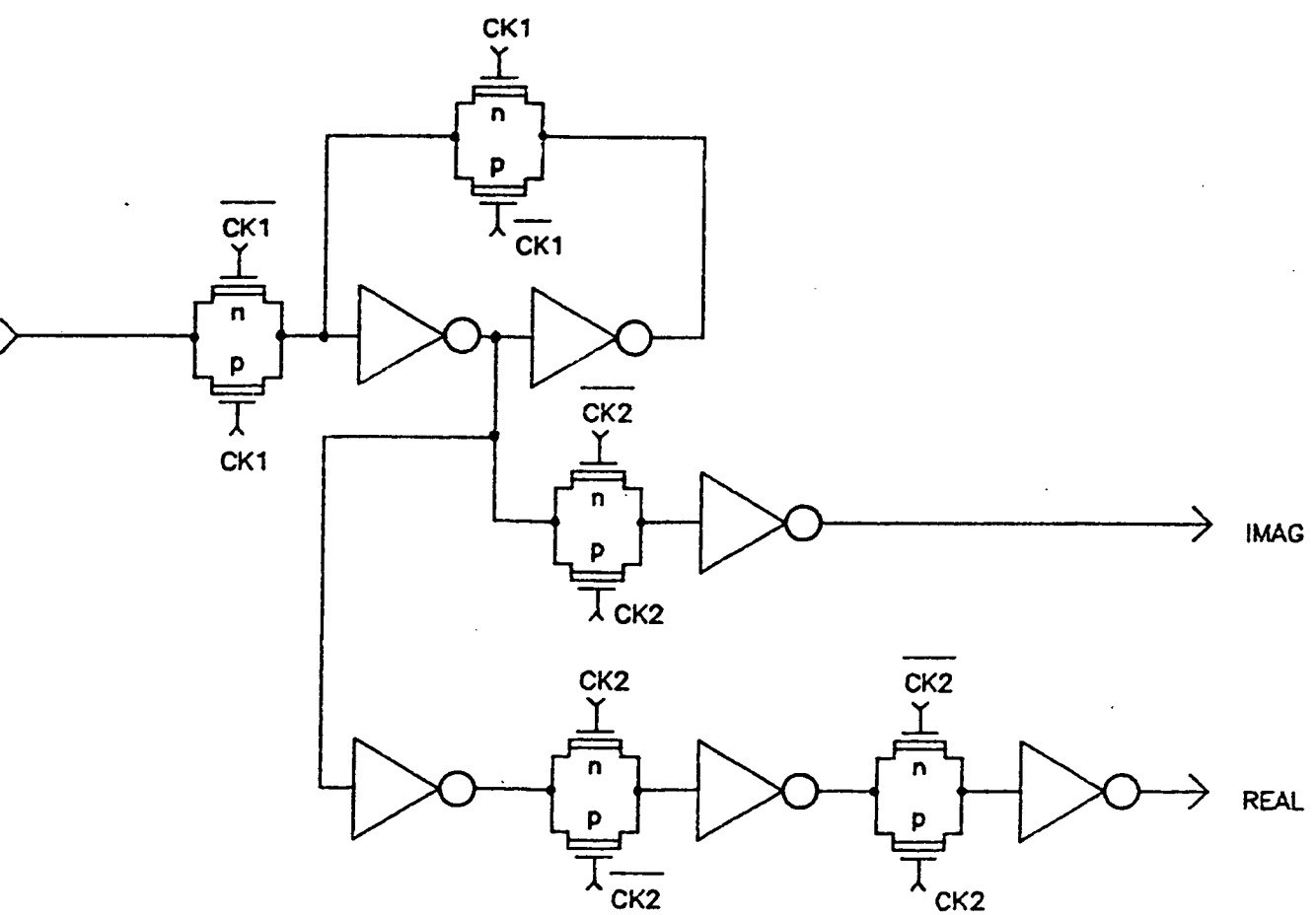
Figure(5.7.1.)



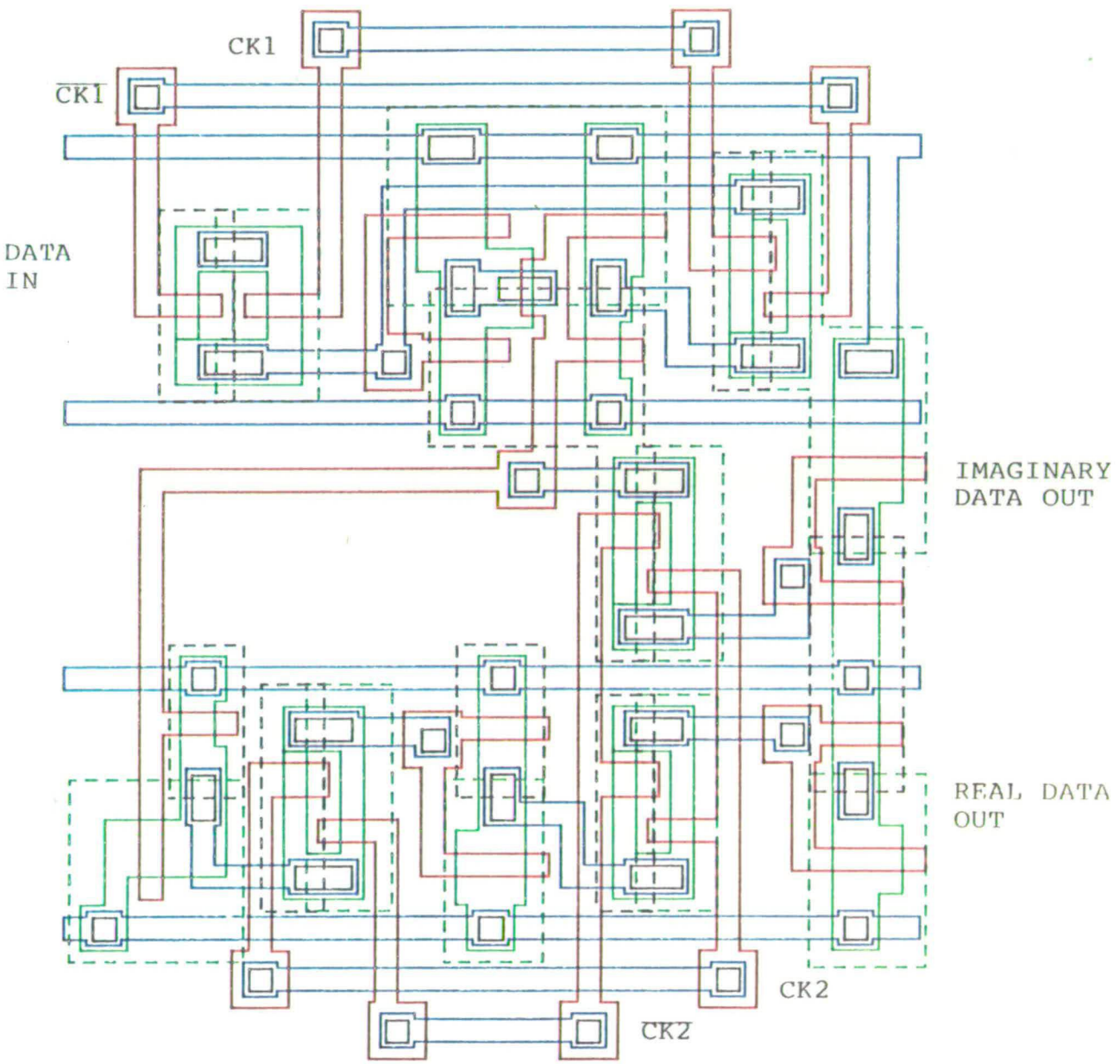
Figure(5.8.1.) Control Cell used in Z686-SOS. Device



Figure(5.8.2.) SOS Layout of Control Cell



Figure(5.8.3.) High Speed De-multiplexer used to separate Real and Imaginary Data.



Figure(5.8.4.) SOS Layout of De-multiplexer

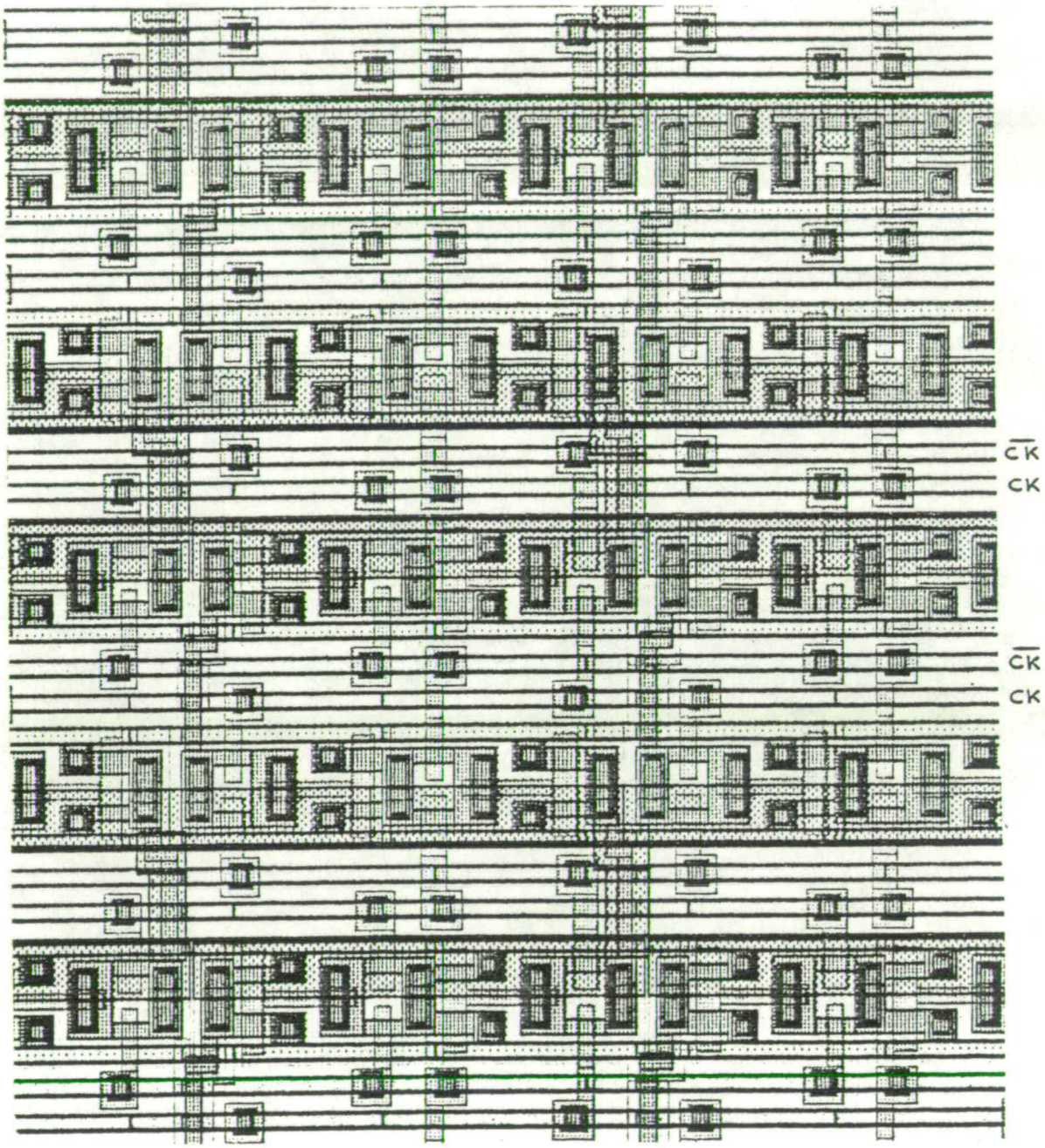


Figure (5.8.5.) SOS Layout of Dynamic Shift Register

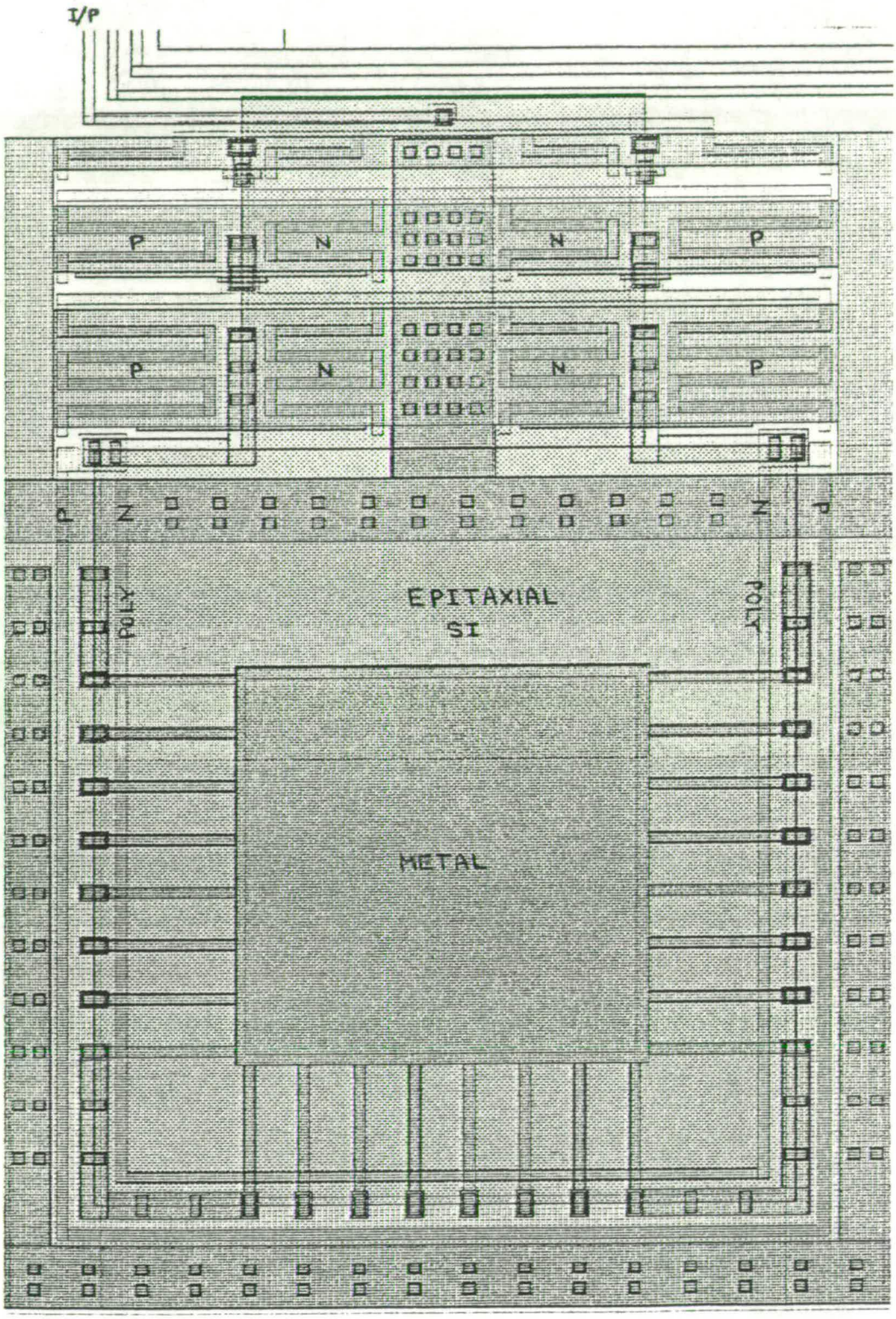
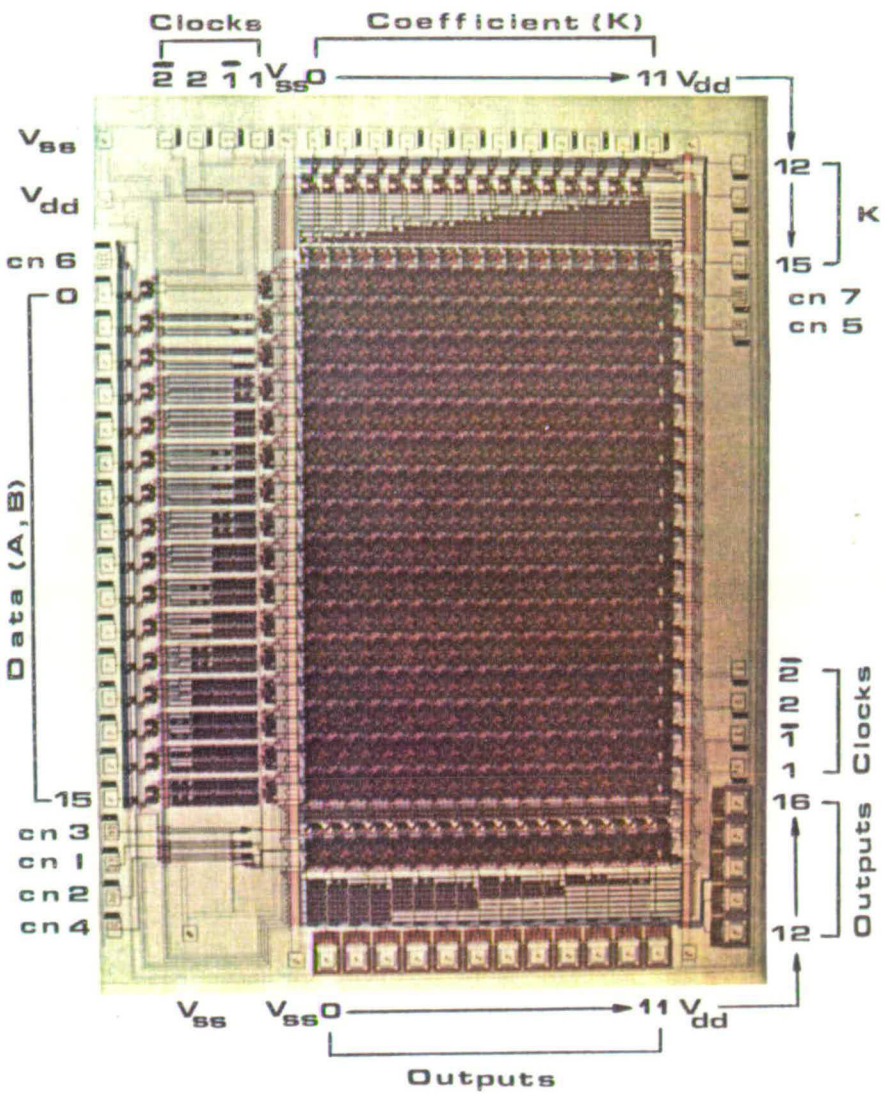


Figure (5.8.6.) SOS Layout of Output Pad



Z686-SOS

Figure(5.8.7.) Completed CMOS-SOS Design with Pad Identification.

shows the output pad used. The complete chip design is shown in Figure (5.8.7) which includes pad identification. It is expected that this device will have a maximum clocking rate of around 40MHz and consume around 0.5 Watts, including clock generation.

5.9. Summary

This chapter has described three LSI to VLSI complexity parallel data arithmetic datapaths which use some of the architectures presented in Chapter 4 for computing the complex multiply very efficiently. It is evident that the use of distributed arithmetic greatly facilitates a regular design approach which is particularly advantageous in the case of parallel data implementations as it avoids the need for multiple bus structures. The testing of device number EU219 is described in depth and it is hoped that the CMOS-SOS device (Z686-SOS) will be fabricated and tested in the near future to provide the basis for a high performance DFT or FFT processing system. Both EU219 and Z686-SOS have the throughput equivalence of two parallel real multipliers by virtue of the distributed arithmetic techniques used. The architectures described therefore offer about double the throughput that comparable [2] single parallel multiplier FFT arithmetic processors can offer.

References

1. D. J. Myers, "Multipliers for LSI and VLSI Signal Processing Application," MSc Project Report MSP5, (September 1981).

2. R J. Karwoski, "A Four-Cycle Butterfly Arithmetic Architecture," LSI Publication TP9-4/80, TRW LSI PRODUCTS 2525 E.El Segundo Blvd. El Segundo, CA 90245 (213) 535-1831, (April 1981).

Chapter 6 - Conclusions

This thesis has covered various aspects in the application of Very Large Scale Integration (VLSI) to digital computation of the discrete Fourier transform and the fast Fourier transform which approximates the continuous Fourier transform. In particular, much of the work has concentrated on VLSI architectures for implementing the arithmetic requirements of these computations with high data rates.

A variety of algorithms and system design methodologies have been reviewed in order to highlight the range of structures that are possible. It has been observed that whilst the discrete Fourier transform can be easily realised as a single monolithic VLSI processor, the fast Fourier transform is not quite so easily realised as a single monolithic device owing to the higher levels of integration required. A notable characteristic of the fast Fourier transform is that various distinct levels of arithmetic concurrency are possible which allows a great variety of system configurations and partitions as well as processing bandwidths. It has been noted, however, that very high bandwidth systems must be based on a system design methodology which allows a high degree of programmability to obtain the necessary versatility for system use. It is believed that configurable pipelines can offer both high bandwidths and suitable versatility for systems use, although it is recognised that silicon compiler approaches, such as "FIRST" described in chapter 2 which obtain versatility by offering control over the actual hardware, has a role to play where the extra cost of mask-making and fabrication is not a major consideration.

It is apparent that an important requirement for efficient VLSI designs is regularity and modularity. Not only do such approaches allow quick design times, but they are more likely to result in structures to which yield enhancement approaches can be applied if required. It is noted that distributed arithmetic approaches in particular can yield highly regular structures thus allowing the design of highly efficient VLSI arithmetic processors. In this thesis, a great deal of attention has been focussed on the arithmetic requirements of the fast Fourier transform as it has been noted that the availability of a high performance arithmetic unit does not constrain a system to any one level of arithmetic concurrency or impose limitations on transform size (assuming wordlengths are adequate).

Chapter 5 has described three LSI/VLSI MOS distributed arithmetic devices which offer very high bandwidths. In the case of the complex multiply, the throughput rate is effectively doubled as a result of using distributed arithmetic. Moreover, these approaches offer a highly regular VLSI design approach. Figure (6.1) shows a comparison of the expected performance of the CMOS-SOS device (Z686-SOS) and the measured performance of EU219 in computing complex multiply and accumulate operations against that which could be obtained using commercially available devices, including single real multipliers in conjunction with accumulators. The TRW figure is based on the MPY16HJ 16 by 16 bit real multiplier device, which is commonly available. The GaAs figure is derived from the 16 by 16 bit multiplier device described in chapter 2.

It is believed that distributed arithmetic offers a significantly lower power consumption than is possible with conventional

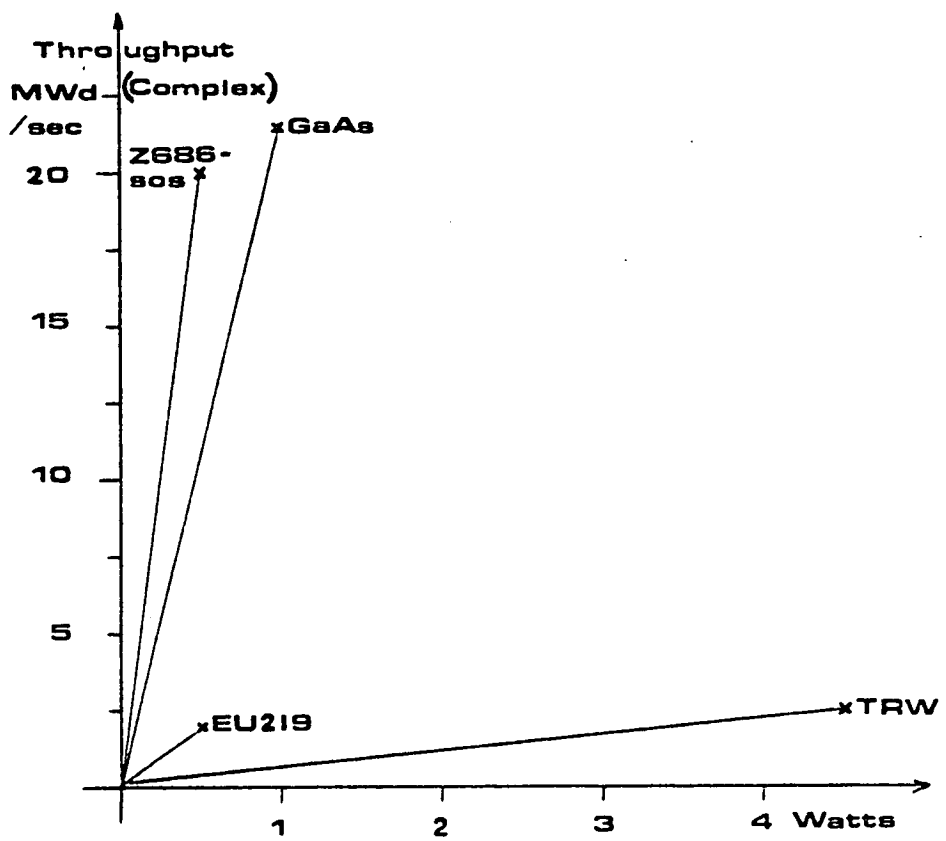


Figure (6.1) Comparision of Commercial Devices with Datapaths described in this thesis in Computing Complex Multiplications

multiplier based arithmetic. This reasoning is based on the fact that only a single accumulation process is required. The extra power consumption required by data-select operations can be made small as the number of logic gates required to toggle in a given data-select operation is not large. Low power consumptions are particularly important in high bandwidth fast Fourier transform systems, such as are used in avionics, where heat dissipation constraints and power constraints often exist.

6.1. Future Research Work

The various devices described in this thesis have moved progressively towards some form of (low level) programmability through pipeline re-configuration. It is felt that this area could be further explored. The principle advantage of developing configurable arithmetic pipelines further is not only the consequent increase in versatility, but also the possibility of bypassing defective circuitry through redundancy which would provide a basis for yield enhancement. Distributed arithmetic has a definite role to play in the construction of such pipelines because of the regularity which it allows. Looking ahead to wafer scale integration, power consumptions become increasingly important. Here again it is believed that distributed arithmetic will have a role to play.

Appendix 1

Programmable Logic Array's (PLA's) in Silicon Compilers

The main problem with the PLA, is that its speed is dependent on its size. Larger PLA structures cannot be clocked as fast as smaller ones. This is due to an increase in internal capacitances which cannot be matched with a lowering in the "ON" resistance of the switching elements in that structure. A clocked control circuit using a number of PLA's of different size will only clock as fast as the slowest PLA. The PLA itself, however, is a particularly effective way of mapping logic directly on to silicon.

A silicon-compiler that generates synchronous assemblies is virtually useless if it has no means of controlling the actual speed of individual clocked units in that assembly in order to achieve some control over performance. If an assembly is constructed from a number of PLA's, some means therefore must be found to allow control over the speed of individual PLA's in that assembly. This would then allow some degree of optimisation to be built into the system as well as allowing overall system clocking rates to be achieved that might otherwise not be possible.

PLA Performance Control

The regularity of the PLA has a significant advantage, in addition to those already mentioned. It allows easy calculation of the capacitances in the "AND" and "OR" planes of the PLA as a function of inputs, product terms, outputs, and if desired, the truth table itself. This function should also include process parameters, which would normally be fixed. A knowledge of these capacitances,

together with transistor aspect ratios allows the maximum operating speed of the PLA to be calculated. An experimental FORTRAN program was written, which could both perform this speed calculation and generate nMOS transistor layouts with a variety of aspect ratios (continuously variable). This subroutine, which could generate variable aspect ratio transistors would allow some control over PLA speed performance. The algorithm (not consciously copied) that was used in this speed programmable PLA generator is outlined in Figure (1). This program was primarily written in order to empirically determine the extent of control possible over the PLA speed performance using software techniques. Results are outlined below, in the section "PLA Run Results".

This program might be incorporated in the complier discussed previously as a "soft" operator, allowing several PLA's to be interconnected automatically on a one dimensional routing channel. Such a compiler, suited to control tasks, would offer a high degree of performance control. The design rules used in the PLA generator described here, were based on fixed Mead and Conway type rules for nMOS, however, it would have been feasible to produce PLA silicon layout which was a function of certain design rules, that might have been liable to fluctuations. (The program took into account process length and width modifications.)

PLA Run Results

There were two main points that were noted in running the speed programmable PLA generator program. Firstly, there was a minimum area which occurred at some particular speed. Secondly, once this area was reached, higher speed requirements forced the area to rise

very sharply (and power consumption). This effect is mainly due to the relative interconnect capacitances tending to a very low value. Figure (2) shows the actual silicon layout of four different speed performance decade counters produced by this program. The rise in area was quite pronounced as shown in Figure (3) which shows the area of a decade counter at various speed requests.

It was found that a useful PLA speed range of about 10:1 could be produced by the program. At low speeds, the depletion pull-up devices started to consume too much area and at high speeds the enhancement pull-downs started to cause an area "explosion" as the relative interconnect to active-area capacitance tended to very low values.

It was felt that although little could be done to avoid this area explosion and therefore little done to produce clocking rates beyond a critical point just after this takes place, the program would go some way to optimising the performance of a compiled synchronous control system which is based on a number of different software generated PLA's.

The PLA compiler was used to produce several silicon layouts of counters which were subsequently fabricated and exhibited maximum clocking rates slightly greater (about 20%) than that requested by the software.

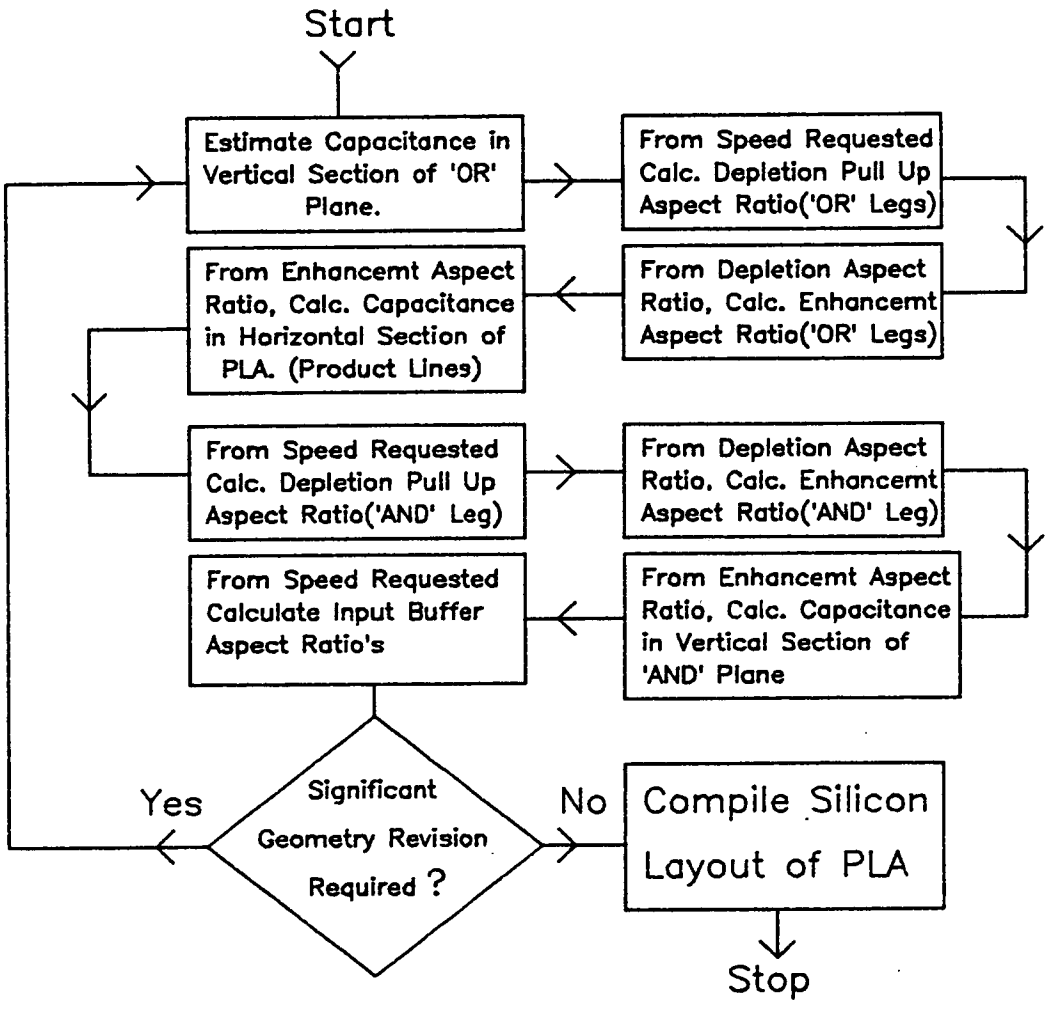

Start
Estimate Capacitance in Vertical Section of 'OR' Plane.
From Speed Requested Calc. Depletion Pull Up Aspect Ratio('OR' Legs)
From Depletion Aspect Ratio, Calc. Enhancemt Aspect Ratio('OR' Legs)
From Enhancemt Aspect Ratio, Calc. Capacitance in Horizontal Section of PLA. (Product Lines)
From Speed Requested Calc. Depletion Pull Up Aspect Ratio('AND' Leg)
From Depletion Aspect Ratio, Calc. Enhancemt Aspect Ratio('AND' Leg)
From Speed Requested Calculate Input Buffer Aspect Ratio's
From Enhancemt Aspect Ratio, Calc. Capacitance in Vertical Section of 'AND' Plane
Significant Geometry Revision Required ?
Yes
No
Compile Silicon Layout of PLA
Stop

Figure 1

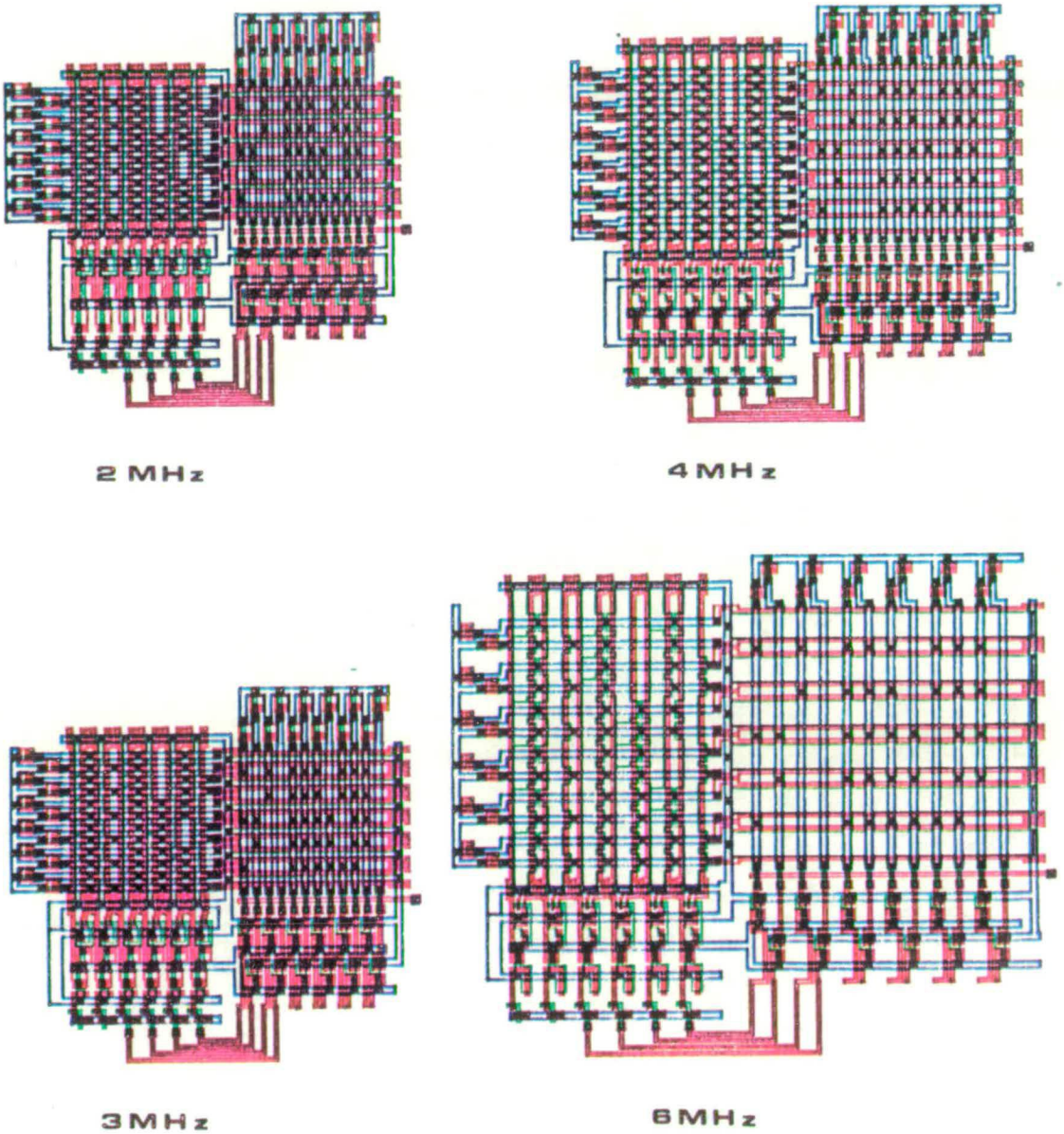


Figure 2. Silicon Layout from Silicon Compiler

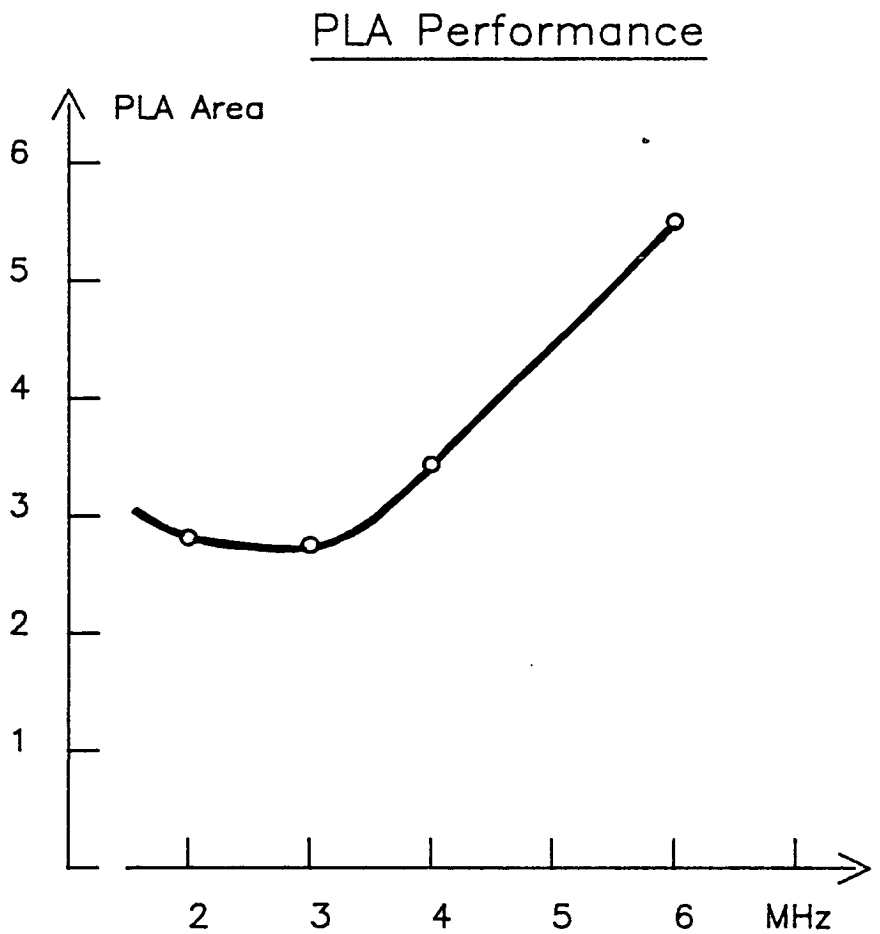


Figure 3

APPENDIX 2

AUTHORS

PUBLICATIONS

RADIX-2 FFT BUTTERFLY PROCESSOR USING DISTRIBUTED ARITHMETIC

Indexing terms: Computers, Fast Fourier transforms, Processors

A parallel-data VLSI architecture for computation of the fast Fourier transform (FFT) is described. The processor is based on a computationally efficient vector rotate algorithm. Use of a 2-dimensional pipeline configuration allows a radix-2 butterfly operation to be performed once every system clock cycle (250 ns) to generate real or imaginary transform components. The architecture is considered to be a computationally efficient VLSI approach for high-bandwidth computation of the FFT. The design and performance of an 8-bit FFT butterfly processor are described.

Central to computation of the FFT algorithm[1] is a requirement for vector rotation (multiplication by a unit vector). The vector rotation involved in the computation of the FFT butterfly can be written

$$\mathrm{Re}\{Z\} = \mathrm{Re}\{B\} \cdot \mathrm{Re}\{W\} - \mathrm{Im}\{B\} \cdot \mathrm{Im}\{W\} \quad (1)$$

$$\mathrm{Im}\{Z\} = \mathrm{Re}\{B\} \cdot \mathrm{Im}\{W\} + \mathrm{Im}\{B\} \cdot \mathrm{Re}\{W\} \quad (2)$$

where B is the input data vector, Z is the output data vector and W is a unit vector termed the 'twiddle factor'.[2] Thus direct implementation of vector rotation involves four multiplications plus an addition and a subtraction. Computation of the Radix-2 FFT butterfly requires a further two additions and two subtractions.[2]

The algorithm employed in the processor considered here is based on the fact that when the outputs of two or more digital (shift-and-add) multipliers are involved in subsequent arithmetic operations such as addition or subtraction, more efficient use can be made of the multiplier structure using distributed arithmetic techniques.[3]

For the vector rotation arrangement described by eqns. 1 and 2, there exist only four possible ways in which the separate components of Z considered on a bit-by-bit basis will be modified by the results of the multiplication operations. These are described in Table 1.

Table 1 PARTIAL PRODUCT FORMATION IN VECTOR ROTATION

Twiddle factor W (bit level)		Resulting partial product Z (word level)	
$\mathrm{Re}\{W\}$	$\mathrm{Im}\{W\}$	$\mathrm{Re}\{Z\}$	$\mathrm{Im}\{Z\}$
0	0	0 − 0	0 + 0
0	1	0 − Im {B}	Re {B} + 0
1	0	Re {B} − 0	0 + Im {B}
1	1	Re {B} − Im {B}	Re {B} + Im {B}

If, instead of using $\mathrm{Re}\{B\}$ and $\mathrm{Im}\{B\}$, the values $K = [\mathrm{Re}\{B\} + \mathrm{Im}\{B\}]/2$ and $K' = [\mathrm{Re}\{B\} - \mathrm{Im}\{B\}]/2$ are made available, it is possible to compute each of these four possible modifications to the partial products of Z for each bit of W as shown in Table 2.

Table 2 PARTIAL PRODUCT FORMATION USING DISTRIBUTED ARITHMETIC

Twiddle factor W (bit level)		Resulting partial product Z (word level)	
$\mathrm{Re}\{W\}$	$\mathrm{Im}\{W\}$	$\mathrm{Re}\{Z\}$	$\mathrm{Im}\{Z\}$
0	0	$K' - [\mathrm{Re}\{B\} - \mathrm{Im}\{B\}]/2$	$K - [\mathrm{Re}\{B\} + \mathrm{Im}\{B\}]/2$
0	1	$K' - [\mathrm{Re}\{B\} + \mathrm{Im}\{B\}]/2$	$K + [\mathrm{Re}\{B\} - \mathrm{Im}\{B\}]/2$
1	0	$K' + [\mathrm{Re}\{B\} + \mathrm{Im}\{B\}]/2$	$K - [\mathrm{Re}\{B\} - \mathrm{Im}\{B\}]/2$
1	1	$K' + [\mathrm{Re}\{B\} - \mathrm{Im}\{B\}]/2$	$K + [\mathrm{Re}\{B\} + \mathrm{Im}\{B\}]/2$

where $K = [\mathrm{Re}\{B\} + \mathrm{Im}\{B\}]/2$ and $K' = [\mathrm{Re}\{B\} - \mathrm{Im}\{B\}]/2$

The K and K' terms as defined in Table 2 are independent of W and can therefore be derived separately from the main accumulation process.

In 2's complement notation, an N-bit digital word can be represented as:

$$A = -a_0 2^0 + \sum_{n=1}^{N-1} a_n 2^{-n} \quad (3)$$

Table 2 can be further simplified since, using 2's complement notation, the subtraction of K' involved in the most significant partial product formation would serve to cancel the accumulated (increasing significance) K' terms, except for a lowest significance term in K'. For the $\mathrm{Im}\{Z\}$ partial products, a similar argument applies for K. The individual partial products of Z can thus be formed by an add or subtract operation involving K or K' at each computation cycle. The selection of K or K' is made on the basis of the exclusive OR or exclusive NOR combination of $\mathrm{Re}\{W\}$ and $\mathrm{Im}\{W\}$. The add/subtract selection is made on the basis of $\mathrm{Re}\{W\}$ or $\mathrm{Im}\{W\}$ for $\mathrm{Re}\{Z\}$ or $\mathrm{Im}\{Z\}$, respectively (see Table 2).

The salient feature of this algorithm is that $\mathrm{Re}\{Z\}$ can be formed by a single accumulation-type operation, and similarly $\mathrm{Im}\{Z\}$ can be formed by a separate accumulation operation thus permitting the design to partition into two identical parts with a single control line to define real or imaginary outputs.

This distributed arithmetic algorithm has been realised as a single monolithic circuit based on a 2-dimensionally pipelined configuration which allows the constant throughput of parallel data. The chip architecture incorporates an array of 2-bit-

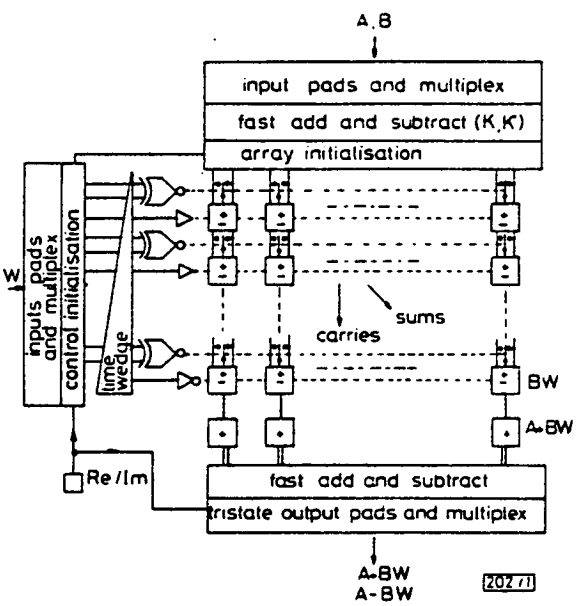


Fig. 1 *Block diagram of distributed arithmetic processor*

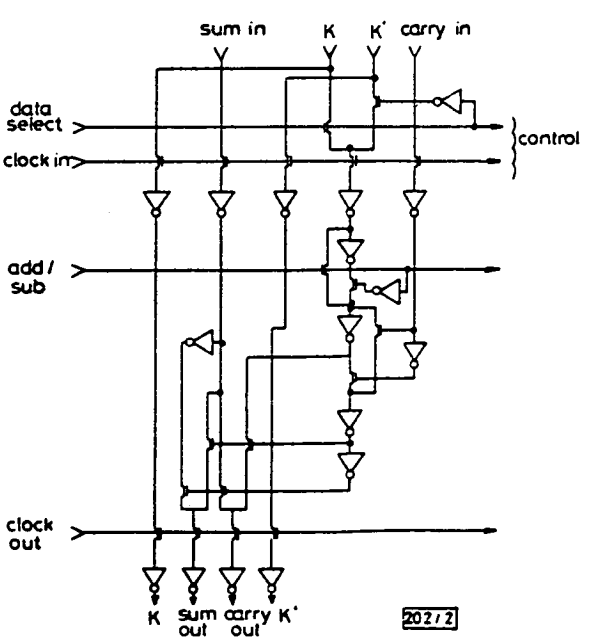


Fig. 2 *Basic cell*

e shift registers which shift the (modified) input words K K' separately from 'top-to-bottom' of the chip through an y of full adder cells, Fig. 1. A data select on these two it words (K and K') is performed at the inputs to each full er cell under the control of the exclusive-OR or exclusive-R block which operates on the separate bits of Re $\{W\}$ Im $\{W\}$ (see Table 2). The word selected (i.e. either K or at each full adder is then either added to, or subtracted n, the accumulated partial product as determined by the ropriate bit of Re $\{W\}$ (for the Re $\{Z\}$ accumulator) or $\{W\}$ (for the Im $\{Z\}$ accumulator), Fig. 2. A time wedge ed on the W input port ensures that a constant data oughput can be maintained in the pipeline.

nclusion: A 2-dimensionally pipelined FFT butterfly pro-or based on a distributed arithmetic algorithm has been orted.

n 8-bit processor based on this algorithm has been fabri-ed on a 5 μm single polysilicon, single metal, N-channel ancement depletion MOS process. The chip, measuring 6·7 6·3 mm, contained approximately 8000 transistors. Power consumption was 0·5 W at the designed speed of 4 MHz. The architecture discussed here is currently being extended to a 16-bit CMOS-SOS implementation, using 3·5 μm geometries.

Acknowledgments: This work was carried out under a UK Science & Engineering Research Council grant.

I. R. MACTAGGART
M. A. JACK
Department of Electrical Engineering
University of Edinburgh
Mayfield Road, Edinburgh EH9 3JL, Scotland

3rd December 1982

References

1 COOLEY, J. W., and TUKEY, J. W.: 'An algorithm for the machine calculation of complex Fourier series', *Math. Comput.*, 1965, **19**
2 BRIGHAM, E. O.: 'The fast Fourier transform' (Prentice Hall, 1974)
3 WHITE, S. A.: 'A simple FFT butterfly arithmetic unit', *IEEE Trans.*, 1981, **CAS-28**

0013-5194/83/020043-02$1.50/0

A DISTRIBUTED ARITHMETIC RADIX-2 FFT BUTTERFLY PROCESSOR

I.R.Mactaggart and M.A.Jack
Department of Electrical Engineering,
Edinburgh University,
Scotland

Abstract

An efficient distributed arithmetic architecture for computation of the Radix-2 FFT butterfly is reported. Results for a prototype NMOS processor exhibit data rates in excess of 8 Mbytes/second.

1. Introduction

This paper describes an eight-bit NMOS processor chip based on an efficient, distributed arithmetic complex multiply algorithm [1,2]. The algorithm is used to compute the Radix-2 FFT Decimation-in-Time "butterfly" [3]. This approach to the computation yields a highly regular structure which is particularly advantageous with parallel arithmetic systems.

2. The Algorithm

The FFT algorithm, like the DFT, has a requirement for vector rotation. This can be realised conveniently as the multiplication by a unit vector in a cartesian coordinate system. The equations for this vector rotation can thus be written as :

$$Re\{Z\} = Re\{B\}.Re\{W\} - Im\{B\}.Im\{W\} \quad (1)$$
$$Im\{Z\} = Re\{B\}.Im\{W\} + Im\{B\}.Re\{W\} \quad (2)$$

Where B is the data vector, Z is the output vector and W is the unit vector coefficient used to rotate B. A direct implementation of these equations requires four multiplies and two additions. The Radix-2 butterfly requires a further four real additions [3].

If, instead of using Re{B} and Im{B}, two new inputs, defined as K = [Re{B} + Im{B}]/2 and K´= [Re{B} - Im{B}]/2 are made available at various levels of significance, then it is possible to merge the partial products of the Re{Z} multipliers together in a single accumulator structure, and similarly it is possible to merge the Im{Z} multiplier partial products into a single accumulation, as illustrated in Table 1.

Coefficient (W) (bit level)		Resulting Partial Product (Z) (word level)	
Re{W}	Im{W}	Re{Z}	Im{Z}
0	0	K´-[Re{B}-Im{B}]/2	K-[Re{B}+Im{B}]/2
0	1	K´-[Re{B}+Im{B}]/2	K+[Re{B}-Im{B}]/2
1	0	K´+[Re{B}+Im{B}]/2	K-[Re{B}-Im{B}]/2
1	1	K´-[Re{B}-Im{B}]/2	K+[Re{B}+Im{B}]/2

Where K = [Re{B}+Im{B}]/2 and K´ = [Re{B}-Im{B}]/2

Table 1 Showing Formation of Z using Distributed Arithmetic

Table 1 illustrates how the exclusive-OR and exclusive-NOR combination of the individual bits of Re{W} and Im{W} can be used to control the selection of words K or K´and how the Re{W} bit or Im{W} bit, for Re{Z} or Im{Z} respectively, can be used to control the add/subtract operation.

The salient feature of this algorithm is that Re{Z} and likewise Im{Z} can be formed in a single accumulation type of operation thus allowing the design to partition into a single programmable distributed arithmetic array, capable of computing either Re{Z} or Im{Z} every clock cycle.

3. Chip Architecture

The chip consists of an array of two-dimensionally pipelined cells of the type shown in Figure 1. These cells each contain a two bit wide shift register which carry K and K´ through the chip from top to bottom. At each cell, a data-select is performed under the control of the exclusive-OR/exclusive NOR gates.

The full adder used in the basic cell makes use of inverter-controlled data-select exclusive OR gates. The add and subtract control signal is fed horizontally through the cell array. A fast adder and subtractor is used at the input to generate K and K´ and similarly a fast add is used at the output to assimilate the sums and carries of the full adder array. The fast adder uses a pre-charged carry chain and was designed to operate with a settle time of 65nS, worst case. Other main features included a time wedge placed on the coefficient (W) input port, a multiplexer on both input and output ports, tri-state output pads and system control logic.

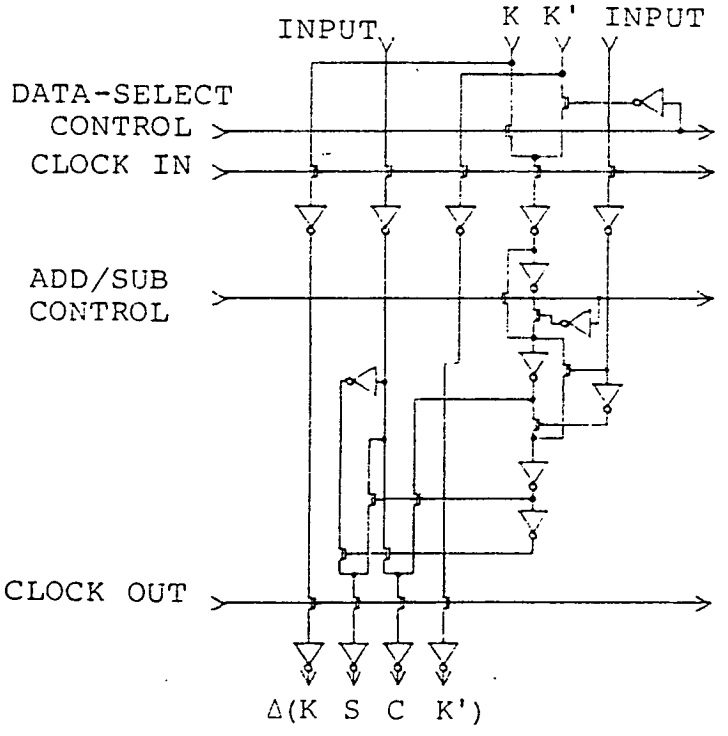


FIGURE 1 Showing Basic Cell

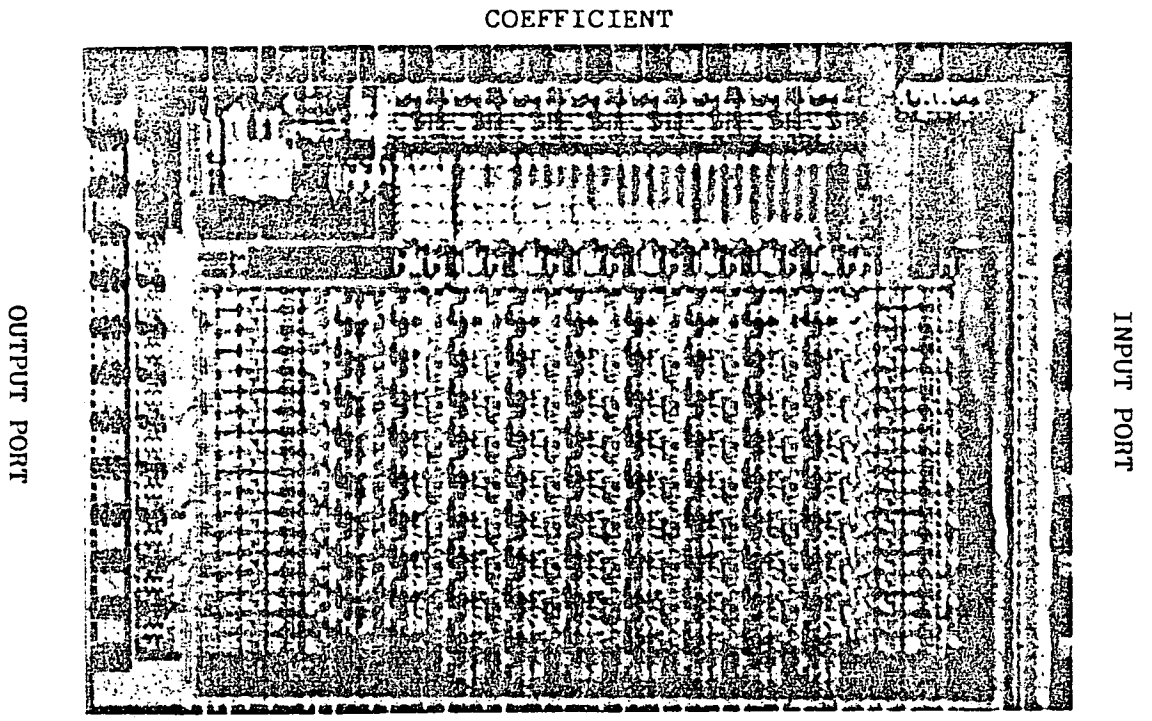


Figure 2 Photograph of Radix-2 FFT Butterfly

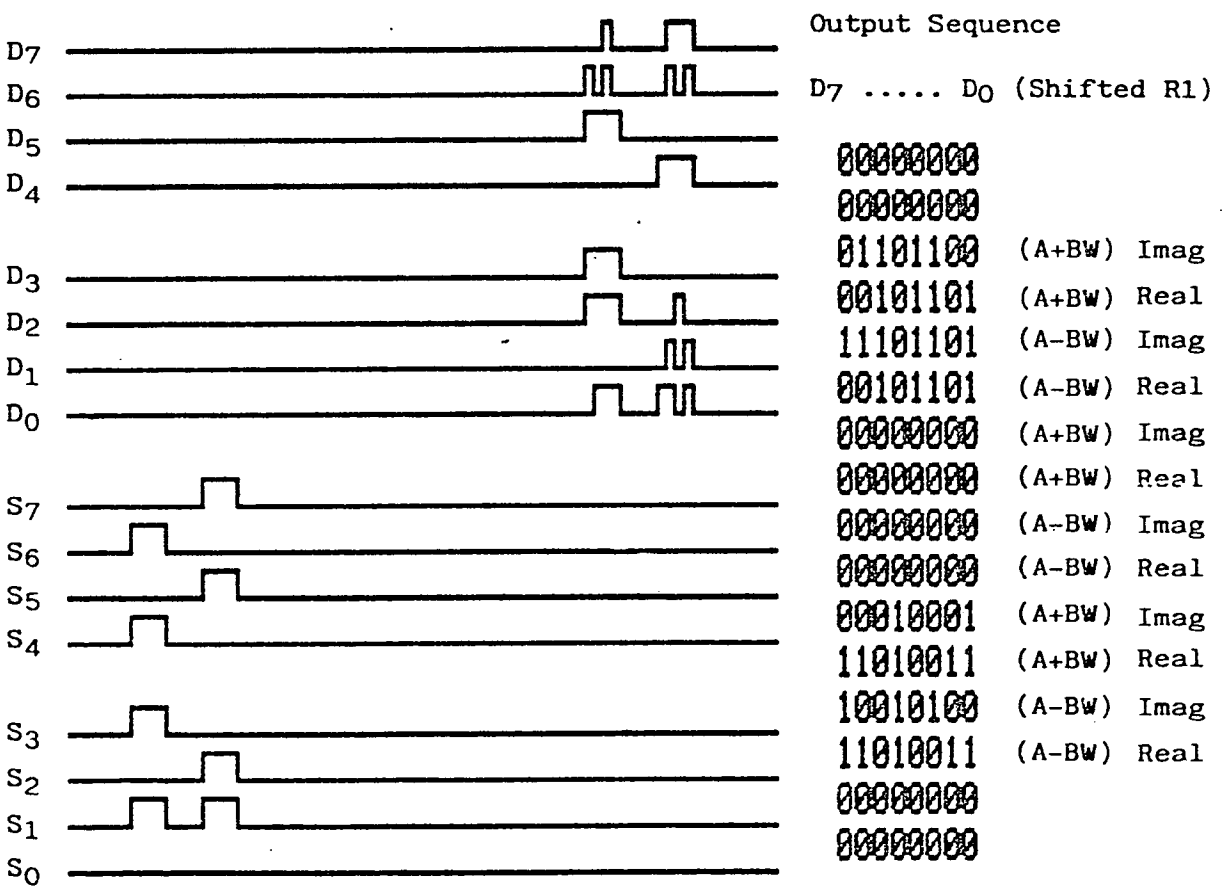


Figure 3 Dynamic Operation at 4Mbytes/s, as Recorded by Logic Analyser

4. Performance

The chip was fabricated on a 6 micron single polysilicon, single metal, N-channel enhancement/depletion MOS process, Figure 2. The chip measured 6.7 by 6.3 mm and contained around 8000 transistors. With a process yield of about 5% a number of devices were obtained and bonded up. Chip parameters are shown in Table 2. Logically, the chip performed as predicted. Figure 3 shows the input and output logic signatures obtained by feeding in four cycles of data equal to the reciprocal of the square root of two (01011010) for all inputs, and then four cycles of zero's, followed by four cycles of the negative of the reciprocal of root two (10100110) for all inputs. The figure shows the latency of the pipeline and its operation under dynamic conditions for this input sequence. This test was performed at 4 Mbytes/second.

Parameter	Value
Computation Cycle	250 nS
Data Rate	8 M Bytes/s
Word Length	8 Bits
Power Consumption	0.5 Watts (Average)
Package	40 Pin Dil

Table 2 Showing Chip Parameters

5. Conclusions

This paper has described a working monolithic FFT butterfly circuit based on a distributed arithmetic algorithm for computation of the complex multiply. It is evident that the resulting distributed arithmetic structure is area efficient and highly regular. Results for the 6 micron prototype are encouraging and given a commercial quality process, an order of magnitude improvement can be expected. The chip does, however, serve to illustrate the modularity afforded by this algorithm for systems based on parallel data flow.

It is possible to remove the requirement for a fast add by inserting extra delays in the horizontal and vertical bit streams. This approach is more suitable for larger word lengths. A 16 bit version of the chip with these extra delays included, is currently being designed for a high performance CMOS-SOS process using 3.5 micron geometries.

6. Acknowledgments

This work was carried out under a UK Science and Engineering Research Council Grant.

7. References

1. Stanley A. White : "A Simple FFT Butterfly Arithmetic Unit",
   IEEE Transactions on Circuits and Systems,
   Vol.CAS-28 No.4 April 1981
2. I. R. Mactaggart and M. A. Jack : "Radix-2 FFT Butterfly Processor
   using distributed arithmetic"
   Elecronics Letters Vol.19 No.2 January 1983
3. L. R. Rabiner and B. Gold : "Theory and Application of Digital Signal Processing".
   Prentice Hall (1975)

A DISTRIBUTED ARITHMETIC RADIX-2 FFT BUTTERFLY PROCESSOR

I.R.Mactaggart and M.A.Jack

Edinburgh University, Scotland

INTRODUCTION

Real time computation of the Fast Fourier Transform (FFT) is assuming an increasingly important role in wideband communication systems. Circuits traditionally implemented using analogue techniques will be implemented digitally as the price of digital signal processing falls. This is possible because very fast analogue-to-digital converters are becoming available (1) to enable these fast processors to operate at their full capability. High bandwidth FFT's, however, require dedicated hardware to achieve the necessary performance. These large arithmetic overheads demand special consideration for cost effective solutions. It will be shown how distributed arithmetic techniques can be used to achieve regular and efficient VLSI designs in the special case of 2-dimensional array structures. An example is given of a FFT butterfly arithmetic unit which has been designed and fabricated in NMOS and is streamlined to computing complex arithmetic with the throughput equivalence of two real parallel multipliers on a single chip. A 16 bit CMOS-SOS design to be fabricated in November '83 is compared with the NMOS design.

The Complex Multiply

The FFT algorithm (2), like the DFT, has a requirement for vector rotation. This can be realised conveniently as the multiplication by a unit vector in a Cartesian coordinate system. The equations for this vector rotation can thus be written as :

$$Re(Z) = Re(B).Re(W) - Im(B).Im(W) \quad (1)$$
$$Im(Z) = Re(B).Im(W) + Im(B).Re(W) \quad (2)$$

Where B is the data vector, Z is the output vector and W is the unit vector coefficient used to rotate B. A direct implementation of these equations requires four real multiplications and two additions. The Radix-2 butterfly requires a further four real additions (2). If we assume a standard shift-and-add multiplier scheme then we can construct a table showing the partial product formation as a function of the inputs. This is shown in Table 1.

Coefficient (W) (Bit-level)		Multiplier Partial Product (Z) (Word Level)	
nth bit		shifted n places	
Re(W)	Im(W)	Re(Z)	Im(Z)
0	0	(0 - 0)	(0 + 0)
0	1	(0 - Im(B))	(Re(B) + 0)
1	0	(Re(B) - 0)	(0 + Im(B))
1	1	(Re(B) - Im(B))	(Re(B) + Im(B))
		M1	M2	M3	M4

(FOUR MULTIPLIERS M1 - M4)

Table 1 Showing Partial Products in Complex Multiplication

In the conventional approach the final product of multiplier M2 is subtracted from M1 to form Re(Z) and M4 is added to M3 to form Im(Z). There is, however, no need to defer this subtraction and addition until final product formation in the multipliers. In distributed arithmetic (3,4), we no longer consider the multiplication as an individual isolated function , but instead we try to derive how the formation of Re(Z) and Im(Z) can be accomplished given special distributed arithmetic operands which are elementary only to the complete function (in this case, the complex multiply). If then, instead of using Re(B) and Im(B), two new inputs, defined as K = (Re(B) + Im(B))/2 and K' = (Re(B) - Im(B))/2 are made available, multiplied by some integer power of two (shifted), it is possible to form the Re(Z) and Im(Z) complex partial products in a single accumulator, as illustrated in Table 2. This approach is not only algorithmically efficient, but, also important, yields a highly regular, and area efficient structure, since real or imaginary merged partial products can be formed at each node in the distributed arithmetic array during each clock cycle.

Coefficient (W) (Bit-level)		Resulting Partial Product (Z) (Word level)	
Re(W)	Im(W)	Re(Z)	Im(Z)
0	0	K'-(Re(B)-Im(B))/2 *	K-(Re(B)+Im(B))/2 *
0	1	K'-(Re(B)+Im(B))/2	K+(Re(B)-Im(B))/2
1	0	K'+(Re(B)+Im(B))/2	K-(Re(B)-Im(B))/2
1	1	K'+(Re(B)-Im(B))/2	K+(Re(B)+Im(B))/2

Where K = (Re(B)+Im(B))/2 and K' = (Re(B)-Im(B))/2

*: W independent - do NOT enter accumulator

Table 2 Showing Formation of Z using Distributed Arithmetic

Table 2 illustrates how the exclusive-OR and exclusive-NOR combination of the individual bits of Re(W) and Im(W) can be used to control the selection of words K or K' and how the Re(W) bit or Im(W) bit, for Re(Z) or Im(Z) respectively, can be used to control the add/subtract operation. The salient feature of this algorithm is that Re(Z) and likewise Im(Z) can be formed in a single accumulation type of operation thus allowing the design to partition into a single, programmable, distributed arithmetic array, capable of computing either Re(Z) or Im(Z) every clock cycle.

Chip Architecture

The chip consists of an array of two-dimensionally pipelined cells of the type shown in Figure 1. These cells each contain a two bit wide shift register which carries K and K' through the chip from top to bottom. At each cell in the array, a data-select

performed under the control of the exclusive-OR/exclusive NOR gates. The full adder used in the basic cell makes use of inverter-controlled data-select exclusive OR gates. The add and subtract control signal is fed horizontally through the cell array. A fast adder and subtractor is used at the input to generate K and K' and similarly a fast adder is used at the output to assimilate the sums and carries of the full adder array. The fast adder uses a pre-charged carry chain and was designed to operate with a settle time of 65nS, worst case. Other main features (Figure 2) include a time wedge placed on the coefficient (W) input port, a multiplexer on both input and output ports, tri-state output pads and system control logic.

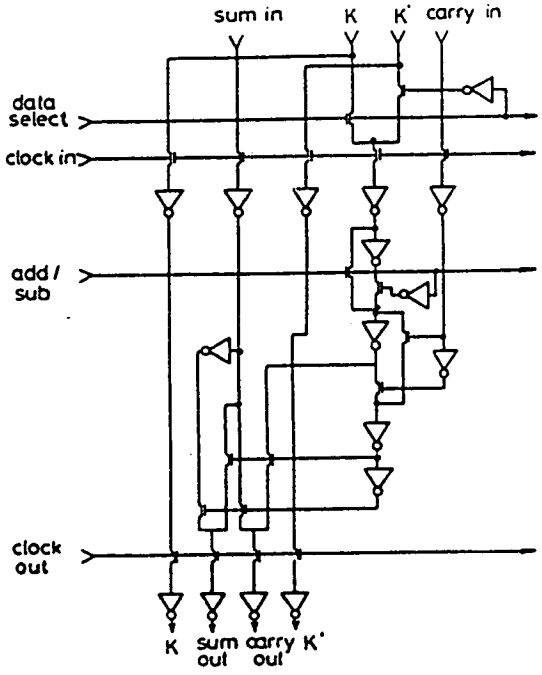


Figure 1: Basic Cell

Performance

The chip was fabricated on a 6 micron single polysilicon, single metal, n-channel enhancement/depletion MOS process and measured 6.7mm by 5.3 mm, containing around 8000 transistors. With a process yield of about 5% a number of devices were obtained and bonded up. Chip parameters are shown in Table 3. The chip performed functionally as predicted. Figure 3 shows the input and output logic signatures obtained by feeding in four cycles of data equal to the reciprocal of the square root of 2 (01011010) for all inputs, and then four cycles of zero's, followed by four cycles of the negative of the reciprocal of root 2 (10100110) for all inputs. Figure 2 shows the latency of the pipeline and its operation under dynamic conditions for this input sequence. This test was performed at 4 Mbytes/second.

Parameter	Value
Computation Cycle	250 nS
Data Rate	8 M Bytes/s
Word Length	8 Bits
Power Consumption (Average)	0.5 Watts
Package	40 Pin Dil

Table 3 Showing Chip Parameters

Conclusions

This paper has described a working monolithic FFT butterfly processor based on a distributed arithmetic algorithm for computation of the complex multiply. The resulting distributed arithmetic structure is area efficient and highly regular. Results for the prototype are encouraging and designs have been completed for a high performance 16-bit CMOS-SOS version. The NMOS chip does, however, serve to illustrate the modularity afforded by this algorithm for systems based on parallel data flow.

The 16-bit CMOS-SOS design features a two dimensional pipeline structure and incorporates extra delays in the horizontal and vertical bit streams - an approach which is more suitable for larger word lengths. The device should be capable of operating at data-rates of around 20 million (complex) multiply, sort and accumulate operations per second.

Acknowledgments

This work was carried out under a UK Science and Engineering Research Council Grant.

REFERENCES

1. Blauschild Robert A. : "An 8b 50ns Monolithic A/D Converter with Internal S/H." Digest of Technical Papers ISSCC 1983 pp.178-179

2. L. R. Rabiner and B. Gold : "Theory and Application of Digital Signal Processing". Prentice Hall (1975)

3. Stanley A. White : "A Simple FFT Butterfly Arithmetic Unit", IEEE Transactions on Circuits and Systems, Vol.CAS-28 No.4 April 1981 pp.352-355

4. I. R. Mactaggart and M. A. Jack : "Radix-2 FFT Butterfly Processor using Distributed Arithmetic" Electronics Letters Vol.19 No.2 January 1983 pp.43-46

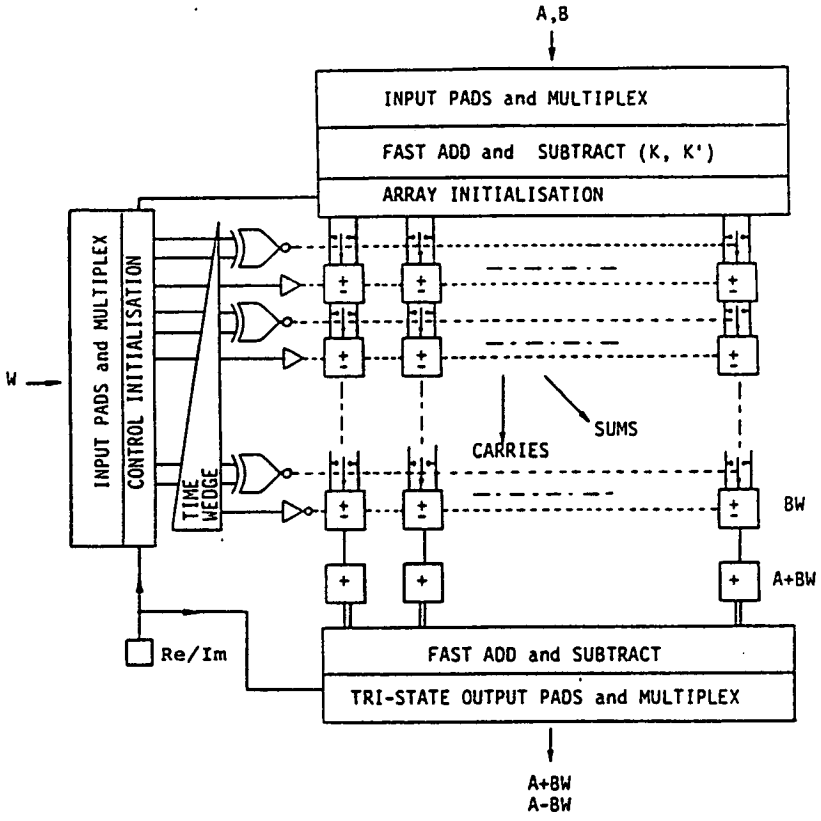


Figure 2: Floorplan

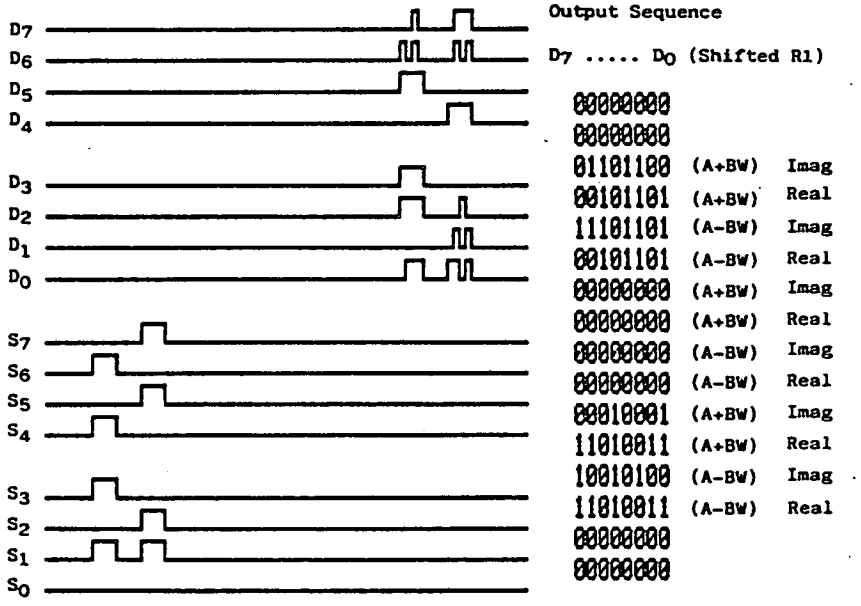


Figure 3: Typical Test Results

A Single Chip Radix-2 FFT Butterfly Architecture Using Parallel Data Distributed Arithmetic

I. ROSS MACTAGGART AND MERVYN A. JACK

Abstract—This paper describes how distributed arithmetic techniques can be applied in parallel-data arithmetic computations to achieve highly regular and efficient VLSI structures on silicon. Two individual arithmetic processor chips are described as examples of the technique.

The chips described, which are intended primarily for computation of the FFT Butterfly, each contain the functional equivalence of two parallel pipelined multipliers.

The first chip is an 8-bit prototype device which has been designed and fabricated on a standard 5 μm silicon gate n-channel MOS process. The second chip is a 16-bit CMOS-SOS design which uses a modified architecture to achieve higher clocking rates and improved versatility in systems use.

I. Introduction

REAL-TIME digital signal processing favors the use of very high-speed parallel data arithmetic operations. Distributed arithmetic techniques [1], [2] offer a means of mapping parallel data systems onto silicon with a high degree of regularity and efficiency. The specific structures considered here are two versions of a Radix-2 Butterfly processor for computation of the fast Fourier transform (FFT) algorithm [3], using distributed arithmetic.

The FFT algorithm is introduced and discussed briefly to highlight the Butterfly processing requirements and to indicate how distributed arithmetic approaches can be used in this processing task.

The paper includes a detailed discussion of the distributed arithmetic reformulation of the FFT Butterfly to show in detail how the silicon floorplan for the Butterfly processor can be derived.

Using an available, in-house NMOS process, a prototype 8-bit processor has been realized to validate the distributed arithmetic architecture. Details of this design are presented and test results together with performance data for this chip are discussed. A similar, but much more powerful 16-bit CMOS-SOS design with modified architecture using a commercially available process will also be described.

Manuscript received October 1, 1983; revised December 21, 1983. This work was supported by a Science and Engineering Research Council Grant.

The authors are with the Department of Electrical Engineering, University of Edinburgh, Edinburgh EH9 3JL, Scotland.

II. The FFT Algorithm

Of the several important FFT algorithms which have been developed for efficient computation of the discrete Fourier transform (DFT) [3], the most widely used is the Radix-2 decimation-in-time FFT [3], [4], where the transform length (N) may be any positive integer power of 2. A symbolic representation of this algorithm is shown in Fig. 1 for $N = 8$. Here, the time-domain sequence (x_n) is converted to the frequency domain sequence (X_n) by means of 12 identical processing nodes, each of which is known as a Butterfly. Each Butterfly processing node consists of a two-point DFT (vector add and subtract), symbolized by the circle in Fig. 1, with a vector rotation requirement (multiplication by a unit vector) on one of the inputs, symbolized by the arrow in Fig. 1. It is this vector rotation requirement which dominates any silicon implementation of the Butterfly since this entails a complex multiplication for each Butterfly operation.

III. Distributed Arithmetic Concepts

Complex multiplication involves four real multiplications, plus an addition and subtraction as shown in Fig. 2(a), to implement the equations

$$\mathrm{Re}\{Z\} = \mathrm{Re}\{B\}\cdot\mathrm{Re}\{W\} - \mathrm{Im}\{B\}\cdot\mathrm{Im}\{W\} \quad (1)$$

$$\mathrm{Im}\{Z\} = \mathrm{Re}\{B\}\cdot\mathrm{Im}\{W\} + \mathrm{Re}\{W\}\cdot\mathrm{Im}\{B\}. \quad (2)$$

It is clear, from Fig. 2(a), that the two multiplier structure used to form $\mathrm{Re}\{Z\}$ is essentially the same as the two multiplier structure used to form $\mathrm{Im}\{Z\}$, differing only in an add and subtract. This two multiplier structure might therefore be considered to be a suitable candidate for a VLSI implementation of the complex multiply requirement of the Butterfly. In the case of parallel arithmetic, however, this general structure does not map onto silicon very efficiently due to problems arising from bus interconnections and irregular multiplier structures when special multiply algorithms, such as Booth's [5], are used. For this reason, as well as for yield considerations, current parallel-data Butterfly devices use a single, multiplexed, parallel multiplier.

This paper shows how it is possible to replace the two parallel multiplier structure in Fig. 2(a) with a single dis-

0018-9200/84/0600-0368$01.00

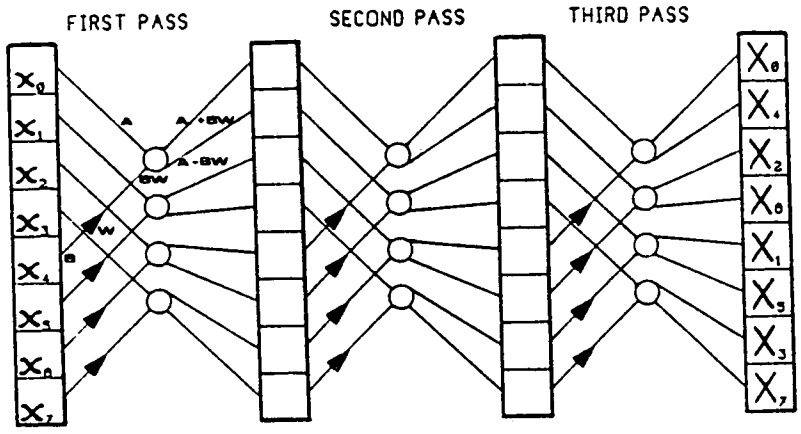


Fig. 1. 8-point FFT.

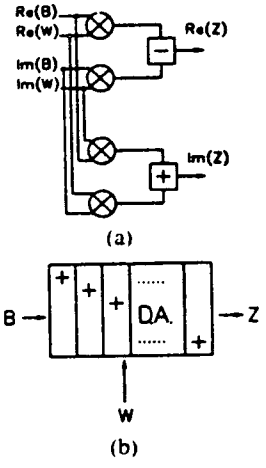


Fig. 2. Distributed arithmetic concept.

tributed arithmetic array which is regular and maps onto silicon efficiently. This is achieved in distributed arithmetic by bringing forward the final add and subtract in the complex multiply structure of Fig. 2(a) to the level of multiplier partial product formation, in order to form new unique arithmetically merged partial products which can be stored in temporary data registers. This allows the formation of real and imaginary complex outputs (Z) by performing a data-select and accumulate operation on these new merged partial products. The importance of this reformulation is that the resulting structure [Fig. 2(b)] involves only a single accumulator and therefore allows a highly regular VLSI structure. The distributed arithmetic approach [1] is generally useful when the products of more than one multiplier are subsequently combined in other arithmetic operations such as add or subtract to form a single output. In the case of the complex multiply, as described in (1) and (2), the real output requires two individual multipliers, each with two possible partial products. The same is true for the imaginary output. The two shift and add multipliers thus present four possible combinations of partial products, corresponding to the four possible combinations of the two real and imaginary coefficient bits being considered. It is the number of combinations of partial products that is important, as this determines the number of arithmetically merged distributed arithmetic partial products that will need to be stored. It will be shown later, however, that only two merged partial products are nontrivial in this case, and actually need to be stored.

Let us assume that N-bit, fixed point, two's complement arithmetic is used so that $\mathrm{Re}\{W\}$ and $\mathrm{Im}\{W\}$ might be described as

$$\mathrm{Re}\{W\} = -W_{RO} + \sum_{n=1}^{N-1} W_{Rn}\cdot 2^{-n} \tag{3}$$

$$\mathrm{Im}\{W\} = -W_{IO} + \sum_{n=1}^{N-1} W_{In}\cdot 2^{-n}. \tag{4}$$

This allows equation (1) for $\mathrm{Re}\{Z\}$ to be expressed as

$$\mathrm{Re}\{Z\} = \left[-W_{RO} + \sum_{n=1}^{N-1} W_{Rn}\cdot 2^{-n}\right]\cdot \mathrm{Re}\{B\} - \left[-W_{IO} + \sum_{n=1}^{N-1} W_{In}\cdot 2^{-n}\right]\cdot \mathrm{Im}\{B\}. \tag{5}$$

Combining the separate summations into one summation and decoding all possible combinations of the real and imaginary w bits to select these new merged partial products gives

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & \overline{W}_{RO}\cdot\overline{W}_{IO}(0) \\ & + \overline{W}_{RO}\cdot W_{IO}(\mathrm{Im}\{B\}) \\ & + W_{RO}\cdot\overline{W}_{IO}(-\mathrm{Re}\{B\}) \\ & + W_{RO}\cdot W_{IO}(-\mathrm{Re}\{B\}+\mathrm{Im}\{B\}) \\ & + \sum_{n=1}^{N-1}\left[\overline{W}_{Rn}\cdot\overline{W}_{In}(0)\right. \\ & + \overline{W}_{Rn}\cdot W_{In}(-\mathrm{Im}\{B\}) \\ & + W_{Rn}\cdot\overline{W}_{In}(\mathrm{Re}\{B\}) \\ & \left. + W_{Rn}\cdot W_{In}(\mathrm{Re}\{B\}-\mathrm{Im}\{B\})\right]\cdot 2^{-n}.\end{aligned} \tag{6}$$

Equation (6) shows how $\mathrm{Re}\{Z\}$ can be formed in a single accumulator by selecting one of four merged partial products. This was not the chosen solution, however, as only two are actually required if we define

$$K = (\mathrm{Re}\{B\}+\mathrm{Im}\{B\})/2 \quad \text{and}$$

$$K' = (\mathrm{Re}\{B\}-\mathrm{Im}\{B\})/2. \tag{7}$$

Replacing the $\mathrm{Re}\{B\}$ and $\mathrm{Im}\{B\}$ terms in (6) by the K and K' terms shown in (7) (and Table I), yields (8), which

TABLE I
PERFORMANCE

PARAMETER	NMOS	CMOS-SOS *
CLOCK CYCLE	250 nS	25nS
DATA RATE	8 M Wds/s	40 M Wds/s
WORD LENGTH	8 Bits	16 Bits
POWER	0.5 Watts	0.25 Watts
PACKAGE	40 Pin Dil	64 Pin Dil

* Available for testing December '83

can be further simplified to give (9):

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & \overline{W}_{RO}\cdot\overline{W}_{IO}(-K'+K') \\ & + \overline{W}_{RO}W_{IO}(-K'+K) \\ & + W_{RO}\overline{W}_{IO}(-K'-K) \\ & + W_{RO}W_{IO}(-K'-K') \\ & + \sum_{n=1}^{N-1}\big[\overline{W}_{Rn}\cdot\overline{W}_{In}(K'-K') \\ & \quad \overline{W}_{Rn}\cdot W_{In}(K'-K) \\ & \quad W_{Rn}\cdot\overline{W}_{In}(K'+K) \\ & \quad W_{Rn}\cdot W_{In}(K'+K')\big]\cdot 2^{-n} \qquad (8)\end{aligned}$$

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & -K'\cdot 2^{-(N-1)} + \overline{W}_{RO}\cdot\overline{W}_{IO}(+K') \\ & + \overline{W}_{RO}\cdot W_{IO}(+K) \\ & + W_{RO}\cdot\overline{W}_{IO}(-K) \\ & + W_{RO}\cdot W_{IO}(-K') \\ & + \sum_{n=1}^{N-1}\big[+\overline{W}_{Rn}\cdot\overline{W}_{In}(-K') \\ & \quad + \overline{W}_{Rn}\cdot W_{In}(-K) \\ & \quad + W_{Rn}\cdot\overline{W}_{In}(+K) \\ & \quad + W_{Rn}\cdot W_{In}(+K')\big]\cdot 2^{-n}. \qquad (9)\end{aligned}$$

Equation (9) shows how $\mathrm{Re}\{Z\}$ can be formed in a single accumulator by adding in or subtracting a selected K or K' as a function of the real and imaginary w bits. The selection of K or K' can be based on the Exclusive-OR of the real and imaginary w bits and the add/subtract logic can be derived from the appropriate w bit directly, as shown in (10):

$$\begin{aligned}\mathrm{Re}\{Z\} = {} & -K'\cdot 2^{-(N-1)} + \overline{W}_{RO}\big(W_{RO}\overline{\oplus}W_{IO}\big)(+K') \\ & + \overline{W}_{RO}(W_{RO}\oplus W_{IO})(+K) \\ & + W_{RO}(W_{RO}\oplus W_{IO})(-K) \\ & + W_{RO}\big(W_{RO}\overline{\oplus}W_{IO}\big)(-K') \\ & + \sum_{n=1}^{N-1}\big[+\overline{W}_{Rn}\big(W_{Rn}\overline{\oplus}W_{In}\big)(-K') \\ & \quad + \overline{W}_{Rn}(W_{Rn}\oplus W_{In})(-K) \\ & \quad + W_{Rn}(W_{Rn}\oplus W_{In})(+K) \\ & \quad + W_{Rn}\big(W_{Rn}\overline{\oplus}W_{In}\big)(+K')\big]\cdot 2^{-n}. \qquad (10)\end{aligned}$$

The expression for $\mathrm{Im}\{Z\}$ can be obtained similarly, giving

$$\begin{aligned}\mathrm{Im}\{Z\} = {} & -K\cdot 2^{-(N-1)} + \overline{W}_{IO}(W_{IO}\oplus W_{RO})(+K') \\ & + W_{IO}\big(W_{IO}\overline{\oplus}W_{RO}\big)(-K) \\ & + \overline{W}_{IO}\big(W_{IO}\overline{\oplus}W_{RO}\big)(+K) \\ & + W_{IO}(W_{IO}\oplus W_{RO})(-K') \\ & + \sum_{n=1}^{N-1}\big[+\overline{W}_{In}(W_{In}\oplus W_{Rn})(-K') \\ & \quad + W_{In}\big(W_{In}\overline{\oplus}W_{Rn}\big)(+K) \\ & \quad + \overline{W}_{In}\big(W_{In}\overline{\oplus}W_{Rn}\big)(-K) \\ & \quad + W_{In}(W_{In}\oplus W_{Rn})(+K')\big]\cdot 2^{-N}. \qquad (11)\end{aligned}$$

Table I depicts this algorithm for nonsign bits only, alongside the conventional arithmetic approach using shift and add multipliers. This table serves to illustrate how the individual merged partial products in the distributed arithmetic approach are related to the individual partial products in the conventional shift-and-add multiplier scheme. For example, in the $\mathrm{Re}\{Z\}$ formation columns (1–4) of Table I, row 3 shows how $\mathrm{Re}\{B\}$ can be expressed as $K'+K$, with a W independent K' term. In the same columns, row 4 shows how $\mathrm{Re}\{B\}-\mathrm{Im}\{B\}$ can be expressed as $K'+K'$.

In the shift and add multiplier, the final product is formed by the successive accumulation of partial products which are formed by the logical "AND" of the data word (B) with successive coefficient bits (W) at various levels of significance which are all powers of 2. The partial products for the four multipliers in the conventional arithmetic case are shown in columns 1,2; 5,6 in the table as a function of the individual bits of W. Table I shows how the data word is added in, only if the coefficient bit (W) is a "1." However, in the conventional approach, the final subtract (for $\mathrm{Re}\{Z\}$) and add (for $\mathrm{Im}\{Z\}$) is not considered until final product formation in the individual multipliers. Table I shows how the final add and subtract operation can be brought forward to the level of partial product formation so as to form four new merged partial products. Thus, individual multiplier partial products in columns 1 and 2 are now considered to be combined arithmetically to form a single column containing merged partial products for $\mathrm{Re}\{Z\}$. Similarly, columns 5 and 6 are now considered to be merged to form a single column from which $\mathrm{Im}\{Z\}$ can be formed directly. Table I goes on to show how these merged multiplier partial products can be replaced with the expressions involving K and K' (7) in columns 3,4,7,8. The K' term in column 3 for $\mathrm{Re}\{Z\}$ and the K term in column 7 for $\mathrm{Im}\{Z\}$ are both independent of the W coefficient bits. This means that these columns do not need to be included in the main accumulation process used to form real and imaginary Z. Instead, they can be accounted for during array initialization. The table shows how, by the W-controlled selection of $+K, -K, +K', -K'$ [+/−(K

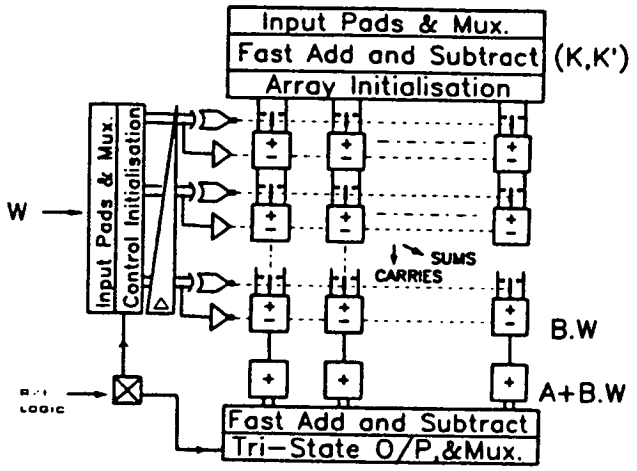


Fig. 3. Floorplan of NMOS chip.

or K')] (as shown in columns 4 for Re$\{Z\}$ and column 8 for Im$\{Z\}$), the complex product Re$\{Z\}$ and similarly Im$\{Z\}$ can be formed in a single data-select and accumulate structure. It can be seen how an Exclusive-OR/NOR type relation of the W bits can be used to select either K or K' and how the real W bit (imaginary W bit) can determine whether this selected K or K' is added or subtracted for Re$\{Z\}$ (Im$\{Z\}$).

IV. Distributed Arithmetic Architecture

Using these distributed arithmetic concepts, the two multiplier structure of Fig. 2(a) can now be replaced with a regular array of bit-level data-select/accumulate cells to form the floorplan of the 8-bit NMOS chip shown in Fig. 3. Data words (A, B) enter the chip and are demultiplexed into real and imaginary components. A fast adder and subtractor is used at this point to convert Re$\{B\}$ and Im$\{B\}$ to K and K', (7) which are then fed down to the first row of cells in the distributed arithmetic array, together with an array initialization word which comprises the very low significance $-K'$ or $-K$ present as the first term in (10) and (11) and a rounding word which is fixed. This rounding word was equal to the mean value of all the sums which had to be truncated in the array. At each cell, K or K' was selected under the control of the Exclusive-OR (Re$\{Z\}$) or Exclusive-NOR (Im$\{Z\}$) gates whose inputs were the real and imaginary W coefficient bits. Each cell was also fed an add/subtract control signal which was derived from the buffered real or imaginary W coefficient bits directly, as outlined in Table I. Only for the sign bits of W, when the bits have a negative significance is the add/subtract logic inverted so that the selected K or K' is added if the appropriate W bit is a "zero" instead of a "one"—as is the case with the nonsign bits of W.

In the NMOS chip, the CARRY data is fed forward along with the SUMS, so that it is necessary to assimilate SUM and CARRY data of equal significance at the output of the array. This was accomplished by means of a fast adder employing a precharged carry-chain. SUM and CARRY data in the array were latched, so it was necessary to skew the coefficient W input data to the control gates as depicted in Fig. 3. There was no need for skewing input or output data because of the use of the fast adder at the output. The nonrotated Butterfly input (A), which is shifted directly through the complex multiplier was finally added to the complex output (BW) to form the Butterfly output ($A + BW$). The other Butterfly output ($A - BW$) was formed as ($2A - (A + BW)$) as this avoided the need to feed BW forward, through the row of cells used to form ($A + BW$).

Fig. 4 shows the basic cell logic in detail. Invertor controlled data-select type Exclusive-OR gates were used in the carry-save adder, as this offers a good tradeoff in area–speed–power.

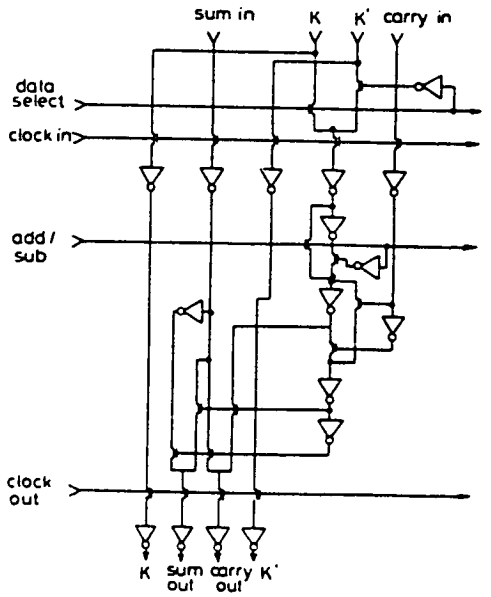


Fig. 4. Basic cell (NMOS).

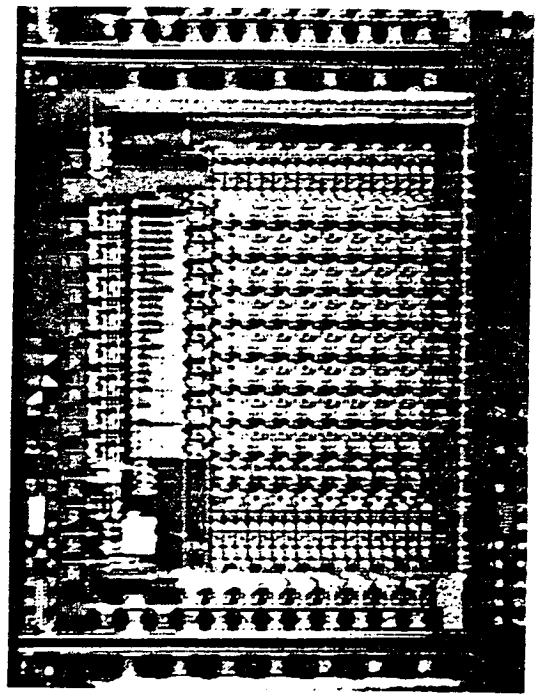


Fig. 5. NMOS chip photograph.

TABLE II
DISTRIBUTED ARITHMETIC ALGORITHM (COMPUTERS $Z = B \cdot W$)

W		REAL(Z)				IMAG(Z)			
Re	Im	1	2	3	4	5	6	7	8
0	0	0-0			=K'-K'	0+0			=K-K
0	1	0-Im(B)=K'-K				Re(B)+0			=K+K'
1	0	Re(B)-0			=K'+K	0+Im(B)=K-K'			
1	1	Re(B)-Im(B)=K'+K'				Re(B)+Im(B)=K'+K			

(Bit-Level)

Where $K=(Re(B)+Im(B))/2$ and $K'=(Re(B)-Im(B))/2$

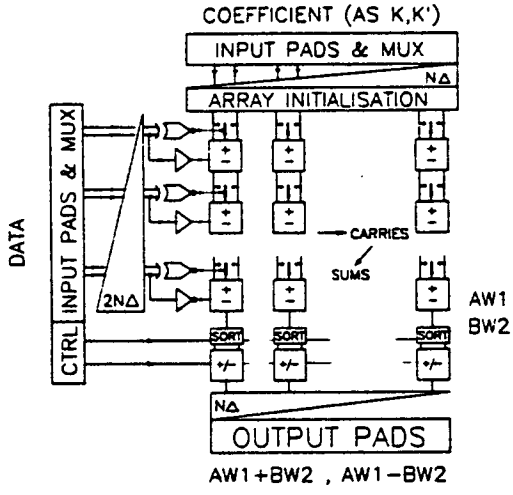


Fig. 6. Floorplan of SOS chip.

V. Details and Performance of NMOS Processor

The NMOS prototype chip (Fig. 5) contains around 8000 transistors and measures about 5.3×6.7 mm. The process used was a standard 5 μm feature size, single polysilicon, single metal n-channel MOS process using depletion mode load devices. Table II shows the measured performance of the 8-bit NMOS processor. The device clocked at 4 MHz corresponding to a data rate of 8 megabytes/s, which was slightly slower than expected owing to the use of a clock input pad which was limiting internal clock risetimes.

VI. CMOS-SOS 16-Bit Processor

In general, 8-bit word lengths are not adequate to cover most FFT application areas, such as radar signal processing, where 12–16-bit accuracy is typically required and for these reasons a 16-bit processor design which was similarly based on the distributed arithmetic complex multiply algorithm was undertaken. The floorplan for this device is shown in Fig. 6. For larger word lengths it is desirable to pipeline the distributed arithmetic in two dimensions so as to eliminate the fast add requirement. Extra latches (delays) must then be inserted into the basic cell.

CMOS Processor Pipeline

In the CMOS distributed arithmetic processor, (coefficient) data entering from the top is skewed, with the nth input bit receiving a delay of n, going from least to most significant bits. This allows the control and carries in each 17 cell row to be latched (extra cell per row for two's complement operation). The effect of this pipelining scheme is to produce a computation front which moves down through the array at an angle of 45°. This skewed computation front now means that data entering the vertical data port associated with the Exclusive-OR control gates, needs to be skewed by $2n$ delays for the nth input bit, moving from least significant to most significant bits, in order that control signals will match up correctly with data in the array. Fig. 7 shows the basic cell used in the 16-bit CMOS-SOS processor chip which results in a completely systolic architecture [6]. This cell feeds the CARRY right and the SUM is fed down and left to scale down the result of each cell by 2. The control passes from left to right at the same rate as the carries. The distributed arithmetic coefficients, K and K' needed to be delayed by two clock cycles in each 2-D pipelined cell because of the 45° skew on the computation front. As the same K and K' needs to be made available for both real and imaginary computations, this delay was implemented in a single shift register, clocked at half the main clock rate. This was done to save chip area, with the only condition that outputs would have to alternate between real and imaginary. The maximum clock rate of the chip is determined primarily by the time to produce a carry-out from the basic cell. It was stated that the vertical delay through this cell is equal to two clock cycles. This gives the array a latency of the order of $2n$ where n is the word length; however, the time-wedge used at the input to the array and the output of the array to skew and deskew data increases the latency of the chip by another n resulting in a total latency of around $3n$.

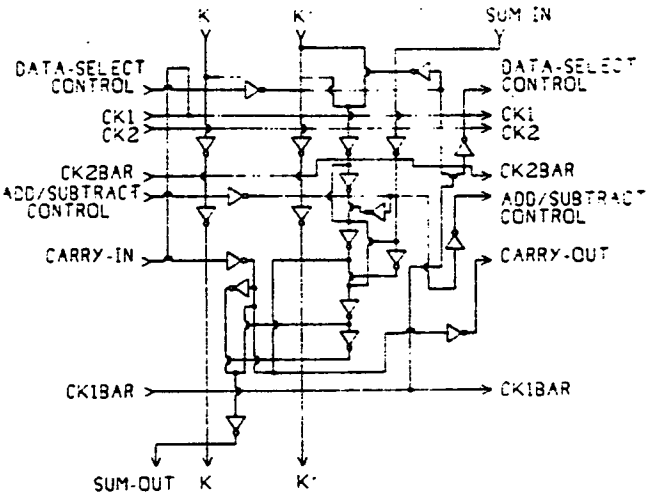


Fig. 7. Basic cell (SOS).

CMOS Architectural Modifications

The CMOS-SOS design contains some other significant architectural modifications. In the NMOS chip, data enters at the top of the chip, and was converted to the form of K and K' as defined in (7). In the CMOS-SOS design, the coefficient enters the top data port in the form of K and K'. The coefficient can therefore be stored in this form and

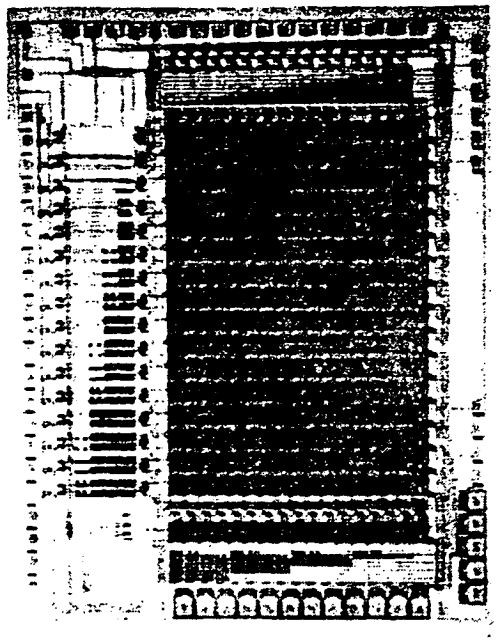


Fig. 8. SOS chip layout.

is not actively computed on the chip. This further lowered the power needed to compute the complex multiply. Unlike the NMOS chip, however, all data in the CMOS-SOS design passes through the complex multiplier. This has several advantages which are:

1) simple time-domain windowing on the first pass if required;

2) lower Butterfly noise caused by amplitude errors in W coefficient; and

3) easier system design with fewer components.

Further, the CMOS-SOS chip can compute a two-point nontrivial DFT, allowing larger DFT's to be built up using a slower external accumulator.

VII. Details and Expected Performance of CMOS-SOS Device

The CMOS-SOS device (Fig. 8) measures 7×8 mm and contains around 30 000 transistors. The device was designed using 4 μm feature CMOS-SOS design rules. This device uses an external clock generator to allow the highest possible clock rates to be achieved. Table II shows the expected performance of the CMOS-SOS device in comparison to the measured performance for the NMOS prototype.

VIII. Conclusions

Two LSI/VLSI chips which use distributed arithmetic to compute the arithmetic requirements of the Radix-2 FFT Butterfly have been described. Each of these devices has the throughput equivalence of two parallel multipliers, allowing very high bandwidths.

Distributed arithmetic offers a highly regular design approach in parallel data systems and also offers lower power consumptions than is possible using conventional arithmetic.

These techniques are thus highly suited to parallel data arithmetic, where an irregular structure can be replaced with a highly regular and compact array which offers a high degree of algorithmic efficiency.

Acknowledgment

The authors would like to thank the Edinburgh Microfabrication Facility for silicon processing and the General Electric Company for providing design rules for their high performance SOS process.

References

[1] S. A. White, "A simple FFT butterfly arithmetic unit," *IEEE Trans. Circuits Syst.*, vol. CAS-28, Apr. 1981.

[2] I. R. Mactaggart and M. A. Jack, "Radix-2 butterfly processor using distributed arithmetic," *Electron. Lett.*, vol. 19, pp. 43–44, Jan. 1983.

[3] L. R. Rabiner and B. I. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[4] E. O. Brigham, *The Fast Fourier Transform*, 1974.

[5] D. A. Booth, "A signed binary multiplication technique," *Q.J. Mech. Appl. Maths.*, vol. 4, pp. 236–240, 1951 (Oxford Univ. Press, no. 4, Apr. 1951).

[6] H. T. Kung and C. Leiserson, "Systolic arrays for (VLSI)," in *Introduction to VLSI Systems*, Mead and Conways, Eds. Reading, MA: Addison-Wesley, 1980.

I. Ross Mactaggart received the B.Sc. (Hons.) degree in chemical physics in 1980 and the M.Sc. degree in the design and manufacture of microelectronic systems, both from Edinburgh University.

He is currently a member of the Integrated Systems Group at Edinburgh University where he is involved in semi-custom and full-custom design and research activities.

Mervyn A. Jack was born in Edinburgh, Scotland, on June 20, 1949. He received the B.Sc. degree in electronic engineering and the M.Sc. degree in digital techniques from the Heriot-Watt University, Edinburgh, in 1971 and 1975, respectively, and the Ph.D. degree from the University of Edinburgh in 1978.

From 1971 to 1975 he worked as a project engineer with Microwave and Electronic Systems, Ltd., Edinburgh, where he was responsible for the design and development of security systems based on passive infrared and microwave Doppler intruder detectors. In 1975 he was appointed to a Research Fellowship at the University of Edinburgh to study the design and application of Fourier transform processors based on surface acoustic wave and charge coupled devices. In 1979 he was appointed to a lectureship in the Department of Electrical Engineering at Edinburgh University.

Dr. Jack is a member of the Institution of Electrical Engineers.