Department of Electrical Engineering and Electronics

University of Manchester
Institute of Science and Technology

A NIBBLE-SERIAL PROGRAMMABLE DIGITAL SIGNAL PROCESSOR FOR VLSI IMPLEMENTATION

by

Robert A. Cottrell

A thesis submitted to the
UNIVERSITY OF MANCHESTER
for the degree of
DOCTOR OF PHILOSOPHY
in the Faculty of Technology

May 1987

ABSTRACT

This thesis describes the design of a programmable Signal Processing Element (SPE). Each SPE is 4 bits (nibble) wide. The word length is programmable, and arithmetic is performed in a nibble-serial manner, i.e. in parallel over 4 bits and then serially, four bits at a time, to provide the desired word length. The SPE architecture is specifically tailored to the implementation of difference equation type calculations, such as are used in digital filters. The SPE is intended for VLSI implementation, in which case a large number of SPEs could be fabricated on a single chip or wafer. The thesis describes the development of the SPE architecture, its simulation at a register transfer level, and the detailed NMOS design of certain sections of a single SPE. This NMOS design permits assessment of the silicon area occupied by these sections, and estimation of their performance. A particular aim of the work was to provide an efficient means of using silicon to provide the processing power required by digital signal processing systems, while maintaining the flexibility of programmable systems. This was to be done by allowing programmable word length, and by restricting the class of applications. However, it transpires that the cost of providing programmable word length is rather high and significantly reduces the efficiency of SPE based systems. Despite this, it is undoubted that SPEs provide a means of implementing programmable digital signal processing systems. Future work on programmable systems is likely to concentrate on architectures like the Texas Instruments TMS320 or the Inmos Transputer. Silicon compilers could well provide an easy means of providing custom silicon implementations at much reduced design effort.

ABSTRACT

This thesis describes the design of a programmable Signal Processing Element (SPE). Each SPE is 4 bits (nibble) wide. The word length is programmable, and arithmetic is performed in a nibble-serial manner, i.e. in parallel over 4 bits and then serially, four bits at a time, to provide the desired word length. The SPE architecture is specifically tailored to the implementation of difference equation type calculations, such as are used in digital filters. The SPE is intended for VLSI implementation, in which case a large number of SPEs could be fabricated on a single chip or wafer. The thesis describes the development of the SPE architecture, its simulation at a register transfer level, and the detailed NMOS design of certain sections of a single SPE. This NMOS design permits assessment of the silicon area occupied by these sections, and estimation of their performance. A particular aim of the work was to provide an efficient means of using silicon to provide the processing power required by digital signal processing systems, while maintaining the flexibility of programmable systems. This was to be done by allowing programmable word length, and by restricting the class of applications. However, it transpires that the cost of providing programmable word length is rather high and significantly reduces the efficiency of SPE based systems. Despite this, it is undoubted that SPEs provide a means of implementing programmable digital signal processing systems. Future work on programmable systems is likely to concentrate on architectures like the Texas Instruments TMS320 or the Inmos Transputer. Silicon compilers could well provide an easy means of providing custom silicon implementations at much reduced design effort.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to the Professors for making available the facilities of the department, and to Professor E.T. Powner for his supervision and encouragement throughout the work. Thanks are also due to all members of the Digital Processes Group and the Microprocessor Engineering Unit for their advice and assistance.

DECLARATION

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

LIST OF FIGURES

LIST OF FIGURES

PREFACE

One of the main challenges posed by Very Large Scale Integration (VLSI) is how best to make use of its potential. It is not difficult to envisage applications which could benefit from this enormous potential, but it is not always so easy to devise means of harnessing the raw processing power in any particular application.

Digital signal processing promises to be a very fruitful area for the exploitation of VLSI. Processing signals digitally in real time demands vast raw processing power. Moreover, many of the algorithms employed show a high degree of concurrency, making them strong contenders for the successful application of parallel processing techniques.

This thesis is concerned with the means of harnessing the potential processing power afforded by VLSI for digital signal processing applications. The merits and failings of both custom designed hardware and programmable systems are recognised, but it is on programmable systems which this thesis concentrates its attention. In the VLSI era, custom hardware will increasingly imply the use of Application Specific Integrated Circuits (ASICs), and this is also a key area for research in relation to digital signal processing applications.

The first chapter of the thesis constitutes a survey of the techniques of digital signal processing and their applications. This is followed in chapter two by a survey of past attempts to design pro-

grammable digital signal processors.

Chapter three sets out and justifies the aims of the project, namely to design a programmable processor, specifically tailored to the implementation of difference equation calculations, and suitable for connecting in a pipeline to exploit concurrency. Several processors which were available at the start of the project are considered for suitability, but all are rejected for a variety of reasons. The application area was deliberately chosen to be narrow to give the best chance of obtaining an efficient solution.

The development of the processor architecture is described in chapter four. A Signal Processing Element (SPE) is developed, which can be used to produce processors of configurable wordlength. This is achieved by using synchronous carry in a nibble-serial fashion. It is envisaged that many such SPEs would be integrated on a single VLSI chip, together with all necessary memory. The feasibility of an SPE based approach is demonstrated by simulation at the register transfer level, as described in chapter five. This includes the simulation of a 2nd order low-pass filter.

Certain key elements of an SPE were designed in detail for implementation in NMOS technology. This is described in chapter six. Two test chips were fabricated and tested. These designs provide important information on the silicon area occupied by various parts of the SPE, and circuit level simulations yield information on likely performance.

The final chapter is concerned with evaluation of the project and suggestions for further research. Although the achievements of the project are recognised, it is admitted that the approach adopted is probably not the most promising. Further research is suggested both in programmable systems utilising multiple processors, and in Application Specific Integrated Circuits (ASICs) for custom hardware implementation of digital signal processing systems.

CHAPTER ONE

INTRODUCTION TO DIGITAL SIGNAL PROCESSING

There are many advantages in using the techniques of digital electronics in the processing of signals. Digital circuits are not subject to the effects of component drift as are their analogue counterparts. Their performance can be accurately predicted and will not vary with time or temperature. Analogue processing elements always introduce unwanted noise; the noise introduced by digital processing can be reduced as far as is desired simply by using higher precision arithmetic. Analogue recordings deteriorate with time, and each re-recording introduces noise and distortion. Digital recordings do not deteriorate, or at least signals can be perfectly restored with no noise or distortion being introduced in the process. These advantages have been well understood for some years now, as have the basic algorithms for digital signal processing. It is however only recently, with the advent of Large Scale and Very Large Scale Integration (LSI and VLSI), that electronic components have become available at sufficiently low cost and in sufficiently large quantities to permit digital signal processing techniques to be adopted in all but the most sophisticated applications.

For something which has as many advantages as digital signal processing, there has to be a major disadvantage. In this case, it is the sheer computational power required. Digital signal processing systems operate on sampled data in discrete time. Sampled data signals contain all the information of the continuous time signals which they represent, provided that the sampling frequency is greater than

twice the maximum frequency component of the signal [32]. In a system such as a digital filter, a particular set of calculations must be performed for each and every sample. Thus the computational power required for a particular application is determined by the signal bandwidth and hence the sampling frequency, and the complexity of the calculation to be performed. For example, a simple second order filter section requires four multiplications and four additions. To perform this operation on a high fidelity music signal with a bandwidth of 15kHz requires one multiplication and one addition to be performed 120 000 times per second, or within 8 microsecond. To perform the same operation on a television signal with a bandwidth of 5MHz requires the same operations to be performed within 25 nanosecond. As this operation is one of the simplest which can be performed, it is evident that the computational power required can be fearsome.

1.1 ALGORITHMS FOR DIGITAL SIGNAL PROCESSING

1.1.1 Digital Filtering

The simplest digital signal processing system is the digital filter. The aim of the digital filter is the same as that of its analogue counterpart, and hence there are low-pass, high-pass, band-pass filters etc. There are similar factors to consider when designing digital filters, such as stability and phase linearity.

Analogue filters are commonly represented in the complex frequency, s, or Laplace transform domain, whereas digital filters are represented in the Z-transform domain. Just as an analogue filter is described by the locations of poles and zeros in the s-plane, so a

digital filter is described by poles and zeros in the z-plane. There is, in fact, a direct mapping between the s-plane and the z-plane, thus a digital filter may be designed from its analogue equivalent. The reader is referred to an introductory text on digital signal processing, such as [11], for further details.

In the z-transform domain, a generalised digital filter is represented by the equation:

$$Y(z) = H(z).X(z)$$

where X(z) and Y(z) are the z-transforms of the input and output sequences, and H(z) is the z-transform of the impulse response: the transfer function of the filter. H(z) is expanded as follows:

$$H(z) = \frac{\sum_{k=0}^{M} a_k z^{-k}}{1 + \sum_{k=1}^{L} b_k z^{-k}}$$

The roots of this numerator and denominator are, respectively, the zeros and poles of H(z). Such a filter as this is represented in the time domain as

$$y_n = \sum_{k=0}^{M} a_k x_{n-k} - \sum_{k=1}^{L} b_k y_{n-k}$$

and can thus be realised as a difference equation.

The filter described here is known as a RECURSIVE or INFINITE IMPULSE RESPONSE (IIR) filter. It is recursive because previous values of the output sequence are used to determine the value of the next output sample. The impulse response is infinite, not because y tends to infinity at any time (this would be an unstable filter), but

because the impulse response does not become zero within a finite time. Thus the energy of the impulse response is infinite.

There is a second class of digital filters known as TRANSVERSAL or FINITE IMPULSE RESPONSE (FIR) filters. Such filters have no poles, and thus the transfer function H(z) is simply

$$H(z) = \sum_{k=0}^{M} a_k z^{-k}$$

and each output sample is determined as

$$y_n = \sum_{k=0}^{M} a_k x_{n-k}$$

Thus the impulse response is finite: it becomes zero after M samples. It can be seen that FIR filters represent the digital convolution of the input response with the impulse response sequence a_k, k=0,1,2,...,m.

Both IIR and FIR filters can be designed to meet almost any performance requirement. In general, FIR filters require more stages to achieve the same response. However, IIR filters need much more care to be taken to ensure their stability (FIR filters, possessing no feedback, cannot be unstable). Instabilities in IIR filters can arise not only from poles outside the unit circle in the z-plane, but also from roundoff noise due to the finite word length used for calculations. In general IIR filters need longer word lengths than FIR filters, both due to the stability problem and to the possibility for arithmetic errors to accumulate as the output is fed back to the input.

because the impulse response does not become zero within a finite time. Thus the energy of the impulse response is infinite.

There is a second class of digital filters known as TRANSVERSAL or FINITE IMPULSE RESPONSE (FIR) filters. Such filters have no poles, and thus the transfer function H(z) is simply

$$H(z) = \sum_{k=0}^{M} a_k z^{-k}$$

and each output sample is determined as

$$y_n = \sum_{k=0}^{M} a_k x_{n-k}$$

Thus the impulse response is finite: it becomes zero after M samples. It can be seen that FIR filters represent the digital convolution of the input response with the impulse response sequence a_k, k=0,1,2,...,m.

Both IIR and FIR filters can be designed to meet almost any performance requirement. In general, FIR filters require more stages to achieve the same response. However, IIR filters need much more care to be taken to ensure their stability (FIR filters, possessing no feedback, cannot be unstable). Instabilities in IIR filters can arise not only from poles outside the unit circle in the z-plane, but also from roundoff noise due to the finite word length used for calculations. In general IIR filters need longer word lengths than FIR filters, both due to the stability problem and to the possibility for arithmetic errors to accumulate as the output is fed back to the input.

It is extremely difficult to establish the stability of an IIR filter of greater than second order, i.e. with more than two poles. However this is no real limitation as more complex filters can be reduced to a number of second order sections, either connected in parallel or cascade [11 pp16-19].

FIR filters have a number of advantages, including the shorter word length requirement and their inherent stability. They are used in adaptive filters[33], where the filter coefficients are time-varying, dependent on some function of the input or output signal, as it is impossible to establish the stability of IIR filters with time-varying coefficients. In many cases, the adaptation algorithms tend to be simpler for FIR filters. The execution of FIR filters can be speeded up by using the Fast Fourier Transform algorithms in their implementation (see next section), although the reduced number of computations required is balanced by the increased complexity of data management.

1.1.2 The Discrete Fourier Transform and the FFT

Just as the power spectrum of an analogue signal can be represented by the Fourier transform of the signal, so there exists a similar algorithm, the Discrete Fourier Transform (DFT), for digital signals. The DFT of an N-point complex sequence (x_n), $n=0,1,2,\ldots,N-1$ is also an N-point complex sequence (A_p), $p=0,1,1,\ldots,N-1$ and is defined by the relation:

$$A_p = \sum_{n=0}^{N-1} x_n (i_N)^{pn}$$

where $W_N = e^{\frac{-2i\pi}{N}}$

The DFT can be used to determine the power spectrum of a signal [11 pp156-159]. As it is impractical to determine directly the DFT of a very long sequence, the practice is to form DFTs of short subsequences, modified by a window function to avoid discontinuities at the ends of the subsequences, and form averages of these DFTs.

As mentioned in the previous section, DFTs can be used to implement FIR filters [11 pp159-164]. This is because convolution in the time domain corresponds to multiplication in the frequency domain. Thus the DFT of the input sequence is formed, it is multiplied by the DFT of the impulse response, and the output sequence is reconstructed by an inverse DFT. The DFT of the impulse response is constant and need be calculated only once. The input signal is partitioned into segments for this process, and some corrections are required at the boundaries to obtain the correct result. It does not seem initially that this implementation should be faster then the direct implementation. However it proves to be considerably faster when the Fast Fourier Transform algorithm is used to implement the DFT

The existence of the Fast Fourier Transform (FFT) algorithms [12] is one of the main reasons for the great interest in the DFT today. In the direct implementation of the DFT, the number of computations grows as N^2, where N is the length of the sequence. The FFT algorithms form the DFT of a sequence from a combination of the DFTs of subsequences. By doing this recursively, the computation rate only grows as (N ln N). A simple description of the FFT can be found in [11 pp142-152]. The major disadvantage of the FFT is that in reduc-

ing the computation rate it increases the complexity of the data flow.

1.1.3 Homomorphic Filtering and the Cepstrum

In general, non-linear systems are difficult to analyse and apply. However, there is a class of non-linear systems which are useful, known as homomorphic systems. These obey an extension of the superposition principle for linear systems, described in [6 p172]. One such system makes use of the logarithm. If two signals are multiplied in the time domain, such as in amplitude modulation where the modulating signal is multiplied by the carrier, then taking the logarithm means that the signals are now added. Thus the signals can more easily be separated and processed independently. It is necessary to use the complex logarithm, as the input signal cannot be constrained to being positive; thus homomorphic processing has no equivalent in analogue terms. Exponentiation is required at the output to restore the signal.

Homomorphic processing can be usefully applied to convolved signals as well as multiplied signals. Here, the DFT is used to convert the convolved signal in the time domain into a multiplied signal in the frequency domain. The logarithm can then be taken to yield an additive system. If a DFT is once again applied, then the cepstrum, the spectrum of the log magnitude of the spectrum, is formed [34]. The cepstrum is once again in the time domain, and may be considered as a measure of the periodicity of the spectrum. It can thus be used to detect such things as echo or reverberation.

1.1.4 Adaptive Filtering

Adaptive filtering, as its name implies, is where the filter adapts itself to some function of its environment, usually to some function of the input or output signal. In general, some measure is made of how the output signal diverges from the desired output, and an attempt is made to alter the filter coefficients to produce a result nearer to that desired. In some ways, the term "adaptive" is a misnomer, as the system is wholly deterministic and the output can be expressed as a function of the input. It is, however, a good description of how the systems are implemented.

The problems with adaptive systems are in determining a suitable, simple error measure and an algorithm for estimating the best filter coefficients to minimise the error. This estimation is usually a recursive algorithm, and hence the computational power required can be enormous. Compromise is usually necessary in order to perform the computations in the available time.

1.2 APPLICATIONS OF DIGITAL SIGNAL PROCESSING

Digital signal processing began as an alternative to analogue approaches such as active filters. The digital approach provides absolute stability which is unavailable to the analogue designer due to component drift. Digital signal processing was particularly useful in low frequency systems, where the computational power required is not too great and analogue components are often large and cumbersome.

Digital signal processing has now matured into a subject in its own right; indeed there are digital processing functions which have

no obvious meaning in analogue terms. Digital signal processing has found application in many areas of engineering, which have widely varying requirements in terms of speed (due to the signal bandwidth) and computational complexity.

What follows is a very brief overview of some of the application areas where digital signal processing techniques have been successfully used. For a more detailed consideration, the reader is referred to [13].

1.2.1 Speech Processing

Speech processing includes a wide range of functions, such as speech recognition, speech synthesis and speech data compression. This last function is used for the efficient digital storage and transmission of speech.

All areas of speech processing depend on a model of the speech signal. The most popular model is that of a filter, representing the vocal tract, excited either by a periodic waveform (for voiced sounds) or by a noise source (for unvoiced sounds). Naturally, both the excitation and the filter must be time-varying.

The main problem in speech processing is the deconvolution of the speech signal into these two components. The two most popular techniques are to use homomorphic filtering to perform cepstral smoothing, or to use linear prediction to produce an all-pole model of the vocal tract filter. For the purpose of this thesis, it is most important to note that the former relies heavily on the FFT, whereas the latter uses matrix operations.

1.2.2 Telecommunications

Telecommunications is in many ways the home of signal processing, as it is wholly concerned with the transmission of signals. Digital filters can be used to implement the high order filters required to band-limit signals prior to multiplexing, and there are many cases where signals need to be shifted in frequency, modulated, demodulated etc. With the increasing use of digital coding for the transmission of signals, it makes sense to do much of this processing digitally too.

Tone detectors are an important part of modern telephone systems, and these are more and more being implemented digitally. The system must be able to recognise the control tones used, and to differentiate between these and speech signals.

One of the more complicated operations in telephony is the control of echo on long distance circuits, especially on international lines. The favoured approach today is to use echo cancellers which are adaptive filters. These can adapt to the properties of different lines, and to the variations with time on a single line. The main difficulty here is differentiating between echo signals and true outgoing signals.

1.2.3 Audio and Music Processing

This area is one of the richest for signal processing, with many varying requirements throughout. The recording and transmission of music signals is concerned with giving the listener subjectively the best sensation, and thus many of the operations performed must be developed with the principles of psychoacoustics in mind. At the

simplest level, this means taking account of the frequency range of human hearing. There is little virtue in the high fidelity reproduction of signals which the human ear cannot hear, unless it is intended to please the listener's dog or confuse a passing bat!

Digital techniques are being used more and more in the music industry, both for recording and processing. Since the advent of compact discs, digital recording, or at least playback, has been available in home systems. In analogue systems, each component degrades the signal and introduces noise. This need not be so with digital techniques, which can be essentially error-free, and added noise made negligible by selecting an adequate word length.

Electro-acoustic transducers: microphones, loudspeakers, etc. are a major cause of problems in audio systems. Signal processing techniques can be used to correct for any errors inherent in particular transducers, and also has a large part to play in research and development for new improved transducers

Modern recordings are typically made on multi track tapes, with one track for each instrument or group of instruments. This significantly reduces the amount of acoustic background noise. However, if the signals were simply mixed together, the result would be unbearable. At "mix-down", artificial reverberation has to be introduced to simulate the effect of the concert hall, and the spectrum of each signal must be corrected for the idiosyncrasies of the recording environment. Many other effects are available to the recording engineer. For example, a single instrument or voice can be made to sound like a chorus by introducing random delays and frequency shifts, etc. There is a great need for flexible, programmable sys-

tems in this application area to give the sound engineer as much control as possible over the final sound.

There are a host of other applications for signal processing in music. Companding, or dynamic range compression, is ideally suited to digital homomorphic filtering techniques. Many other processes are available, such as "ambience enhancement" for producing a pseudo-quadrophonic signal from a stereo signal, echo removal for removing tape "print-through" using cepstral techniques, and blind deconvolution techniques for restoring old recordings. The possibilities are limited only by the inventiveness of the mind.

1.2.4 Image Processing

There is always some doubt as to whether image processing falls within the area of signal processing. However it is clear that many of the algorithms and techniques of signal processing are used in image processing.

One important area within image processing is data rate compression. This is due to the enormous quantities of data concerned with images. A single picture of 512x512 pixels and 8 bits per pixel requires two megabits of storage. However the high spatial correlation of most images, and especially the inter-frame correlation of moving pictures, mean that data rates can often be reduced by the order of 30:1 to 50:1. Various techniques are used. One technique, similar to the linear predictive coding of speech, uses a statistical predictor to predict the next sample value. Thus only the difference from the expected value need be transmitted.

Image restoration and enhancement is another important area. Out-of-focus images can be restored, and some assistance is available here in that a mathematical model of out-of-focus photography can be constructed. Contrast and edge enhancement are also possible. Important techniques here are spatial filtering, analogous to ordinary digital filters but in two dimensions, and especially homomorphic techniques.

A final area of importance is the reconstruction of images from projections. This is of use in medicine, in head and body scanners, and in locating hidden metallurgical defects. The FFT is all-important here. The DFT is formed for each projection, and all the DFTs are put together to form what is essentially a 2-dimensional DFT of the image. It is necessary to perform interpolations to convert the polar arrangements of the data from the projections to the cartesian arrangement needed to perform the inverse transform to reconstruct the image.

1.2.5 Radar

Radar is an area of signal processing which is of particular importance to the military, but is also relevant to other applications such as weather forecasting. The main problem with radar is the high bandwidth of the signals involved. Nobody seriously wants to process the microwave radar signals themselves digitally, but the modulating signals have bandwidths in the region of 10 to 100 MHz. This puts considerable strain on the signal processing system, not least on the analogue-to-digital converters themselves. Radar systems begin with a signal generator, which may be digital. This signal is then modulated and transmitted. The received signal is

Image restoration and enhancement is another important area. Out-of-focus images can be restored, and some assistance is available here in that a mathematical model of out-of-focus photography can be constructed. Contrast and edge enhancement are also possible. Important techniques here are spatial filtering, analogous to ordinary digital filters but in two dimensions, and especially homomorphic techniques.

A final area of importance is the reconstruction of images from projections. This is of use in medicine, in head and body scanners, and in locating hidden metallurgical defects. The FFT is all-important here. The DFT is formed for each projection, and all the DFTs are put together to form what is essentially a 2-dimensional DFT of the image. It is necessary to perform interpolations to convert the polar arrangements of the data from the projections to the cartesian arrangement needed to perform the inverse transform to reconstruct the image.

1.2.5 Radar

Radar is an area of signal processing which is of particular importance to the military, but is also relevant to other applications such as weather forecasting. The main problem with radar is the high bandwidth of the signals involved. Nobody seriously wants to process the microwave radar signals themselves digitally, but the modulating signals have bandwidths in the region of 10 to 100 MHz. This puts considerable strain on the signal processing system, not least on the analogue-to-digital converters themselves. Radar systems begin with a signal generator, which may be digital. This signal is then modulated and transmitted. The received signal is

demodulated and passed through a matched filter to reduce noise. When implemented digitally, this is normally an FIR filter implemented by fast convolution using the FFT. Next, there is data rate reduction in the form of thresholding to decide which signals correspond to real targets. The threshold level can be varied to maintain a constant "false alarm" rate. On this reduced data, calculations of target metrics, such as position and velocity, are performed. The whole system is under the control of a host computer.

The data rates involved decrease further away from the actual radar signals, so digital techniques first found application here. After the introduction of a host computer, the next function to be "digitised" was the target metric generation. This is commonly a programmable system, as great benefit can be obtained from the flexibility this brings. More recently, the thresholding and matched filtering have come to be implemented digitally with the availability of higher speed and more dense digital logic.

1.2.6 Sonar

Sonar is to the ocean as radar is to the air, except that sonar is used not only to detect objects such as ships and submarines, but also to discover the nature of the sea bed. Sonar depends heavily on models of sound propagation in water; radar has no such problem as the propagation of microwaves in air is essentially rectilinear.

There are two basic types of sonar: active and passive. Active sonar is similar to radar in that pulses are transmitted and their reflections received and interpreted. Many of the operations, matched filtering, thresholding etc., are essentially the same as for

radar.

Passive sonar, as its name implies, consists only of listening, and attempts to locate objects from the sounds they themselves emit. It is thus very wide in scope, but two important techniques are worth mentioning. Spectral estimation using the FFT is very important, as are beamforming techniques using arrays of antennae (or a moving antenna) and adaptive filtering.

1.2.7 Geophysical Signal Processing

Geophysical, or seismic, processing is concerned with research into the structure of the earth. Its main applications are in predicting earthquakes and in the location of mineral deposits, in particular petroleum.

In common with many areas of science, geophysics depends on models of the earth's crust. The most popular model is of well-defined strata, or layers, and that signals are reflected at the boundaries of these strata. In some places, the structure is quite simple and the detection of the stratum boundaries is quite straightforward. In other places, the structure is more complex and multiple reflections cause difficulty in interpreting the signals. It is in this area that digital signal processing has proved useful in geophysics. Using statistical methods on large amounts of data, it is possible to deconvolve the signals, i.e. to separate the effects of the different boundaries and so determine the structure of the ground. Many of the algorithms are quite complex, making significant use of matrix techniques. Many systems today use not only simple vertical reflections, but use arrays of detectors, when 2-dimensional filter-

ing becomes important.

1.2.8 Biomedical Signal Processing

One of the main uses for signal processing in medicine is computer aided tomography, or the reconstruction of 2- and 3-dimensional images from x-ray and ultrasonic projections of the body. This is discussed above under Image Processing. Image enhancement techniques can also be applied to improve ordinary x-ray pictures to aid diagnosis. Indeed image processing systems can be programmed to enhance features associated with a particular ailment.

Signal processing is also used in the analysis of EEG (electro-encephalogram) and ECG (electro-cardiogram) signals. Here, the main concern is with spectral analysis and thus the FFT is all important.

CHAPTER TWO

IMPLEMENTATION OF DIGITAL SIGNAL PROCESSORS

Implementations of digital signal processors can be divided into two major classifications: hardwired systems using specially designed hardware, and programmable systems using general purpose hardware. Many systems, of course, do not fall entirely into one class or the other. Such systems have some specially designed parts under the control of a computer.

Hardwired systems always provide the most efficient means of using a particular technology. The highest performance can be obtained for the lowest component cost. However, design costs can be astronomical, as can the cost of any modifications required at a later date. Thus hardwired implementations are used in very high performance systems, where programmable systems simply cannot provide the required processing power, and in high volume applications, where the design cost can be amortised over many units.

Programmable systems do provide many advantages. System design now becomes essentially software design, which can be done much more quickly and cheaply than hardware design, and is much more easily verified. Flexibility is provided too; if it becomes necessary to alter the specification at a later date, this can often be done simply by re-writing part of the program. However, the component cost of a programmable system is likely to be much higher that for a hardwired system. Thus programmable systems find application mainly in relatively low volume or low performance systems. They are particularly useful in research laboratories, in that algorithms can be

developed, tested and modified with great ease.

2.1 COMPUTER ARCHITECTURES FOR SIGNAL PROCESSING

The advantages afforded by programmable systems in terms of flexibility and ease of design have led to many attempts to improve the cost-performance ratio of general purpose computers. Almost all general purpose computers today are based on the architecture proposed by John von Neumann [3], although he was designing a special purpose machine for the calculation of artillery trajectories! The essentials of this architecture are a unified memory containing both instructions and data, communicating with a Central Processing Unit (CPU) via data and address buses. Instructions are fetched from memory and executed in strict sequence. It is this sequential fetch-execute operation which gives the machine its elegant simplicity and general applicability, but it is also its most severe limitation. Processing power is limited by the speed of the fetch-execute cycle, and there is no mechanism for any concurrency whatsoever. A number of ameliorative techniques have been developed over the years, such as instruction queues, cache memories, interrupts and direct memory access. Such techniques have been successful in providing an increase in speed greater than that due simply to technological advance, albeit at the expense of additional circuit complexity. However, although they all serve to speed up the fetch-execute cycle, they do not provide any real concurrency of operation.

Over the last twenty years or so, much effort has been expended in trying to improve the performance of computers for signal processing and general scientific work such as weather forecasting. Much of the work has been on the exploitation of concurrency in the

algorithms, and within the arithmetic operations themselves. In general, this means using multiple processors in some sense, connecting them up in some suitable arrangement, and dividing up algorithms among them. This last operation is often non-trivial.

Systems using multiple processors can be classified into three different groups [1]: Single Instruction stream Multiple Data stream (SIMD), Multiple Instruction stream Single Data stream (MISD) and Multiple Instruction stream Multiple Data stream (MIMD). SIMD machines, as their name implies, perform the same operation concurrently on multiple items of data. They are often known as vector or array processors. Modern supercomputers are generally SIMD machines. They are particularly suited to matrix arithmetic, and can be used successfully in many signal processing applications. They are also very useful in many general scientific applications such as weather forecasting. They are not very efficient as general purpose computers because the degree of concurrency in many algorithms is difficult or impossible to identify.

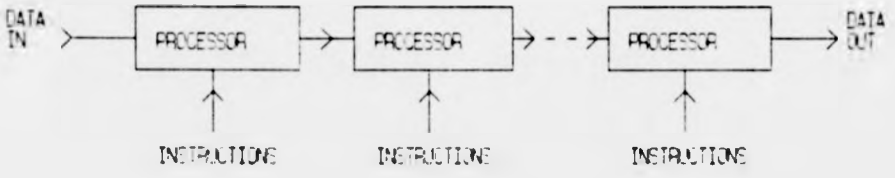


Fig. 2.1 Pipelined Processors

The most popular implementation of MISD machines is the pipeline. A pipeline of processors is illustrated in figure 2.1. Each item of data is processed by each processor in turn. Although the multiple

instruction streams are not operating on a single item of data concurrently, the pipeline is still classified as MISD because there is a single identifiable data stream. Pipelined architectures are not popular in programmable systems, being probably even less generally useful than vector processors, but they are very popular in hardwired systems. They are particularly appropriate for the implementation of recursive digital filters, which are usually formed from a cascade of second order elements.

The most common MIMD architecture is the multiprocessor. In this case, the individual processors operate in an essentially independent fashion, but sharing a common bus and common memory. The individual processors may also have a private bus and local memory. Tasks are shared out among the processors, and communication takes place via common memory. The main limitation on multiprocessors is the bandwidth of the common bus. Five or ten processors might typically give a performance improvement of two or three times; adding further processors will be to little avail. There is also the problem of sharing the tasks between the processors, which varies widely from application to application. Other MIMD architectures are possible, but there is always a trade-off between communications bandwidth and cost of provision. A fully interconnected network, where each processor has a direct connection to every other processor, soon becomes impractical! All MIMD structures suffer from the problem of dividing up the task.

2.1.1 Early Super-Computers

The first of the machines known as super-computers was the ILLIAC-IV, designed at the University of Illinois. This consisted of four quadrants, each with 64 processors. The quadrants could be configured to operate independently or cooperatively. Each quadrant contained one control unit, and each processor executed the same instruction but on different data. It was therefore basically a SIMD machine, although it in fact had four instruction streams. The processors were on a square grid, and each was connected to its four nearest neighbours. As can be imagined, the ILLIAC performed magnificently on suitable tasks such as matrix operations, but was of little use for much else.

The Texas Instruments Advanced Scientific Computer (ASC) [15] attacked the problem in another way. At the centre of this processor were four highly pipelined arithmetic units operating in parallel. The rest of the machine was concerned with getting the data in and out fast enough to satisfy these units. The whole machine was controlled by a peripheral processor which ran the operating system. The high data bandwidth was obtained by the special design of the memory. The memory was organised into 8 separate banks, each of which could be connected to any of 8 processor ports. Thus the memory accesses could occur in parallel, provided they were to different memory banks, and thus the processor-memory bandwidth was increased by a factor of 8. The ASC was therefore a MIMD machine with much attention paid to communications bandwidth.

These two computers demonstrate an important principle in the design of high-performance computers, and that is that the communica-

tions problem is much more difficult to solve than that of the computation alone. Thus the ASC, with its more general communications and control scheme, was more generally useful than the ILLIAC-IV, which was limited to a very small set of algorithms for efficient operation, although the ILLIAC outperformed the ASC on appropriate calculations. The overall complexity of the ASC was also greater, although it had only four processing units.

The ASC also demonstrates the exploitation of parallelism within the processing elements themselves. The fundamental arithmetic operations are divided up into various different parts, fetching the data, multiplication, normalisation, etc., and connected together in a pipeline. This provides the benefits of parallel processing to algorithms which otherwise could not support it.

2.1.2 Attached Processors

One of the most significant developments in the implementation of programmable digital signal processing systems, and general scientific computation, has been the attached processor. This is a special purpose processor which is "attached" to a general purpose minicomputer by some communications link, and has its architecture tuned to perform scientific computing algorithms efficiently.

One of the earliest attached processors was the Fast Digital Processor (FDP) [16] developed at the Lincoln Laboratories. This operated under the control of a Univac 1219 computer. The FDP uses four identical arithmetic elements (AEs), each consisting of a multiplier and an adder operating in parallel, and registers. There were also two data memories, accessible simultaneously, and a separate

instruction memory. The data memories were connected via buffers to the main Univac memory. Two instructions were executed at once: one to control data flow and the other to control the AEs. The design and interconnection of the AEs were optimised for complex arithmetic, second order filter sections and the FFT "butterfly" operation.

The FDP had two main problems. One was the bottleneck of communications between the Univac memory and the FDP memories, and the other was the difficulty of programming the four AEs in parallel.

Using the experience gained with the FDP, the LSP/2 was designed, again at the Lincoln Laboratories [6 pp261-264]. Rather than using four identical arithmetic units, the LSP/2 used separate functional units: an arithmetic logical unit (ALU), a multiplier, a divider, a shift and normalise unit, etc. There were 64 dual copy registers and a 4Kx32 data memory. All these units communicated on three buses. Program memory was separate. The common buses and memory proved to be a bottleneck, and in any one cycle an operation could only be initiated in one unit, along with a result being captured from one other unit. This made programming the LSP/2 for maximum efficiency a very difficult task.

These two attached processors illustrate, among other things, the usefulness of separating program and data memory. The combination of these into a single entity was one of Von Neumann's main contributions to computer design. However their separation increases the processor-memory bandwidth and thus speed of operation, at the expense of some loss of generality.

The problems in programming these early devices have been greatly reduced with the introduction of microprogramming techniques. The

Data General AP/130 [6 pp267ff] is designed as an attached signal processor for the S/130 minicomputer. This was not greatly different from earlier machines in its architecture. It had separate pipelined multipliers and add/subtract units and a sine/cosine look-up table. Programming the machine, however, was much simpler. Instead of programming the functional units separately, as with the LSP/2, the AP/130 will execute array operations, digital filters and FFTs all as single instructions which are extensions of the S/130 assembly language. This makes the programming of many signal processing systems an almost trivial task!

The recent trend towards providing computing power by distributed workstations has been accompanied by a number of attached processors for such workstations. Due to advancing technology they are much faster and cheaper than their predecessors. They can be attached to one workstation on the network and accessed as a network resource; indeed there could well be a few different special purpose attached processors on one network.

2.1.3 Single Chip Signal Processors

More suitable for use in systems which are likely to be produced in quantity are the single chip signal processors which have appeared on the market. The S2811 from AMI [17] is similar to the attached processors mentioned above, but it is designed to be micro-programmed by the user, and is much less tightly coupled to the host computer than the AP/130. The S2811 in fact has several architectural similarities to the FDP: it has two data memories and a separate instruction memory. It has a multiplier and an adder which operate concurrently, as does the arithmetic element of the FDP. The S2811,

however, has only one arithmetic element.

The Intel 2920 [2] is different in that it is designed to operate independently of any host computer, and in that it includes analogue circuitry as well as digital. It is in fact designed as an analogue-in, analogue-out, single-chip digital signal processor. It can perform 9-bit analogue-to-digital and digital-to-analogue conversion, although internal arithmetic operations are 24-bits wide. It has no multiplier, multiplications being performed as a series of additions and shifts. The chip employs a barrel shifter and an arithmetic unit which operate concurrently. It has found wide application in the telecommunications market for which it was primarily designed.

More recent than the S2811 and the 2920 are the TMS 320 series of processors from Texas Instruments [35], and a number of similar products from other manufacturers. These have concentrated on providing the basic operations as fast as possible, such that the latest version of the TMS320, the 320C25 can perform 16-bit multiply-accumulate operations in only 100ns. This is tremendous power from a microprocessor.

The Inmos transputer [36] is also interesting for signal processing. The transputer architecture is particularly designed to relieve the communications bottleneck in multi-computer systems, but the individual transputers also provide formidable processing power.

however, has only one arithmetic element.

The Intel 2920 [2] is different in that it is designed to operate independently of any host computer, and in that it includes analogue circuitry as well as digital. It is in fact designed as an analogue-in, analogue-out, single-chip digital signal processor. It can perform 9-bit analogue-to-digital and digital-to-analogue conversion, although internal arithmetic operations are 24-bits wide. It has no multiplier, multiplications being performed as a series of additions and shifts. The chip employs a barrel shifter and an arithmetic unit which operate concurrently. It has found wide application in the telecommunications market for which it was primarily designed.

More recent than the S2811 and the 2920 are the TMS 320 series of processors from Texas Instruments [35], and a number of similar products from other manufacturers. These have concentrated on providing the basic operations as fast as possible, such that the latest version of the TMS320, the 320C25 can perform 16-bit multiply-accumulate operations in only 100ns. This is tremendous power from a microprocessor.

The Inmos transputer [36] is also interesting for signal processing. The transputer architecture is particularly designed to relieve the communications bottleneck in multi-computer systems, but the individual transputers also provide formidable processing power.

2.2 THE IMPACT OF VLSI ON DIGITAL SIGNAL PROCESSORS

It has already been pointed out that the main effect of the advances in integrated circuit technology has been to make electronic components available in such volume and at such a low cost as to make digital signal processing viable for many more applications. For example, the FDP was implemented using 10 000 SSI integrated circuits in ECL technology. The AMI S2811 is a single LSI integrated circuit implemented in VMOS technology, and achieves about one quarter the throughput of the FDP [6 p272]. The comparison is quite staggering. It is interesting to note that, although increases in component density in analogue circuits have occurred, they have not been to the same extent as for digital circuits. This is largely because analogue components require a high signal to noise ratio, which cannot be achieved with the small dimension transistors used in digital circuits. This serves to reduce the penalty in terms of component cost for using digital techniques in signal processing. This effect is likely to become more noticeable as component dimensions in digital circuits are further reduced. Scientists believe that 100 nanometre is about the smallest usable transistor which can be fabricated. Such a transistor would, however, be of use only as a switch in digital circuits; it would be useless for analogue applications.

Early hardwired digital signal processing systems were implemented using many SSI circuits. A typical system would consist of many boards full of 16-pin packages. As technology progressed through MSI to LSI, larger building blocks, such as monolithic multipliers, became available. These larger building blocks are connected together using SSI circuits as a kind of "glue". Designers can even use gate arrays to replace these SSI circuits and further reduce the

component count.

With the advent of VLSI, however, it becomes possible to implement quite large signal processing systems on one integrated circuit, or on very few. Apart form the programmable devices already mentioned, there are a number of "standard parts" in this area. Many telecommunications signal processing functions are available off the shelf, and monolithic multipliers are becoming larger and faster. However, as circuit density increases further, more and more signal processing systems designers are needing to become involved with integrated circuit design. This situation is by no means unique to signal processing, as witnessed by the greatly increased interest in, and availability of, Application Specific Integrated Circuit (ASIC) design over the last few years.

Where full custom design is appropriate, in high volume applications, this does not pose any major problems. However, lower volume applications may suggest a semi-custom approach, but general purpose semi-custom systems such as gate arrays are likely to be somewhat inappropriate, in the same way as general purpose computers are usually inappropriate for the implementation of programmed digital signal processing systems. Standard cell approaches are more likely to be of use, provided that suitable cells are available for signal processing. Lyon [9] proposes such a system based around bit-serial building blocks. Denyer [10] presents a silicon compiler for digital signal processing, again based around bit-serial arithmetic. This is certainly an exciting and very worthwhile area of research.

This use of bit-serial arithmetic highlights two important issues. First is the fact that bit-serial arithmetic is more effi-

cient than parallel, because there is no carry-ripple delay or complicated carry look-ahead circuitry. This can give an advantage when the algorithms give rise to sufficient concurrency for many operations to be performed at the same time. This is normally so in the case of digital signal processing.

The second issue highlighted by the use of the bit-serial approach is in the area of data transmission. It is a fact of life in integrated circuit design that the wiring tends to take up much more space than the active components, and the inter-processor communication problem has already been highlighted. The interconnection of modules by single data lines has a great advantage over doing the same thing with, say, 24-bit buses. It will also alleviate the problem of pin-out limitations on integrated circuits and simplify printed circuit board design.

This wiring difficulty may even have an effect on the selection of algorithms suitable for VLSI. For example, the FFT algorithm is computationally much more efficient than a straight implementation of the DFT. However, the data communications strategy for an implementation on multiple processors is far more complex. Present means of evaluating algorithms tend to concentrate on the number of multiplications required; they take no account of interconnection strategies. There has been much interest shown by VLSI designers in systolic array structures [25 chapter 8]: arrays of identical processing elements connected together in a regular manner. Such systems put great emphasis on local communications, each processor being connected only to its nearest neighbour. Algorithms suitable for implementation on such structures are likely to be favoured for VLSI.

However, there are always going to be those who, for some reason such as low volume, flexibility or short design time, do not wish to be involved in IC design. This will tend to increase the demand for efficient, easily used, programmable components for digital signal processing. Thus it is worthwhile to investigate the means of making the power of VLSI available most effectively to the designer of programmed digital signal processing systems. It is to this end that the work described in this thesis was carried out.

CHAPTER THREE

AIMS OF THE PROJECT

The principle aim of the project is to make the potential processing power of VLSI available in programmed digital signal processing systems. The advantages of programmable systems over hardwired systems in terms of flexibility and ease of design have already been described. This work is aimed at reducing their disadvantages by improving their efficiency in use of silicon. This is to be achieved by tailoring the systems to a particular class of applications.

It was decided to concentrate effort on efficient implementation of difference equation calculations, such as are widely used in digital filtering applications. It was considered necessary to limit the field under consideration in such a way in order to obtain some concrete results. It would have been possible to concentrate on a different class of algorithms, such as the Fast Fourier Transform (FFT), but as has been pointed out these FFT algorithms pose problems in multiple processor configurations due to the non-locality of communication of data. Once a suitable design has been established for this one class of algorithms, its applicability for other algorithms can be evaluated. Such studies may suggest modifications to the architecture which would improve its suitability for these other algorithms without significant adverse effect on its suitability for the algorithms for which it was primarily intended.

Any attempt to design real-time signal processing systems must involve the exploitation of concurrency. The raw speed is simply not available to permit the majority of signal processing systems to

operate in a purely sequential manner. Fortunately, the majority of signal processing algorithms are amenable to this treatment. Thus the means used to exploit concurrency is a key issue, and in particular the inter-processor communication strategy adopted. It has been seen that the problems associated with the manipulation and communication of data are often much greater than those associated with the actual computations.

3.1 USING STANDARD MICROPROCESSORS

The project began as an investigation of the use of multiple standard microprocessors for the implementation of digital signal processing systems. It was hoped that in this way sufficient processing power could be made available for at least some signal processing applications. However, it was soon discovered that this approach was not really viable.

3.1.1 8-bit microprocessors

Most 8-bit microprocessors were aimed primarily at control type applications requiring little in the way of arithmetic computation. Consequently their arithmetic capabilities are poor. Consider, for example, using the Intel 8085 [22] to implement the digital filter described in appendix A. This filter is used throughout this thesis for the purpose of comparing various different implementation strategies.

The 8085 is an 8-bit machine, thus multiple precision arithmetic must be used for signal processing. In this case, 16 bit arithmetic is used. All arithmetic is performed on the 8085 using a single

accumulator register, known as the A register. Thus the assembly code to add the 16-bit number stored in the D and E registers to one stored in the B and C registers is as follows:

```
operator  operands

MOV       A,C
ADD       E
MOV       C,A
MOV       A,B
ADC       D
MOV       B,A
```

Each of these instructions takes 4 machine cycles to execute, these register-to-register instructions being the fastest on the processor. Thus the total time required for this addition is 24 cycles, or 8 μs on a 3 MHz 8085. There are only two registers left, and these are normally used for holding a 16-bit memory address. Thus if higher precision arithmetic were required, it would be necessary to store one of the numbers in memory. The following is the assembly code for such a 24-bit addition. This adds the 24-bit number stored in the three bytes of memory starting at address "ADR" to that stored in the B, C and D registers:

```
cycles  operator  operands

10      LXI       H,ADR
 4      MOV       A,D
 7      ADD       M
 4      MOV       D,A
 6      INX       H
 4      MOV       A,C
 7      ADC       M
 4      MOV       C,A
 6      INX       H
 4      MOV       A,B
 7      ADC       M
 4      MOV       B,A
```

This takes a total of 67 machine cycles, or over 22 μs.

Shift operations are just as bad. There is no possibility of multi-bit shifts, so all shifts have to be performed one bit at a time, and they can only be performed on the accumulator. Moreover, there is no arithmetic shift instruction for sign extension when 2's complement numbers are to be shifted right; this has to be concocted from a combination of one left rotate and two right rotate through carry instructions. Thus the instructions required to perform a 1-bit right shift on a 16-bit number in registers B and C is as follows:

```
operator  operands

MOV       A,B
RLC
RAR
RAR
MOV       B,A
MOV       A,C
RAR
MOV       C,A
```

Once again, all these instructions take 4 cycles, and thus the total is 32 cycles or nearly 11 μs. This code must simply be repeated for multi-bit shifts.

It can be seen from these examples that the 8085 is rather inefficient for implementing this kind of algorithm. Most of the time is spent shuffling data around rather than on the actual computations. This is due in part to the accumulator being a bottleneck: every arithmetic operation involves moving data to the accumulator before the operation and away again afterwards. Also, the delay involved in using off-chip memory can be seen. It will be seen that providing sufficient on-chip memory can make a significant contribution to the efficiency of signal processors.

3.1.2 16-bit Microprocessors

If the early 8-bit microprocessors were aimed at fairly simple control type applications, the next generation of 16-bit microprocessors are intended for much more general application. The Motorola 68000 series of processors, for example, is very popular in single-user workstations, and the best-selling IBM Personal Computer (PC) is based on the Intel 8086 range. For the purposes of this thesis, the Intel 8086 [23] will be compared with the 8-bit 8085 considered above.

At first glance, it is clear that the 8086 offers significant advantages over the 8085. It has, for example, a multiply instruction, although it still takes up to 133 machine cycles for a 16x16 bit multiplication, or 27 μs for a 5 MHz 8086. The 8086 has four general purpose registers, and all of these can be used for arithmetic operations, thus largely avoiding the accumulator bottleneck of the 8085. It would be possible to perform the add and shift operations of the filter described in appendix A on numbers up to 32-bit precision without needing to store one of the operands in memory. This was only possible up to 16-bit precision with the 8085.

As with the 8085, multi-bit shift operations are not provided. However, an arithmetic right shift is provided, and along with the absence of the accumulator bottleneck this means that shifting can be performed very much faster. It might be thought that the shift operations would not be so critical in a processor with a multiply instruction. However, the sequence of shift and add operations in appendix A for multiplying y_2 by its coefficient can be performed more quickly than using a multiply instruction. The total number of

shifts is 15, with 10 additions. In register-to-register mode, these operations take 2 and 3 machine cycles respectively. Thus 60 cycles are required altogether, as compared with a minimum of 118 for a multiplication. This is because the 8086 multiply instruction is implemented in microcode rather than with multiplication hardware, and coding the multiplier into the program as in appendix A is more efficient. In addition, the multiply instruction yields a 32-bit result, whereas the system proposed here discards the least significant 16 bits of the product. Where word lengths of more than 16 bits are involved, the saving from using shifts and adds are likely to be even more evident, as multiple precision multiplications are rather awkward.

It can be seen, therefore, that the 8086 is much better suited to digital filtering than the 8085. However, this advantage has been gained only at the cost of greatly increased complexity, including a higher chip count for a minimum usable system. Much of this complex functionality would not be used in digital filtering applications and is thus wasted. Moreover, the 8086 is still not ideal, lacking such things as multi-bit shift operations and sufficient on-chip memory. Such facilities would be much more useful than a number of those which are provided.

3.2 SPECIAL PURPOSE MICROPROCESSORS

Special purpose programmable signal processors have been mentioned a number of times already in this thesis. One of the earliest to become available was the Intel 2920 [2]. It is interesting to compare its performance with general purpose processors such as the 8085 and 8086 considered above. The program for implementing the

filter design of appendix A in the 2920 is in appendix B. Note that this program just performs the digital filtering; the 2920 also contains hardware for digital to analogue and analogue to digital conversion, and part of the instruction word is given over to controlling this analogue circuitry. In fact, with a filter as simple as this, the analogue functions take more time than the filter computations, and including them in this program would only confuse the issue. It can be seen that the entire filter can be implemented in 23 instructions. As the 2920 executes one instruction every machine cycle, this will take less than 10μs on a 2.5 MHz 2920. This is clearly much faster than is possible on either an 8086 or an 8085.

This great increase in speed is due to the way the 2920 is tailored to digital filtering applications. It is very efficient at performing additions and shift operations. A barrel shifter is available, permitting multi-bit shifts up to 2 bits left and 13 bits right. One shift operation and one ALU operation may be performed in each machine cycle. Also, the entire system is contained on one chip, and there are none of the delays associated with off-chip memory access. Program and data memories are separate, permitting both to be accessed simultaneously and thus enabling single cycle instruction execution. The amount of memory provided, both for program and data, is, although small, quite adequate for most filtering applications. Indeed, the 2920 is primarily intended for the processing of telephone quality speech signals with a sampling frequency of 8 kHz. If more memory were provided to permit the implementation of more complex algorithms, then this sampling frequency could not be maintained.

3.3 NETWORKS FOR SIGNAL PROCESSING

In the design of systems involving multiple processors, two problems are frequently encountered. One is the partitioning of algorithms to make use of the parallel processing power available. The other is the establishment of an efficient inter-processor communication strategy. Fortunately, in signal processing the partitioning of algorithms to exploit concurrency is often much simpler than is the case in general.

Various different interconnection strategies have been proposed for multiple processor systems. In the most general case, each processor is connected to every other processor by a direct link. This is clearly not practical for more than a very few processors. One alternative is to use a regular array of processors, with each connected only to its nearest neighbours. The Illiac IV (qv section 2.2.1) is an example of this, although such an interconnection scheme could also be implemented with the processors executing different instruction streams rather than the common instruction stream of the Illiac. As was pointed out for the Illiac, such architectures are particularly suited to a small set of algorithms such as matrix operations, but are grossly inefficient for much else. It should be pointed out, however, that a multiple instruction stream system might prove to be of somewhat more general applicability than the Illiac.

Other interconnection schemes use multiple ports into a common memory, or communication on a bus connected to all processors. True multi-port memory is very difficult to achieve and typically memory must be split up into "banks", with only one processor able to access a particular bank at any one time. Once again, such systems are only

viable when a fairly small number of processors is involved. Bus systems suffer a similar limitation due to restricted bandwidth, and neither system is suitable for systems involving large numbers of processors.

Swartzlander and Heath [20] point out two unique aspects of signal processing networks. The first is that data can often be grouped into quite large blocks. The second is that the networks are typically topologically irregular, the topology being related directly to the algorithms. They propose a circuit switching scheme, the time taken to set up a circuit being a small part of the transmission time for a reasonably large block of data. Circuit switching nodes can be made much more simply than packet switching nodes, the main disadvantage being this set-up time.

Real time signal processing systems must be able to process data sufficiently fast to keep up with the sampling rate. This could be a potential problem in a system which allocates tasks to different processors, or establishes communication paths, at execution time. This could pose problems to a system implemented using the method proposed by Swartzlander and Heath. It cannot be guaranteed beforehand that a particular link can be established, and thus a data block transmitted, within the time required. One possible solution would be to allocate circuits in advance for particular streams of data, rather than allocating them for individual data blocks during system operation. This approach, however, could mean that there would not be sufficient circuits available to set up all the required communications paths. It would be possible in this case to enable communications paths to share individual interconnection circuits, as with the Swartzlander and Heath scheme, but in this case still allocating them

in advance to provide guaranteed performance. This pre-allocation could be done by simulating a system operating the Swartzlander and Heath scheme to ensure that the system performed adequately.

The adoption of pre-allocation techniques loses the fail-soft aspects of the Swartzlander and Heath approach. However "fail-soft" is a doubtful concept in real-time signal processing, as a soft failure which reduces the speed of operation below the threshold for real-time operation is, in effect, a hard failure, unless the system can somehow reconfigure itself to perform a simpler set of operations. If a system is to be reliable, providing guaranteed performance even in the face of failure of one or more of its processing nodes, then the effect of such a failure must be analysed in advance and allowed for in system design.

It is important to remember when discussing these signal processing networks that the systems under consideration in this thesis are intended for implementation as VLSI circuits. In section 2.2 it was pointed out that locality of communications is very important in VLSI, and indeed it always yields simpler wiring whatever means of implementation is used. This tends to favour regular arrays of processors, connected to their nearest neighbours, and has led to many researchers to seek out algorithms suitable for implementing on such systolic array structures. The scheme of Swartzlander and Heath could in fact be realised on such an array, imposing an irregular communication topology on a regular connection structure. The efficiency of such a scheme would depend on the algorithm being implemented. It might be necessary when using such a scheme for the nodes in any particular communication path to pass data on to subsequent nodes in the path in subsequent cycles, otherwise the delays in

sending data over long paths would negate the speed advantages of local communications.

With regard to the implementation of digital filters, IIR filters can be formed from second order sections connected in cascade or in parallel. FIR filters are most appropriately constructed from cascaded sections of any appropriate length. Thus both IIR and FIR filters can easily be implemented in cascade form on a simple pipeline of processors. The alternative for IIR filters, namely parallel connection, is much harder to handle in practice. The input data must be distributed to all the parallel processors, and their outputs recombined in some way. Pipelines are therefore the most appropriate form of multiple processor for the implementation of digital filters.

This is an example of the way in which selection of a sufficiently narrow field of application can significantly reduce the complexity of a solution. The techniques of Swartzlander and Heath can be left to processors of more general applicability. It also illustrates how algorithms can be selected not only on the grounds of computational efficiency but also architectural efficiency. The parallel and cascade forms of IIR filters have identical computational efficiency, but the cascade form fits in better with the architecturally simple pipeline.

3.4 PROCESSORS FOR SIGNAL PROCESSING NETWORKS

It would be perfectly possible to connect general purpose microprocessors together to form a network suitable for digital signal processing. This would not, however, overcome the essential inefficiency of such processors for signal processing. It is worth

investigating how special purpose signal processors might be used in networks.

The Intel 2920 has been seen to be suitable for the implementation of digital filters. However, it is not designed for use in processor networks; it is intended for use as an analogue-in, analogue-out signal processing system on one chip. It is in fact possible to connect 2920s together, but as they are not designed with this in mind it is rather cumbersome. The AMI S2811 is more hopeful in this regard, being a purely digital processor with no analogue functions, operating under the control of a host mini- or micro-computer. However, this chip is available only in a mask-programmable version, which makes it difficult to use in a research environment. The serial input/output interface of this processor could undoubtedly be very useful for connecting it into networks due to the simplicity of the wiring.

The Texas TMS320 processor is well suited to the implementation of a wide range of digital signal processing algorithms. A single processor provides formidable processing power; a network of such processors could prove very powerful indeed. The Inmos transputer, being designed with interconnection in mind, could be significantly easier to use in networks, and also provides significant processing power. A combination of the two architectures could be very interesting indeed. However, neither was available at the start of the project.

Having discovered problems in the use of any available processor at the time the project began, it was decided that the aim of the project should be the design of a processor for digital signal pro-

cessing, suitable for use in a network. This decision happily coincided with the availability to the author of computer aided design tools for custom integrated circuits, and the possibility of prototype fabrication. In line with all the above considerations , and in order to make the design tractable and the implementation efficient, difference equations were chosen as the target application, and pipelines as the target network architecture. The processor should be as flexible as possible in use, providing that this should not interfere with the overriding requirement for efficiency to enable it to compete with hardwired systems. Once designed, its suitability for other digital signal processing algorithms can be evaluated. This may lead to suggestions for improvements to the architecture. Once again, these would only be implemented if to do so would not be to the detriment of its performance for difference equations.

CHAPTER FOUR

DEVELOPMENT OF ARCHITECTURE

4.1 EFFICIENT ARCHITECTURES FOR SIGNAL PROCESSING

It has been stated that it is always possible to achieve higher performance from hardwired systems than from programmed systems implemented in the same technology. This is because all elements of a hardwired system are designed for the particular application in question. They are likely to be in constant use, or at least for the majority of the time, otherwise their function would be combined with other units. On the other hand, some elements of a programmable system may not be used in a particular application, or may not be ideally suited to it. They may only be operational for a relatively small proportion of the time. Hardwired systems therefore make more efficient use of their constituent parts than programmable systems. The reason for the design of specialised programmable signal processors is to eliminate those parts of a general purpose system which are not required for signal processing applications, and to include extra parts which are useful in signal processing but are not usually provided in general purpose processors. In this way, they can become more efficient in using their constituent parts.

The resources available to the designer of VLSI signal processing systems are silicon area and time; he must try to eliminate waste as far as possible in both these quantities. His own time is indeed another resource which he does not wish to waste: it is not worth spending man-years of effort in squeezing out the last few square microns of chip area. Adopting a strategy of local communications is

an example of trying to conserve both chip area and time. Complicated wire routing is a classic way to use vast areas of silicon, and long-distance communications pose problems due to propagation delays. In fact, wasting time with long delays is similar to wasting area, as it means that a large area of silicon is waiting, or being wasted, during such delays. Efficiency can be assessed roughly by the proportion of a system which is actively being used at any one time.

A signal processing system consists of four types of circuitry: memory, communications, arithmetic, and control. These are required in both hardwired and programmable systems, although in hardwired systems the control logic will be implied in the wiring, whereas in programmable systems the control logic consists of the program memory itself and the instruction decode circuitry. As the main concern in digital signal processing is to perform large numbers of arithmetic operations fast, efficient utilisation of the arithmetic logic is of paramount importance. All other areas, memory, control and communications, must be kept down to a minimum.

It has already been suggested that the area consumed by communications circuitry can be kept down by employing only local communications. It is also important to consider memory. Memory is an important part of signal processing systems, the Z-transform function z^{-1} representing delays of one sample period and implying memory. In hardwired systems, only the required amount of memory is provided. It is important in programmable systems to waste as little memory as possible. Memory is wasted when word-lengths are longer then required, and when too many memory locations are provided. Using excessively long word-lengths causes inefficient use of arithmetic logic as well as memory. From this point of view, it would be

desirable to have variable word-length available under program control.

The question of how much data memory to provide is a thorny problem. Providing too much is wasteful; providing too little can mean that certain algorithms cannot be implemented. When designing a processor with on-chip memory, or worse still a processor with its own memory which can be part of a multi-processor chip, then the problem becomes acute. It is understandable that one should err on the side of generosity. However, when the chip area taken up by memory becomes significantly larger then the rest of the processor, it makes little sense to add more memory. It makes more sense to replicate the processor and have two processors and twice as much memory. This strategy provides more efficient use of chip area where there is sufficient memory in one processor, and the second processor may be used for another task. In the case where all the memory is required by one processor, the unused processor does not consume a large proportion of the overall silicon area. This consideration should be a great help in determining the amount of memory to be provided with each processor. It is, in fact, unlikely that memory from one processor will be used by an adjacent processor as implied in this discussion; what is more likely is that the algorithm will be further subdivided to run on two processors when the memory available on one processor is insufficient. This does not, however, alter the basic argument as to the amount of memory which ought to be provided.

Reducing the silicon area consumed by the control circuitry is best achieved by reducing the size of the program memory. The size of the instruction decoder is also important. Good instruction set design can help to keep down both the complexity of the decoder and

the amount of program memory required. As with the data memory, it makes sense to use extra processors to provide extra memory rather than to provide vast amounts with each processor. In fact, the requirements for program memory and data memory are related, and a sensible ratio between these could be established for the type of algorithm being considered.

4.2 ADVANTAGES AND DISADVANTAGES OF BIT-SERIAL ARITHMETIC

A number of researchers in the field of digital signal processing have advocated the use of bit-serial arithmetic techniques (qv section 2.2). This means that carry propagation is performed synchronously, and thus cycle times can be lower as it is not necessary to wait for carry signals to propagate across the full word length in each cycle, nor is it necessary to use complex carry look-ahead logic. This reduced cycle time can lead to higher throughput provided that sufficient arithmetic operations can be performed concurrently. This can be illustrated by a simple example. If it is required to add together 8 pairs of 8-bit numbers, then 8 cycles and 8 full adders are required to perform the additions sequentially using parallel arithmetic or concurrently using serial arithmetic. However, in the former case the requirement for carry ripple will increase the cycle time, making the throughput higher in the latter. This higher throughput represents more efficient use of silicon.

Bit-serial transmission of data has also been seen to be advantageous over parallel transmission, especially when the communications requirements are complex. It must be pointed out, however, that parallel arithmetic and serial communications are not wholly incompatible as data may be serialised for transmission to other processing

units.

A major disadvantage of bit-serial arithmetic is that there is no algorithm for bit-serial division. This is pointed out by Chen and Willoner in their paper on a bit-serial multiplier [24]. Although this would be devastating in a general purpose computing environment, division is not used in the vast majority of signal processing algorithms.

It is notable that bit-serial techniques are only proposed for hard-wired signal processing systems. In such systems, the required function is defined by the interconnection of the processing elements. In programmable systems, the function is defined by a program which is interpreted by the control portion of the processor. This control portion will typically be about the same complexity however many bits there are in the words being processed. Thus in bit-serial approaches the ratio of silicon area consumed by control logic to that consumed by the arithmetic logic is likely to be very high. This is highly undesirable when the desired end is to perform arithmetic operations as efficiently as possible. It may be possible to get round this efficiency problem by using a SIMD (Single Instruction-stream Multiple Data-stream) architecture, where each instruction would operate on a number of data items. This, however, produces the limitation that only identical operations could be performed concurrently. In other words, the possible applications would be further restricted.

There is a further advantage in using bit-serial arithmetic: it makes configurable wordlength easier to achieve. In the case of hardwired systems, the wordlength is determined by the designer. In

the case of programmable systems, however, configurable wordlength would be a significant advantage, increasing efficiency for systems requiring a short wordlength and easing the implementation of those requiring long wordlength. For example, IIR filters typically require more precise calculations than FIR filters, since the output is fed back and roundoff errors can accumulate. Also, as the poles of IIR filters approach the unit circle in the z-plane, the required precision increases; as the poles approach the unit circle the filter is becoming nearer to instability, and accumulated roundoff error could make the filter unstable. It is generally agreed that 24 bit precision is adequate for nearly all filters except the most unstable, therefore processors such as the Intel 2920 provide 24 bit precision for all calculations. However, most FIR filters can be adequately implemented using 12 bits and most IIR filters using 16 bits. What is more, all memory locations are 24 bits wide in the 2920, and this can result in a significant waste of resources if it is not required, as memory typically accounts for a large proportion of all digital systems.

4.3 VARIABLE PRECISION PROGRAMMABLE COMPUTERS

A computer architecture has been proposed by Kartashev and Kartashev [21], which uses 16-bit wide computer elements (CEs). These can be configured dynamically, under program control, to form a varying number of dynamic computer (DC) groups. Each DC group contains a number of CEs and has a word-length which is a multiple of 16 bits. Although this architecture is intended for large complex supercomputers, it does give some insight into a means for providing programmable systems with varying word-length.

When a DC group is formed in the Kartashev and Kartashev system, arithmetic is performed in parallel within the group, thus there is a problem of carry propagation when large word-lengths are used. It is obvious, for example, that carry propagation in a 64-bit DC group will take longer than in a 32-bit group. An interesting alternative to consider is using synchronous carry between CEs within a DC group. This would avoid the carry propagation problem but would bring with it many new ones of its own making. Such is engineering.

4.4 A NIBBLE-SERIAL ARCHITECTURE

What is being proposed is, in essence, a compromise between a bit-serial and a fully parallel architecture. If the word-length of the CEs is sufficiently short, then the carry propagation delay within each CE will not be significant, and adequate flexibility of overall word-length may be provided. If it is sufficiently long, then the ratio of silicon area used for control logic to that used for arithmetic logic and memory may be low enough for reasonable efficiency. The area occupied by control logic in each CE will, to a first approximation, be independent of the wordlength of the CE. Clearly a compromise must be made. A word-length of 4 bits was chosen for this project, as it seemed to give such a reasonable compromise. Undoubtedly carry propagation across 4 bits should pose no problem, and a word-length variable in increments of 4 bits should give adequate flexibility. In terms of data communications, the routing of 4-bit buses is not as easy as for single lines, but gives a significant advantage over routing 24-bit buses. The option of serialising the data for transmission remains a possibility. It is more difficult at this stage to assess the ratio of control logic to

arithmetic logic; this will be clearer when the project is completed, and may suggest that a longer word-length would be more appropriate. The choice of 4 bits is certainly not sacrosanct. As 4 bits has been chosen, the architecture can be described as nibble-serial. The 4-bit wide processing element will be known as a Signal Processing Ele-

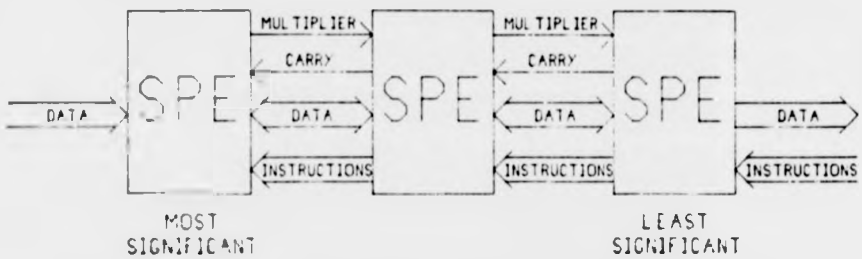


Fig. 4.1 3 SPEs Form a 12-bit Computer

ment (SPE). The fundamental mode of operation can best be explained by means of a simple example. Consider a 12-bit computer made up from three SPEs as in fig. 4.1. If two 12-bit numbers are to be added together, then in the first machine cycle the least significant four bits of each number are added together in SPE1. The carry bit from this addition is then passed on to SPE2, where in the next cycle this is used as the carry input for the addition of the next most significant four bits of the two numbers. Once again, the carry output from this operation is used as the carry input for the addition performed during the next cycle in SPE3 of the four most significant bits.

Once an SPE has completed its part of any operation, it is free to go on to perform a new operation while other SPEs are still performing the original operation on more significant sections of the

operands. It must also be noted that one particular SPE always operates on the same 4-bit field in every number, and thus all memory for this particular field for every word is incorporated within that SPE.

It can be seen that an operation which is performed during one cycle is performed during the next cycle in the next most significant SPE: the SPE next on the left. It seems reasonable, therefore, that SPE instructions should be passed each cycle from one SPE to its neighbour on the left. This will ensure proper synchronisation of operation.

The mode of operation is in many ways similar to performing multiple precision arithmetic on an 8-bit microprocessor. In this case, the operations are also performed on successively more significant fields of the numbers, and the carry output from one operation is used as the carry input for the next. However, the operations are all performed on the same piece of hardware. Although this use of one piece of hardware throughout is simpler conceptually and avoids a number of problems, it does have some fundamental disadvantages. The most obvious is reduced throughput: the processor must wait until an entire operation is completed before proceeding to the next. The throughput of the SPE approach will generally be n times greater, where n is the number of nibbles in a word. The word-length used can be increased in the SPE approach simply by using an extra SPE and with virtually no effect on throughput.

Another disadvantage of the single processor approach is not so apparent. It has to do with the provision of memory. If a single processor has enough memory to store all the data required for its

envisaged applications using 16-bit words, it will not have sufficient if the word-length must be increased to 24 bits. On the other hand, using the SPE approach, the extra memory will automatically be added with the extra SPE which must be added to increase the word-length.

The main problem with this nibble-serial approach lies in the implementation of conditional instructions. In a conventional computer, condition codes are set according to the results of arithmetic operations, for instance indicating if a result is zero, or if an overflow has occurred. With the nibble-serial approach an instruction is not completed, nor can condition codes be set, until the following instructions are being executed. In addition, the codes would be generated in the most significant SPE, some distance from the least significant SPE where the instructions are being generated. It is thus considered impractical to implement a condition mechanism. This has the side effect of making division impossible, but division is not generally required in signal processing (qv section 4.2). The lack of conditionals, in particular overflow detection, might be considered serious. However, a comparison with the Intel 2920 is once again helpful; this processor provides very little in the way of condition handling, and overflow is handled by controlling whether a number "wraps around" or limits on overflow. If no overflow handling is provided, numbers will wrap around. The limiting behaviour of the 2920 is used to simulate clipping in analogue signal processing systems, however good design can eliminate the possibility of overflow.

In bit-sequential processors, data must be supplied least-significant bit first, as carry propagates from least-significant to most-significant. Similarly, with nibble-sequential processors, the

data must be supplied least-significant nibble first. It is also desirable that all input data should be supplied to the same place, that is the same SPE. If it were necessary to supply data to the SPE which is to process that particular nibble, then extra circuitry would be required to direct it to the correct place. If data is supplied least-significant nibble first to the most-significant SPE, and passed on in subsequent cycles to the next less significant SPE, the SPE on the right, then after the requisite number of cycles the data will be in the correct place. Similarly data can be output from the least-significant SPE, least-significant nibble first.

Fig. 4.2 demonstrates the basic operation of a processor made from SPEs. In this diagram, a 12-bit number is input to a processor made from 3 SPEs, a 12-bit number stored in memory in the SPEs is added to it, and the result is output. Each box represents the state of an SPE, the upper half containing the instruction being executed, and the lower half the contents of the accumulator at the end of the machine cycle. An "XIO" instruction, eXternal Input Output, causes an SPE's accumulator to be loaded with the contents of the accumulator of the SPE to its left. The leftmost SPE's accumulator will be loaded with externally presented data, and the contents of the rightmost SPE's accumulator will be output. An "ADD" instruction causes the contents of register B of an SPE to be added to its accumulator. A "NOP" instruction means no operation. More detail on the operation of these and the rest of the SPE instructions will be given later. X_2, X_1 and X_0 represent the three nibbles of the number being input, and Y_2, Y_1 and Y_0 the nibbles of the output number. Successive rows of the figure represent successive machine cycles.

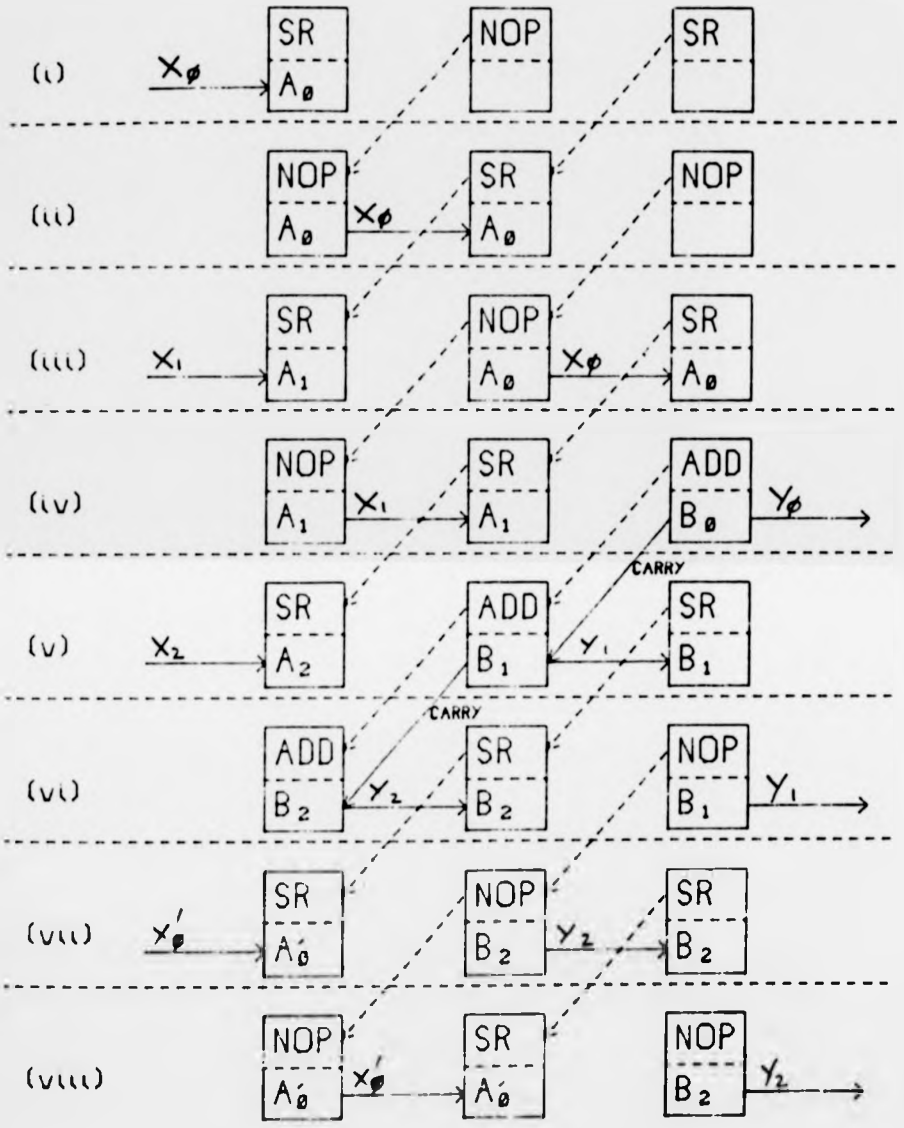


Fig. 4.2 Basic Operation of Processor made from SPEs

There are two points of interest worth noting at this stage. The first is the requirement for NOP instructions between successive XIO instructions. This is unavoidable in this scheme, although the NOPs can be replaced with any internal SPE instruction which does not affect the accumulator. The second is that more cycles are used shifting numbers in and out than in the addition itself. However, in real systems the calculations will be more complex than a single addition, and the overhead for the input and output of data will not be such a great proportion.

4.5 FUNCTIONALITY REQUIRED

Having established the basic mode of operation of processors formed from SPEs, it is necessary to establish exactly what functionality is required for the chosen application area. Difference equations involve above all the calculation of sums of products.

This implies the basic operations of addition, subtraction and multiplication. It is worth considering the functionality of the Intel 2920, as this processor has been found useful in digital filtering applications. It does not include a hardware multiplier. In many cases, the multiplications required in digital filters are by a pre-determined constant, for example one of the filter coefficients; this can be implemented as a series of additions or subtractions and shifts coded directly in the program. Multiplication by a variable, required for example by adaptive filters, can also be implemented by this shift and add approach, provided that the arithmetic operations can be made conditional upon certain bit-positions in the multiplier operand. Because different SPEs are assigned to specific nibbles in the numbers, it would be difficult to implement

any kind of hardware multiplier, even if it were to operate only on 4 bit sections of the numbers. However, the approach used in the 2920, based on shifts and additions, is perfectly possible.

As has been seen, division is not generally required in signal processing. Scaling down can be performed by multiplication, and in some cases scaling by the nearest power of two by shifting is adequate. The 2920 does not provide any means for division. Both for this scaling, and for multiplication as described above, it is vital to provide shift operations. It was pointed out in section 3.1 that one of the drawbacks of general purpose microprocessors for signal processing is the lack of multi-bit shift operations. The Intel 2920 provides shifts varying from 2 places left to 13 places right in a single instruction; in fact these shifts are performed on one of the operands to an ALU operation in the same cycle as the ALU operation. With an SPE-based processor, it would be possible to implement shifts of up to 4 bits in either direction by incorporating a barrel shifter; any longer shifts in a single cycle would violate the rule of local communications only, as one SPE would have to obtain its data other that from its own nearest neighbours.

It may be thought that this limiting of shift distance to 4 bits, rather than the 13 bits in the 2920 might lead to inefficiencies in the implementation of multiplication. However, there are two distinct ways of implementing shift and add multiplications, as illustrated in Fig 4.3. In the first case, Fig 4.3 (a), which corresponds to standard long multiplication as taught at junior schools, the multiplicand is shifted by an appropriate number if bits prior to being added into the product. At each stage the multiplicand is only added into the product if the corresponding bit of the multiplier is a one.

```
                0101 x 0011   (5 x 3)

                   0000
                    0000
(a)                  0101
                      0101
                   --------
                   00001111    (15)
                   --------


                   00000000
                 + 0101
                   --------
                   01010000
               shift 00101000
                 + 0101
                   --------
                   01111000
(b)            shift 00111100
                 + 0000
                   --------
                   00111100
               shift 00011110
                 + 0000
                   --------
                   00011110
               shift 00001111    (15)
                   --------
```

Fig. 4.3 Two methods for shift-and-add multiplication

In the second case, Fig 4.3 (b), the multiplicand is always added into the high order bits of the product, and the product is shifted right by one bit after each addition. Once again, the addition is only performed if the corresponding bit of the multiplier is a one, and in this case the bits of the multiplier must be taken least significant first. This method removes the need for the long shifts required by the first technique, and has another significant advantage. It is usual in signal processing systems to use fractional numbers, i.e. the most significant bit represents $\frac{1}{2}$, the next bit $\frac{1}{4}$, etc. Thus when two 4-bit numbers are multiplied to give an 8-bit product, then the least significant 4 bits may be discarded to leave

a number of the same significance and precision as the original numbers. Thus only a 4-bit adder is required, and the bits which are shifted out at the least significant end may be discarded. In the first case a full 8-bit adder is required.

Memory is another function required in a signal processor, as implied by the delay operation z^{-1}. As has been said, it is difficult to decide how much memory to provide. One second order filter section with both poles and zeros requires four memory locations, so the 40 locations provided on the 2920 would allow for 10 such sections. As this seemed a not unreasonable complexity of calculation for one processor, it was decided to provide 64 locations, this being the nearest number also a power of two and thus making best use of the 6-bit address required to address 40 locations. As with the choice of 4 bits for the width of an SPE, this decision is not sacrosanct. It will be easier to assess if this is a reasonable choice when it becomes apparent what proportion of the silicon area is taken up by this memory (q.v. section 4.1), and what kind of maximum data rate would be permitted when using all the memory.

While considering memory, it is worth pointing out that the z^{-1} delay essentially consists of moving data from one memory location to another. On most processors, including the 2920, this can only be done one datum at a time. It may be useful to be able to shift larger blocks of memory in one cycle by building this operation into the design of the memory. It is at least worth investigating the cost in terms of silicon area of building this operation into the memory.

4.6 OPERATIONAL DETAILS

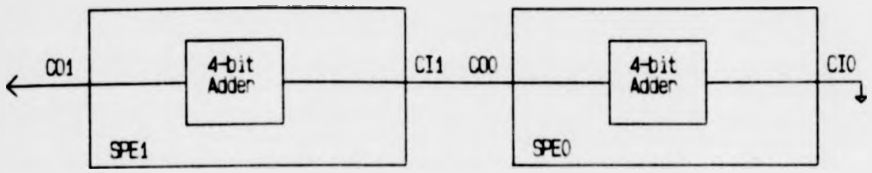
4.6.1 Addition and Subtraction

The operation of addition has already been described in section 4.4. Subtraction is easily implemented by the use of 2's complement number representation, whereby a subtraction is implemented by complementing and adding. Forming a 2's complement involves negating each bit and adding a one at the least significant bit position. This is most easily implemented by setting the carry input to the adder to a one, rather than to zero as is used for addition. For this reason, the carry input to an SPE is inverted for subtraction, so that it can be held at zero permanently for the least significant SPE. To permit proper carry propagation between SPEs, the carry output must also be inverted. Fig 4.4 shows how carry propagation operates for both addition and subtraction.

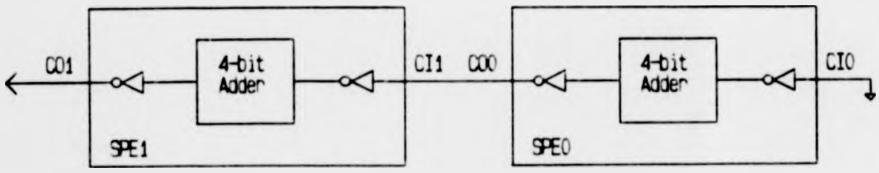
4.6.2 Shifting, Input and Output of Data

Shift operations of one to four bits are to be provided in each direction. All shift operation by their nature require communication between neighbouring SPEs, which is achieved by using 4-bit wide bidirectional ports on each side of an SPE.

When an SPE executes a right shift instruction, data is required from the SPE to its left. This SPE, however, is not yet aware of the arrival of the shift instruction. It is therefore necessary for each SPE to output the contents of its accumulator on its right hand data port at all times when it is not executing a left shift instruction. In this way data is always available to the SPE on the right should it be executing a right shift instruction.



(a) Addition



(b) Subtraction

Fig. 4.4 Carry propagation for addition and subtraction

A problem arises in the most significant SPE due to the adoption of 2's complement number representation. When a 2's complement number is shifted right, the sign bit must be replicated rather than zeros being shifted into the most significant bit position. Thus the most significant SPE must know that it is such; there seems to be no alternative but to include this in the configuration programming.

Input and output of data is achieved by a sequence of right shift operations. In this case all the shifts are 4 bits in length. When data is input, the most significant SPE must transfer data which, it is assumed, will be presented at its left hand data port. This is in

contrast to ordinary shifts, when this SPE must rather replicate the sign bit. The output of data requires no such special operation, except that it must coincide with the input of data by the SPE to the right of the least significant SPE, which will be the most significant SPE of the next processor in the pipeline.

Left shifts require data to be delayed by one cycle before being presented to the SPE on the left. Thus when executing a left shift instruction, an SPE loads the contents of its accumulator into a special purpose latch before loading the data presented at its left hand data port into its accumulator. In the next cycle, when the SPE to its left is executing the left shift instruction, the data from the latch will be output to the left hand data port. As indicated above, this is the only situation where the transfer of data is from right to left, and the contents of the accumulator are not output on the right-hand data port.

4.6.3 Multiplication

Multiplications are to be implemented by means of shifts and additions, in accordance with the scheme of Fig 4.3 (b). This scheme does not cover the use of 2's complement numbers, but can be simply extended to implement Booth's Algorithm. In the unsigned case, an addition is performed if the corresponding bit in the multiplier is a one. With Booth's Algorithm, the possibility of a subtraction is introduced, and the operation is selected according to two bits of the multiplier, the current bit and the previous bit. The least significant bit is still considered first. The operation is selected according to Fig 4.5. With the first bit of the multiplier, there is no previous bit and it is considered to be zero. In many cases it

Present bit	Previous bit	Operation
0	0	No-operation
0	1	Addition
1	0	Subtraction
1	1	No-operation

Fig. 4.5 Selection of operation for Booth's Algorithm

will be desired to multiply by a constant. In this case, the constant can be programmed as an appropriate sequence of shifts, additions and subtractions. Multi-bit shifts may be used where a series of no-operations is required.

In some circumstances, however, it will be necessary to multiply by a variable. This requires additions, subtractions and no-operations to be performed conditionally according to the appropriate bit-positions of the multiplier. A multiplier register is provided which may be shifted right by one bit positions in any one machine cycle, in a manner similar to the shifting of the accumulator. The least significant SPE executes an instruction which causes an addition, subtraction or no-operation to be performed dependent upon the value of the least significant bit of the multiplier and its previous value. This addition, subtraction or no-operation is then passed on to subsequent SPEs in place of the original instruction.

It is worth noting that it is not necessary for the multiplication to use the full configured precision of the processor for the multiplier. For example, in a particular filter it may be sufficient to use 8-bit filter coefficients while using 16-bit arithmetic. In this case the multiplications could be performed in 16 cycles rather than 32 cycles if a full 16-bit multiplier were used. Two cycles are required for each multiplier bit: one for the addition, subtraction

or no-operation and the other for shifting the product.

4.7 INSTRUCTION SET

As the Intel 2920 has been used as a guide to the functionality of the SPE, it is reasonable to look to it for assistance in defining the instruction set. It is however alarming to realise that the instruction words of this processor are 24 bits wide. Admittedly some of these bits are used to control the analogue elements of the 2920, but a significant problem remains. Each SPE has to pass instructions in and out, making such an instruction width totally unacceptable.

It is therefore of vital importance to reduce the instruction width. The 2920 uses 2-operand memory to memory operations, and thus the instruction includes source and destination addresses, each 6 bits wide. These can be eliminated by using registers with implied function. The only operations involving memory are load and store, and these have only one address.

An SPE has 64 memory locations and thus requires 6 bits of address. There are three registers: an accumulator, a B register and a multiplier. The B register is used to contain the second operand for the two-operand operations add and subtract and hence for multiplicands. To provide load and store instructions for each of three registers requires six operations, or three bits, leaving 128 codes out of a 9-bit word for non-memory operations. However, the B register and the multiplier never need to be stored, as the results of all operations are put into the accumulator. Thus only four load and store operations are required, but with an 8-bit word and 6-bit

addresses no codes are left for other operations. If the "load accumulator" instruction is omitted, then 64 codes remain for non-memory operations.

This loss of the "load accumulator" instruction is not expected to be serious. An instruction can be provided to load the accumulator from register B, thus making it possible to load the accumulator in two cycles. In fact it will rarely be necessary to load the accumulator. When performing multiplications, the accumulator must be initialised to zero, for which an instruction can be provided. For the sum of product type calculations required to implement difference equations, a multiplication can be performed first, and then other numbers can be loaded into register B and added into the accumulator. The result can then be stored in memory.

The instruction set is described fully in Appendix C. It can be seen that the remaining instructions fit easily into the 64 possible codes. The arithmetic operations of addition, subtractions, etc. require 8 codes and the shift instructions require 4 codes in each direction to indicate the length of the shift from one to four bits. Thus 48 codes remain for input-output instructions and the "special" instructions to implement Booth's Algorithm multiplication and to move half of memory by one location to implement the z^{-1} delay. There is ample space for more instructions should they be required.

CHAPTER FIVE

REGISTER TRANSFER LEVEL DESIGN OF AN SPE

Having developed an architecture for a nibble-serial processor, it is necessary to verify its operation by building a model. The ideal vehicle for this is a functional or register-transfer level simulator, which permits the concepts to be tested before proceeding to more detailed design and implementation.

However, although in theory design may proceed in a totally top-down fashion, in practice it is preferable to take account of likely implementations if wise, informed decisions are to be taken. This was highlighted by Mead and Conway [25 p157] when they incorporated a barrel shifter into their OM data path chip. A designer working with standard TTL parts would avoid a barrel shifter at all costs, whereas in NMOS it is relatively simple, small and regular. This was presumably an influence in the inclusion of a barrel shifter in the Intel 2920. It certainly was in the case of the SPE, thus demonstrating that likely implementation has already affected the design process. In fact, the whole philosophy of the SPE and nibble-serial processing arises from the intention to fabricate many of them on a single VLSI device.

It is becoming very clear that CMOS is destined to be the dominant technology for VLSI, largely due to its low power consumption. The speed of MOS processes is also increasing steadily as device geometries shrink. Whereas high speed bipolar technologies such as ECL may have a future, they are ruled out for VLSI on account of their high power consumption. Even NMOS is likely to consume too

much power for the implementation of the densest (ULSI?) chips. Circuits based on III-V semiconductors such as Gallium Arsenide (GaAs) may well become popular in the future due to their inherently faster operation resulting from the higher mobilities of the charge carriers, but the technology is at present only in its infancy and it will be some time before it poses a real challenge to silicon. Therefore the SPE designs should be targeted towards CMOS implementation.

Apart from the items already mentioned, such as the inclusion of a barrel shifter, one major influence of the technology is the selection of the clocking strategy. This is particularly important in the SPE with the requirement that all instructions be executed in precisely one cycle. The most popular clocking strategy for MOS designs is that based on two-phase non-overlapping clocks, used in conjunction with level sensitive transparent latches [30 pp203ff]. This posed a problem with the first register transfer simulator tried, ISPS. The registers in this simulator are all edge-triggered, making it difficult to simulate such MOS designs sensibly. In particular, if designs make use of the transparency of the latches, simulation is virtually impossible. It was attempted to work around the problem by timing some transfers at the beginning of a clock phase and others at the end, but whereas this did help a little, it proved rather cumbersome and error-prone.

When the HILO simulator [27] became available, it was decided to investigate the possibility of using it at the functional level in place of ISPS. HILO provides the level sensitive latches required and permits the declaration of arrays of HILO circuits, which is how a system incorporating several SPEs is described. One disadvantage

of HILO when compared with ISPS is the inflexible output format, the output being available only in binary. Where an output represents a digitised analogue quantity as in signal processing, a decimal representation would be more useful. However, it was easy to write a simple post-processor program to convert the HILO output into a more suitable format. The advantage of being able to use level-sensitive latches and thus produce a much more accurate simulation model certainly outweighed the disadvantages of HILO.

In fact, graphical output packages were available for post-processing output from both HILO and SPICE, the circuit simulator used [28]. The HILO graphical post-processor produces a timing diagram typical of logic analyser output, whereas the SPICE post-processor produces plots af analogue voltages and currents against time. When simulating SPEs, the digital output from HILO actually represents an analogue quantity. A program called SPEPRO was therefore written to convert the HILO output file into the form of a SPICE results file, and it was thus possible to plot the output of the system as an analogue quantity using the SPICE post-processor. This was much simpler than writing a new graphics package.

5.1 AN SPE ASSEMBLER

It is evident that adequate verification of the SPE architecture requires the development of programs for filters. Whereas it is always possible to write machine code by hand, it is a slow, error-prone task. The HILO simulations were to prove to be very slow, taking several hours of CPU time on a PRIME 750 minicomputer to simulate a few cycles of a filter program, and the debugging of programs was therefore a slow and expensive process.

On the other hand, it would be an enormous task to develop an assembler specifically for the SPE, so an alternative approach was sought. A microcode assembler system had been developed at UMIST to assist people working with bit-slice microprocessors to develop their instruction sets [29]. The system allows fields to be defined within an instruction word with particular characteristics, and associates mnemonics with operation codes in these fields. Finally, legitimate combinations of fields for constructing an instruction word are defined. A new assembler can be constructed by providing a definition file containing this information. This system was found to be

```
word 8
field memop <7:6>
field memad <5:0>
field miers <5> (default 0,, overlap or)
field instr <7:0> (,,overlap or)
field sftop <7:2> (,,overlap or)
field sftln <1:0>
field extio <7:0> (,,overlap or)
setof memop sacc=01b,lrb=11b,lmier=10b
setof miers smier=1b
setof instr
    cond=00001100b,
    smem=00001101b,
    nop=000b,
    neg=001b,
    a0=010b,
    b0=011b,
    ab=100b,
    ba=101b,
    add=110b,
    sub=111b
setof sftop sl=010b,sr=100b
setof sftln b1=00b,b2=01b,b3=10b,b4=11b
setof extio xi=10100b,xo=11000b,xio=11100b
inform memop,memad
inform instr,miers
inform sftop,sftln,miers
inform extio,miers
```

Fig. 5.1 Definition file for the SPE assembler

suitable to implement a simple assembler for the SPE. The definition file for the SPE assembler is included as Fig. 5.1. The reader is

not expected to fully comprehend this text, but rather to gain an appreciation of how simple it is to create a new assembler using this system.

The assembler produces a listing file showing the source and the object code produced in hexadecimal format. It also produces a binary file containing the object code. The system is written in the PASCAL programming language, and PASCAL procedures are provided to access the object code from the binary file. A program called MCSTOHILO was written in PASCAL using these procedures to translate the binary file into a format suitable for use as input to HILO simulations.

5.2 A REGISTER-TRANSFER MODEL OF THE SPE

A block diagram of a single SPE is given in Fig 5.2. The detailed operation of an SPE will be described in this section, and the various blocks related to their declarations in the HILO description of an SPE, which can be found in Appendix D. The operation of the SPE is greatly constrained by the need for each operation to be executed in precisely one clock cycle. This is a great contrast to the majority of general purpose microprocessors. In the MOS implementation of an SPE, the use of a two-phase non-overlapping clock is envisaged, as is customary with such a technology. It is therefore vital to assign tasks to the appropriate clock phase early in the design cycle.

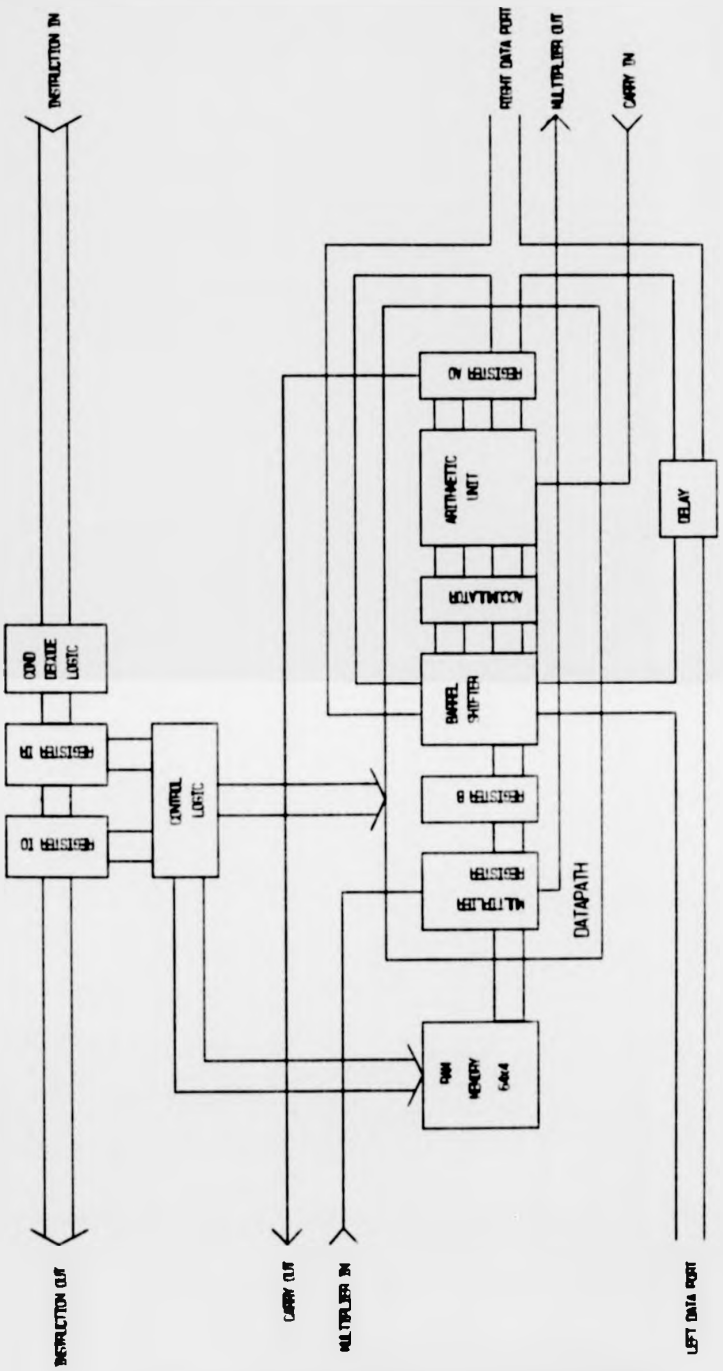


Fig. 5.2 Block Diagram of a Single SPE

5.2.1 Assignment of Tasks to Clock Phases

The overall timing framework is governed by the relationship between arithmetic operations within one SPE and the transfer of data between adjacent SPEs. For example, when a right shift operation is performed immediately following an addition, then the result of the addition performed in a particular SPE must be available for transfer to the SPE on its right during the same clock cycle. This imposes a basic discipline that arithmetic operations will be performed during ø1 and inter-SPE data transfers during ø2. It is assumed that two adjacent SPEs will be controlled by the same clock signals, and therefore that all SPEs on a chip will be clocked together. At first this seems to violate the principle of local communications, but this is not the case. The clock must be generated in one location, but can then be passed on from one SPE to the next. In this way there may be significant clock skew over the entire chip, but it will not be significant between two adjacent SPEs. In any case, this chip-wide distribution of one or two signals for a synchronised clock is a very different matter to the global communication of random signals. VLSI technologies are likely to use more than one layer of metal. The clock signals could be distributed on one of the upper layers where the metal will be thicker and the tracks wider, thus keeping propagation delays down to a minimum.

The HILO functional description of the SPE may be found an Appendix D. The reader may find it useful to refer to this while reading the remainder of this section.

As the arithmetic operations are performed during ø1, instructions must be properly in place by the start of this phase. There-

fore instructions, as well as data, must be transferred during Ø2. The new instruction is latched into the instruction register, referred to as IR in the HILO description, at the end of this phase. The "COND" instruction (q.v. section 4.3.1), which causes addition, subtraction or no-operation to be performed according to certain bits of the multiplier register, is interpreted at the same time, and it is an "ADD", "SUB" or "NOP" instruction which is latched into the instruction register rather than the "COND" instruction itself. This not only ensures that the correct arithmetic instruction is ready for execution during Ø1, but also ensures that the interpretation of the "COND" instruction occurs in the least significant SPE. The instruction register is therefore constant during Ø1, but cannot be used to control operations during Ø2. For these, the Instruction Output register, IO, is decoded. This is loaded directly from the Instruction Register IR and latched at the end of Ø1. It is therefore constant during Ø2. It is also directly from this register that the instruction is output to the next SPE on the left.

The timing of the arithmetic operations is rather complicated. They are implemented using an adder whose output is latched into the ALU output register AO at the end of Ø1 in every clock cycle. The selection of operation is achieved by controlling the inputs to the adder: INA, INB and INCI. The adder is to be implemented using a Manchester carry chain [25 p150], which will be precharged during Ø2 so that the operation can take place in Ø1. The inputs to the adder must therefore be correctly determined before the start of Ø1 so that the carry lines are not discharged due to incorrect data. INA and INB, which do not represent real registers but rather combinatorial logic and thus are never latched, therefore need to be stable before

ø1. In fact, the logic represented by INA and INB could have been incorporated into the definition of AO; their separate definition serves only to improve the clarity of the HILO description. They are loaded according to IR from the accumulator and the bus, and are constant from before the end of ø2 of the previous cycle until after the beginning of ø2 of the current cycle. INCI, the carry input, is a real latch which depends on IR and is controlled by ø2. Thus INCI is also constant for the same period. It has been said that IO should be used rather than IR to control operations which take place during ø2, however in this case the latching of INCI can be viewed as preparation for the current cycle during ø2 of the previous cycle, and therefore controlled by the new instruction being latched into IR rather than the existing instruction to be found in IO. IR will be valid before the end of ø2 and can therefore be used to control a latch whose output must be valid by, and is latched on, the falling edge of ø2.

The ALU output register AO is latched at the end of ø1. It should be noted that the ALU performs a NOP operation, i.e. the AO register is loaded with the contents of the accumulator, whenever the SPE is executing a non-arithmetic instruction. The AO register is actually 5 bits wide, the most significant bit being used to store the carry output from the ALU operations. This bit appears at the carry output of the SPE (CO in the HILO description) for transfer to the next most significant SPE. Note the inversion of the carry signal during subtraction to achieve correct 2's complement operation.

It is from the least significant 4 bits of the AO register that data are taken for output on the right hand and left hand data ports respectively. This ensures that this output, which occurs during ø2,

takes account of any arithmetic operations which have taken place during ø1 while being unaffected by any shift operations which may affect the contents of the accumulator during ø2. It is this use of the AO register which makes it important that it is loaded from the accumulator during ø1 for all non-arithmetic operations.

The output to the right hand data port RDP occurs during all cycles other than those when a left shift is being performed. The tri-state buffer RDPB, controlled by the signal RDPEN, enables data from AO onto the RDP. During cycles when such output is required, RDPEN is raised at the start of ø2 and lowered at the start of ø1. The value of RDPEN is determined from IO rather than IR, in accordance with the overall timing strategy. The timing of RDPEN ensures that data is valid at the end of ø2, when it may be required by the adjacent SPE, while also ensuring that it will never be enabled at the same time as the left hand data port of the adjacent SPE, which would cause a clash.

Output to the left hand data port LDP is similarly under the control of the signal LDPEN and the tri-state buffer LDPB. In this case, output to the LDP must only take place one cycle after the execution of a left shift instruction. This delay is achieved for the data by the latches LDPX1 and LDPX2, and for the control signal LPDEN by latches LDPEX1 and LDPEX2.

The accumulator itself is called ACC and is latched at the end of ø2. Its input is taken from the output of a barrel shifter which allows it to take its data from some combination of the left data port LDP, the ALU output register AO and the right data port RDP. Unless a shift operation is being performed, the barrel shifter

implements a zero bit shift and the input to the accumulator is taken directly from the AO register. During right shift operations, the signal MSPE, which must be set to one in the most significant SPE, controls whether data is taken from the LDP or by sign extension from the AO register.

The multiplier register consists of two registers, MIER and MIERX, which are latched by the clock signals Ø2 and Ø1 respectively. Each is 5 bits wide, as the Booth algorithm for multiplication requires the previous value of the least significant bit of the multiplier to be remembered. The multiplier register may be loaded from memory during Ø1, or shifted right by one place during Ø2. Hence all inter-SPE transfers take place during Ø2. As with the accumulator shifts, successive shifts of the multiplier should not occur in successive cycles. This does not pose any speed reduction for multiplication as each stage consists of two instructions: a "COND" instruction and a right shift instruction.

A BUS is used for certain communications within the SPE to reduce the number of direct connections required: it is used for the transfer of data to and from the SPE's memory and to take data from the B register RB to the INB logic for input to the adder. During Ø2, BUS is always loaded from RB so that the data for the adder can be valid before the start of Ø1 as is required. Being fabricated in MOS technology, the bus will have capacitance and can be used as a short term storage element and is therefore declared as a register in the HILO description. Memory reading and writing is performed during Ø1. Although this will overwrite the value of RB loaded onto the bus during Ø2, it does not matter in this case as memory and arithmetic operations cannot be performed simultaneously and therefore the data

from RB cannot be required by the adder. HILO does not allow RAM to be declared as level sensitive, so writing the RAM locations is achieved using the "WHEN" statements at the end of the description. These statement also simulate the operation of the "SMEM" instruction which causes all data in memory to be shifted by one location. The register XADR, which does not exist within a real SPE, is required for this simulation. The memory writing and shifting is where the HILO functional description deviates furthest from the eventual implementation, but in this case it is not considered to be a significant problem. HILO certainly enables the SPE to be modelled much more accurately than would have been possible with ISPS.

The Data Strobe Input and Output lines (DSI and DSO) are used to indicate when the most significant SPE requires data from the outside world and when the least significant SPE has data available for the outside world. They may or may not be used within an eventual system, but are used within the HILO simulations.

The fact that the functional HILO description of the SPE only occupies about one page of computer printout should not lead the reader to consider that its derivation is trivial. Indeed the development of the SPE architecture to the point of this description represents the majority of the work involved in this doctorate. The subsequent detailed design in NMOS technology did not involve as much effort, and the original work of the thesis is largely contained within this succinct description. The following simulations based on this HILO description will demonstrate that the concept of SPE-based systems can indeed be made to work, and the detailed implementation, described in chapter six, is of secondary importance in the academic value of the project.

5.3 SIMULATIONS BASED ON THE REGISTER-TRANSFER MODEL

The register transfer model of the SPE provides an ideal vehicle for the verification of the concepts used in the development of the architecture. The simulations included in this section are based on a 16-bit processor consisting of four SPEs, and they progress from a simple 16-bit addition to a full filter implementation. No attempt has been made to simulate a pipeline of such processors. There are two reasons for this. The first reason is that the simulations based on this processor used significant amounts of computer time; a more complex processor would consume even more resources. The second reason is that the full implications of attempting to integrate a reconfigurable pipeline of processors on a VLSI chip have not yet been addressed. Further consideration will be given to this topic in chapter seven.

5.3.1 A MODEL OF A 4-SPE PROCESSOR

Before any simulations could begin, it was necessary to develop a model of a system consisting of four SPEs. This is fairly trivial using HILO, and the description of such a system, known as SYS4, is given in figure 5.3. Data and instructions are provided by the HILO circuits DATAIN and INSTIN. They are purely for the purposes of the simulations, and do not represent circuits which would be implemented in a real system. They are very simple circuits which consist of ROMs with a strobe line. Whenever the strobe line is pulsed, the ROM address is incremented and the next value output. DATAIN is strobed by the Data Strobe Input (DSI) line from the most significant SPE. INSTIN is simply strobed by ø1, to provide a new instruction every clock cycle. In general, INSTIN is configured so that the program

```
CCT SYS4(CLR);
SPE
    MSPE(GND,,LDP[3:0],,,,,,,,,M[2],C[2],D3[2],D2[2],D1[2],D0[2],
        I7[2],I6[2],I5[2],I4[2],I3[2],I2[2],I1[2],I0[2],
        DSI,,VCC,PHI1,PHI2)
    LSPE(M[0],C[0],D3[0],D2[0],D1[0],D0[0],
        I7[0],I6[0],I5[0],I4[0],I3[0],I2[0],I1[0],I0[0],,
        GND,RDP[3:0],IIN[7:0],,DSO,GND,PHI1,PHI2)
    XSPE[1:0](M[2:1],C[2:1],D3[2:1],D2[2:1],D1[2:1],D0[2:1],
        I7[2:1],I6[2:1],I5[2:1],I4[2:1],I3[2:1],I2[2:1],I1[2:1],
        I0[2:1],M[1:0],C[1:0],D3[1:0],D2[1:0],D1[1:0],D0[1:0],
        I7[1:0],I6[1:0],I5[1:0],I4[1:0],I3[1:0],I2[1:0],I1[1:0],
        I0[1:0],,,GND,PHI1,PHI2);
DATAIN GETDATA(DIN[3:0],DSI,CLR);
INSTIN GETINST(IIN[7:0],PHI1,CLR);
BUFIF1(1,1) DIBUF[3:0](LDP[3:0],DIN[3:0],DSI)
    BODGE[3:0](XXXX,DOUT,CLR);
CLOCK0(100,300,700) P1(PHI1);
CLOCK0(600,300,700) P2(PHI2);
SUPPLY0 GND;
SUPPLY1 VCC;
TRI LDP[3:0] RDP[3:0] XXXX[3:0] D3[2:0] D2[2:0] D1[2:0] D0[2:0];
UNID PHI1 PHI2 DSI DSO DIN[3:0] IIN[7:0] I7[2:0] I6[2:0] I5[2:0]
    I4[2:0] I3[2:0] I2[2:0] I1[2:0] I0[2:0] M[2:0] C[2:0];
INPUT CLR;
REGISTER(1,1) DOUT[3:0]=0 LOADIF1 CLR;
WHEN DSO(1 TO 0) DO DOUT=RDP;
WHEN DSO(0 TO 1) DO DOUT=XXXX.
```

Fig. 5.3 HILO Description of a 4-SPE Processor

goes back to the start when all the instructions have been executed; this would be the normal mode of operation for implementing filters. DATAIN is usually configured to stop the simulation when all the data have been used, to provide a means of stopping the simulations. This method of data and instruction input has been found to be very flexible and useful, meaning that different simulations could be performed simply by changing the definitions of these two circuits. The program which converts the SPE assembler output into HILO is written to produce an appropriate definition of INSTIN directly, making the process even simpler. The definitions of DATAIN for the simple simulations were written by hand. For the filter simulations, a program was written to produce a definition of DATAIN corresponding to a

sinusoid of a particular frequency.

It proved a little more difficult to get the data output in a suitable format. This in fact illustrates one of the limitations of HILO. Output is only produced in binary; several lines cannot be grouped together and, for example, output as an octal or decimal number. Also, the control of when output values are printed is also crude. Output can be printed whenever one of the output values changes, using the DISPLAYCHANGE command, or alternatively whenever the circuit enters a stable state, using a DISPLAYSTABLE command. What is desired in these simulations is that the data output should be printed whenever an active transition occurs on the Data Strobe Output (DSO) line of the least significant SPE. HILO provides no obvious means of achieving this. A register DOUT was defined within SYS4 (see figure 5.3), which takes its data from the right hand data port of the least significant SPE on every negative-going transition of the DSO line of that SPE. This should ensure that DOUT changes every time a transition occurs on the DSO line, and therefore the HILO DISPLAYCHANGE command can be used. However, it is possible that two successive values of DOUT will be identical, thus preventing the data from being printed by the DISPLAYCHANGE command. It was therefore arranged that on every positive-going edge of DSO, DOUT should be set to "don't care" (binary XXXX in HILO). This indeed ensures that the value of DOUT is printed every time a pulse occurs on DSO. Unfortunately, the intermediate XXXX value is also printed, but this can be ignored.

5.3.2 A SIMPLE SIMULATION: ADDING TWO NUMBERS

```
loc   in co  line   source.

0000  4 1c     1. xio
0001  2 00     2. nop
0002  4 1c     3. xio
0003  2 00     4. nop
0004  4 1c     5. xio
0005  2 00     6. nop
0006  4 1c     7. xio
0007  2 05     8. ba
0008  4 14     9. xi
0009  2 00    10. nop
000a  4 14    11. xi
000b  2 00    12. nop
000c  4 14    13. xi
000d  2 00    14. nop
000e  4 14    15. xi
000f  2 06    16. add
```

Fig. 5.4 SPE program to add two 16-bit numbers

This first simulation is of the addition of two 16-bit numbers. Figure 5.4 shows the listing file from the SPE assembler for this simulation. The "xio" instructions, eXternal Input and Output, cause a 4-bit number to be read into the accumulator of the most significant SPE, and output from the accumulator of the least significant SPE. The "xi" instructions cause only eXternal Input. These instructions are, of course, essentially 4-bit shift instructions, causing the value in the accumulator of one SPE to be transferred to the accumulator of the SPE on its right. As with all shift instructions, they have to be separated by "nop" (No OPeration) instructions to ensure correct operation. However, these can be replaced by other non-shift instructions, and in this case the "ba" instruction causes register "b" to be loaded from the accumulator, and the "add" instruction causes the value from register "b" to be added into the accumulator. This means that the calculation can be embedded entirely in the time required to implement the shifts. In more complex programs, this

would not be so, however the time taken up by input and output may still not be insignificant.

```
CCT INSTIN(INST[7:0],STB,CLR);
INPUT STB CLR;
REGISTER(1,1) INST[7:0]=0 LOADIF1 CLR
    ADR[3:0]=0 LOADIF1 CLR;
ROM(0:15) IROM[7:0] (8 BIN)
00011100,
00000000,
00011100,
00000000,
00011100,
00000000,
00011100,
00000101,
00010100,
00000000,
00010100,
00000000,
00010100,
00000000,
00010100,
00000110;
WHEN STB(0 TO 1) DO INST=IROM[ADR];
WHEN STB(1 TO 0) DO IF ADR=15 THEN ADR=0
    ELSE ADR=ADR+1 ENDIF.
```

Fig. 5.5 Circuit INSTIN for simulation of addition

Figure 5.5 shows the HILO definition for the circuit INSTIN which corresponds to the program of figure 5.4. It is produced automatically from the assembler binary file by a simple pascal program written by the author. As can be seen, it simply defines a ROM, which contains the SPE machine code for the program. Each time a pulse occurs on the strobe line, the ROM address is incremented by one and the next instruction output. The program will loop indefinitely.

Circuit DATAIN provides the data for the simulation and also causes the simulation to finish once all the data have been exhausted. Its HILO definition can be found in figure 5.6. The numbers are input four bits at a time, with the least significant

```
CCT DATAIN(DATA[3:0],STB,CLR);
INPUT STB CLR;
REGISTER(1,1) DATA[3:0]=0 LOADIF1 CLR
ADR[7:0]=0 LOADIF1 CLR;
ROM(0:23) DROM[3:0](4 BIN)
0000, 0000, 0000, 0000,
0000, 0000, 0000, 0000,
0000, 1001, 1000, 0011,
0000, 1011, 1100, 0111,
0000, 0000, 0000, 0000,
0000, 0000, 0000, 0000;
WHEN STB(0 TO 1) DO IF ADR>23 THEN FINISH
    ELSE DATA=DROM[ADR] ENDIF;
WHEN STB(1 TO 0) DO ADR=ADR+1.
```

Fig. 5.6 HILO circuit DATAIN for simulation of addition

four bits first. The first two 16-bit numbers are both zero; the second pair forming the real test of addition. The two numbers are 0011100010010000 and 0111110010110000 respectively. The sum should be 1011010101000000. If the numbers were interpreted in 2's complement notation, this would represent an overflow; however this does not affect the addition test as the addition of unsigned numbers or 2's complement numbers is identical. The final two numbers are again both zero. This is to allow the SPEs to finish operating on the previous numbers before the simulation is terminated.

```
WAVEFORM WSYS
STIMULUS CLR=1;
1 CLR=0.
```

Fig. 5.7 HILO Waveform file for all simulations

HILO simulations require a waveform file to define the inputs to the circuit being simulated. In the case of these simulations, the system SYS4 is virtually self-contained, containing its own instructions, data, and clock definitions. The only signal required in the waveform file is the CLR signal which initialises all the registers.

The actual waveform file used can be found in figure 5.7.

```
                DDDD
                0000
                UUUU
                TTTT
                [[[[
                3210
                ]]]]
---TIME--
       0  0000
    1602  XXXX
    1902  0000
    3602  XXXX
    3902  0000
    5602  XXXX
    5902  0000
    7602  XXXX
    7902  0000
   17602  XXXX
   17902  0000
   19602  XXXX
   19902  0000
   21602  XXXX
   21902  0000
   23602  XXXX
   23902  0000
   33602  XXXX
   33902  0000
   35602  XXXX
   35902  0100
   37602  XXXX
   37902  0101
   39602  XXXX
   39902  1011
   49602  XXXX
   49902  0000
   51602  XXXX
   51902  0000
```

Fig. 5.8 Output from Simulation of Addition

The results of the simulation are shown in figure 5.8. The reason for the "XXXX" lines is described above. The first four non-XXXX values output, from time 1902 to time 7902 are meaningless, as they correspond to output before any data have been input. The next four, from time 17902 to time 23902, represent the output arising from the first two numbers input. As these were both zero, it is not

surprising (but very comforting!) that the output is also zero. The next four, from time 33902 to time 39902 correspond to the second pair of numbers input. The 4-bit nibbles are output least significant nibble first, so this output corresponds to the binary number 1011010101000000, which has already been established as the correct sum for the two numbers. This simulation therefore serves to demonstrate that the SPE architecture developed is at least capable of adding two numbers together. The numbers used were carefully selected to verify the operation of the adders for all combinations of A, B and Carry inputs, and to check the operation of inter-SPE carry propagation. It may seem that the achievement of addition is trivial, however it does verify the majority of the timings carefully derived in the previous sections. Indeed, while the SPE definition was being developed, once the simulation of addition was operating correctly, there were very few flaws revealed by the subsequent simulations.

5.3.3 THE SIMULATION OF MULTIPLICATION

The SPE is intended for the implementation of difference equation type digital filters, and is therefore to be capable of addition, subtraction, multiplication and the storage of data. The capability for addition has already been demonstrated. The filter simulations to follow will demonstrate the majority of the possible functions of an SPE, but they do not implement multiplication by a variable. When a fixed filter is being implemented, as in these simulations, the filter coefficients are programmed as a series of shifts and additions or subtractions. The SPE is, however, capable of multiplication by a variable, which could be of use in adaptive filter applica-

tions.

```
loc   in co  line   source.

0000  4 1c    1. xio
0001  2 00    2. nop
0002  4 1c    3. xio
0003  2 00    4. nop
0004  4 1c    5. xio
0005  2 00    6. nop
0006  4 1c    7. xio
0007  1 40    8. sacc 0
0008  4 14    9. xi
0009  1 80   10. lmier 0
000a  4 14   11. xi
000b  2 00   12. nop
000c  4 14   13. xi
000d  2 00   14. nop
000e  4 14   15. xi
000f  2 05   16. ba
0010  2 02   17. a0
0011  2 0c   18. cond
0012  3 30   19. sr b1 smier
0013  2 0c   20. cond
0014  3 30   21. sr b1 smier
0015  2 0c   22. cond
0016  3 30   23. sr b1 smier
0017  2 0c   24. cond
0018  3 30   25. sr b1 smier
0019  2 0c   26. cond
001a  3 30   27. sr b1 smier
001b  2 0c   28. cond
001c  3 30   29. sr b1 smier
001d  2 0c   30. cond
001e  3 30   31. sr b1 smier
001f  2 0c   32. cond
0020  3 30   33. sr b1 smier
0021  2 0c   34. cond
0022  3 30   35. sr b1 smier
0023  2 0c   36. cond
0024  3 30   37. sr b1 smier
0025  2 0c   38. cond
0026  3 30   39. sr b1 smier
0027  2 0c   40. cond
0028  3 30   41. sr b1 smier
0029  2 0c   42. cond
002a  3 30   43. sr b1 smier
002b  2 0c   44. cond
002c  3 30   45. sr b1 smier
002d  2 0c   46. cond
002e  3 30   47. sr b1 smier
002f  2 0c   48. cond
```

Fig. 5.9 SPE program for multiplication of two 16-bit numbers

The SPE program for the multiplication of two 16-bit numbers is given in figure 5.9. Many of the instructions used are the same as for as addition and require no further explanation. The first number read in is to be the multiplier. As there is no instruction provided for direct transfer from the accumulator to the multiplier register, it is necessary to go via memory. In this program, memory location 0 is used, the transfer being achieved by the instructions "sacc 0" and "lmier 0". These instructions mean "Store ACCumulator" and "Load MultiplIER" respectively. There is no time penalty from using two instructions; they are interleaved in the obligatory gaps between shift instructions. The second number is treated as the multiplicand, and is stored in register "b". The accumulator is set to zero by the "a0" command before commencing multiplication. The multiplication itself is implemented by a sequence of "cond" and "sr b1 smier" instructions. The "cond" or "CONDitional" instructions cause addition, subtraction or no-operation to be performed according to the least significant bits of the multiplier register in the least significant SPE, for the correct operation of Booth's Algorithm multiplication using 2's complement numbers. The "sr" instruction is a "Shift Right" instruction; the "b1" flag signifies that this is a one-bit shift. The "smier" or "Shift MultiplIER" flag may be added to any instruction which does not access memory to cause the multiplier register to be shifted right by one bit. In this case, there is one less "sr" instruction than there are "cond" instructions. This is correct when the numbers are considered as fractional, i.e. the leftmost bit is the sign bit and the next bit has a weighting of 0.5. This is the most likely mode of operation for signal processing applications. If integer multiplication is required then an extra "sr" instruction would be necessary.

```
CCT DATAIN(DATA[3:0],STB,CLR);
INPUT STB CLR;
REGISTER(1,1) DATA[3:0]=0 LOADIF1 CLR
    ADR[7:0]=0 LOADIF1 CLR;
ROM(0:43) DROM[3:0](4 BIN)
    0000, 0000, 0000, 0100,
    0000, 0000, 0000, 0010,
    0000, 0000, 0000, 0100,
    0000, 0000, 0000, 1110,
    0000, 0000, 0000, 1100,
    0000, 0000, 0000, 0010,
    0000, 0000, 0000, 1100,
    0000, 0000, 0000, 1110,
    0110, 1100, 1101, 0100,
    1011, 1110, 0111, 1011,
    0000, 0000, 0000, 0000;
WHEN STB(0 TO 1) DO IF ADR>43 THEN FINISH
    ELSE DATA=DROM[ADR] ENDIF;
WHEN STB(1 TO 0) DO ADR=ADR+1.
```

Fig. 5.10 Circuit DATAIN for the simulation of multiplication

Figure 5.10 is the HILO definition for circuit DATAIN for the simulation of multiplication. Each pair of 16-bit numbers will be multiplied together, the former of each pair forming the multiplier and the latter the multiplicand. The format of the DATAIN file is exactly as for the simulation of addition. The first two numbers represent 0.5 and 0.25, using the fractional notation discussed above. The subsequent three numbers represent the other three quadrants of this multiplication, thus the first sum is 0.5 * 0.25, the second 0.5 * -0.25, the third -0.5 * 0.25 and the fourth -0.5 * -0.25. The last two numbers represent, in binary, 0.100110111000110 and -0.100100000010101 respectively. The decimal equivalents are approximately 0.60760 and -0.56314. The correct answer to the multiplication is -0.0101011110011001 (decimal -0.34217). The 2's complement notation for this is 1101010000110011.

The results of the simulation of multiplication are presented in figure 5.11. The XXXX lines have been omitted to save space and

—TIME—	DDDD 0000 UUUU TTTT [[[[3210]]]]
0	0000
1902	0000
3902	0000
5902	0000
7902	0000
49902	0000
51902	0000
53902	0000
55902	0001
97902	0000
99902	0000
101902	0000
103902	1111
145902	0000
147902	0000
149902	0000
151902	1111
193902	0000
195902	0000
197902	0000
199902	0001
241902	0011
243902	0011
245902	0100
247902	1101

Fig. 5.11 Results of simulation of multiplication

avoid confusion; they were, of course, present in the actual output from HILO. Once again, the first four output values, from time 1902 to time 7902, can be ignored. The next four values correspond to the output for the first test. The value output is 0001000000000000, which in our format corresponds to 0.125, the correct result for the first sum which was 0.5 * 0.25. The next output value is 1111000000000000, or -0.125 in decimal, and is once again correct. The subsequent values, -0.125 and 0.125, are correct results for their respective multiplications. The final four 4-bit values, output from time 241902 to time 247902, are the result of the final mul-

tiplication. The value is 1101010000110011, which has been seen to be the correct answer, corresponding to -0.34217. The correct operation of multiplication has thus been demonstrated.

5.3.4 THE SIMULATION OF A REAL FILTER

A 2-pole Chebyshev low-pass filter was designed for the purpose of verifying the operation of a processor built from SPEs. The details of the filter design are given in Appendix A. The cut-off frequency in the digital domain is $\frac{1}{50}$ of the sampling frequency.

The SPE program for this filter is given in figure 5.12. The filter coefficients are built into the program as sequences of "sr" (Shift Right), "add" and "sub" instructions. This is the most efficient way to implement multiplication by a constant on SPEs; multiplication by a variable was verified by the simulations in the previous section. Memory locations 20 and 21 are used as scratchpad memory for the storage of intermediate results. Location 32 is used for the storage of y_{n-1}, the previous value of the filter output, and location 33 for y_{n-2}. Locations 34 through 36 are used for the storage of x, x_{n-1} and x_{n-2} respectively. The "smem" (Shift MEMory) instruction is used in this program. It causes the contents of memory to be moved one location higher, simulating the z^{-1} function, i.e. providing a delay of one sample time. In this case, only three values: y_{n-1}, x and x_{n-1} are affected by this instruction. In filters with many more terms, use of this instruction would result in a significant reduction in program size, and hence execution time, over transferring values from one memory location to the next sequentially via the accumulator.

```
 loc in co line  source      loc in co  line source

0000  4 1c    1. xio        0021  2 07   34. sub
0001  2 00    2. nop        0022  3 13   35. sr b4
0002  4 1c    3. xio        0023  2 00   36. nop
0003  2 00    4. nop        0024  3 10   37. sr b1
0004  4 1c    5. xio        0025  2 07   38. sub
0005  2 00    6. nop        0026  3 10   39. sr b1
0006  4 1c    7. xio        0027  2 07   40. sub
0007  1 e0    8. lrb 32     0028  3 10   41. sr b1
0008  3 11    9. sr b2      0029  2 07   42. sub
0009  1 62   10. sacc 34     002a  3 12   43. sr b3
000a  2 02   11. a0         002b  2 07   44. sub
000b  2 06   12. add        002c  3 11   45. sr b2
000c  3 11   13. sr b2       002d  2 07   46. sub
000d  2 06   14. add        002e  3 10   47. sr b1
000e  3 11   15. sr b2       002f  2 07   48. sub
000f  2 06   16. add        0030  3 10   49. sr b1
0010  3 10   17. sr b1       0031  1 54   50. sacc 20
0011  2 06   18. add        0032  1 e3   51. lrb 35
0012  3 11   19. sr b2       0033  2 04   52. ab
0013  2 06   20. add        0034  2 06   53. add
0014  3 10   21. sr b1       0035  1 e2   54. lrb 34
0015  2 06   22. add        0036  2 06   55. add
0016  3 11   23. sr b2       0037  1 e4   56. lrb 36
0017  2 06   24. add        0038  2 06   57. add
0018  3 12   25. sr b3       0039  3 13   58. sr b4
0019  2 06   26. add        003a  1 d5   59. lrb 21
001a  3 10   27. sr b1       003b  3 13   60. sr b4
001b  2 06   28. add        003c  2 06   61. add
001c  3 10   29. sr b1       003d  1 d4   62. lrb 20
001d  2 06   30. add        003e  2 06   63. add
001e  1 55   31. sacc 21     003f  2 0d   64. smem
001f  2 02   32. a0         0040  1 60   65. sacc 32
0020  1 e1   33. lrb 33
```

Fig. 5.12 SPE program for the filter of Appendix A

The HILO definition for the circuit INSTIN was produced in the same way as for the other simulations, that is automatically from the assembler object file, and configured to repeat the program ad infinitum. The circuit DATAIN was, however, slightly different. A program was written to produce a definition of DATAIN corresponding to one cycle of a sinusoid whose frequency is an integer multiple of the sampling frequency. The peak amplitude in each case was unity. The DATAIN circuit was in this case configured to repeat in the same way

as the INSTIN circuit, thus producing a repeating sinusoid. The simulation was stopped by putting a FINISH statement in the waveform file for the simulation at an appropriate time to permit the simulation of several cycles of the sinusoid. The waveform file was otherwise identical to that of figure 5.7. The output from the simulations came out as usual in the relatively incomprehensible HILO format. A program was written to read in this HILO format and print out the numbers as decimal fractions. The format chosen was identical to that output by the circuit simulator SPICE [28], in order that the graphical post-processor available for that simulator could be used.

Four simulations of the filter were performed, with input sinusoids of frequencies $\frac{1}{100}$, $\frac{1}{50}$, $\frac{1}{25}$ and $\frac{1}{10}$ of the sampling frequency. The filter is a low-pass filter with a cut-off frequency of $\frac{1}{50}$ of the sampling frequency, so these correspond to a half, one, two and five times the cut-off frequency respectively. The output of the filter simulations for each of these frequencies is presented graphically in figure 5.13. The peak amplitudes for these output sinusoids are approximately 0.48, 0.51, 0.28 and 0.04 respectively. At least the simulations show a low-pass filter! In fact, a simple pascal program was written to implement this digital filter, and there is very good agreement between the results obtained from this program and the HILO simulations. These simulations therefore verify that a processor based on SPEs can indeed implement difference equation type digital filters.

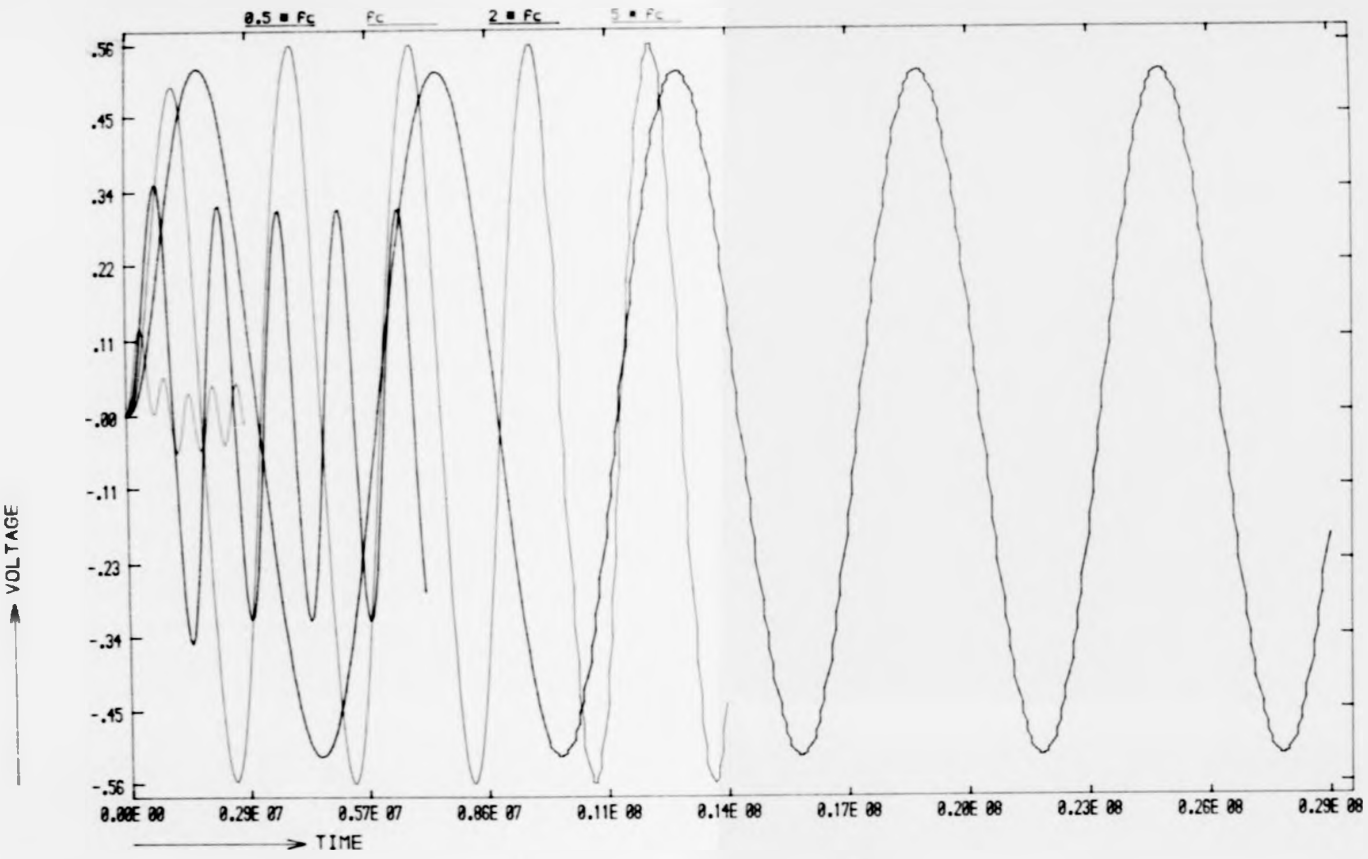


Fig 5.13 Filter output from HILO simulations at various frequencies

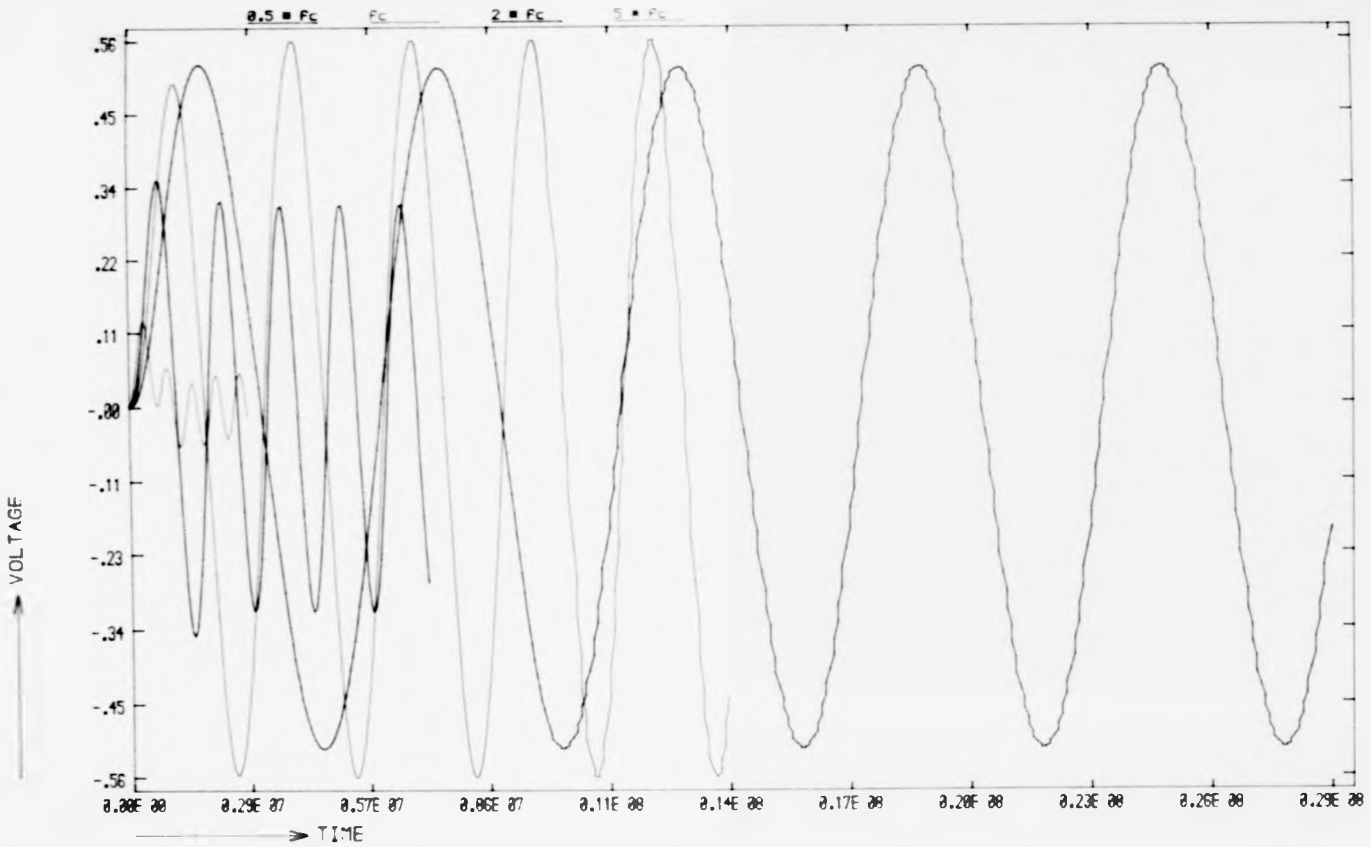


Fig 5.13 Filter output from HILO simulations at various frequencies

CHAPTER SIX

THE NMOS IMPLEMENTATION OF A SINGLE SPE

The theoretical studies so far described demonstrate that processors built from SPEs can indeed be used to implement digital filters. However, the aim of the project is not simply to verify the correctness of the design, but to make some assessment of its efficiency. This basically means providing the required performance at the lowest cost. Efficiency as it relates to signal processors is discussed in some detail in section 4.1, where it is seen that the main aim is to reduce silicon area, in particular the proportion occupied by the control logic and, to a lesser extent, memory. In order to obtain real data to assess the efficiency of SPE based systems, it is necessary to perform some detailed integrated circuit layout and circuit simulation. Fabrication of devices is desirable to verify the design and the accuracy of the simulations, but is not essential to the project. It has already been stated in chapter 5 that the system is likely to be implemented in CMOS, however the only technology available to the author at the time of the design was the 6 micron NMOS process at Edinburgh University. There was a possibility of fabrication if the design were performed for this process. Although a 6 micron process is clearly LSI rather than VLSI, this should not be a problem for prototype design; hopefully the eventual VLSI process would be significantly faster in operation as well as allowing a greater level of integration. The differences between NMOS and CMOS are sufficiently small that data on such matters as silicon area ratios should not be drastically different.

It was therefore decided to attempt the design of a single SPE in this technology with the hope of having it fabricated. In the event, the timescales for the available fabrication runs were such that only sections of the SPE could be fabricated.

An SPE is divided into to three main sections: the memory, the data path and the control logic. Salient points in the design of each of these will be considered in this chapter. The design of the control logic was not completed in time for fabrication and so the memory and data path sections were fabricated as individual chips. The provision of program memory is not considered at this stage; the problems associated with this are discussed in the next chapter.

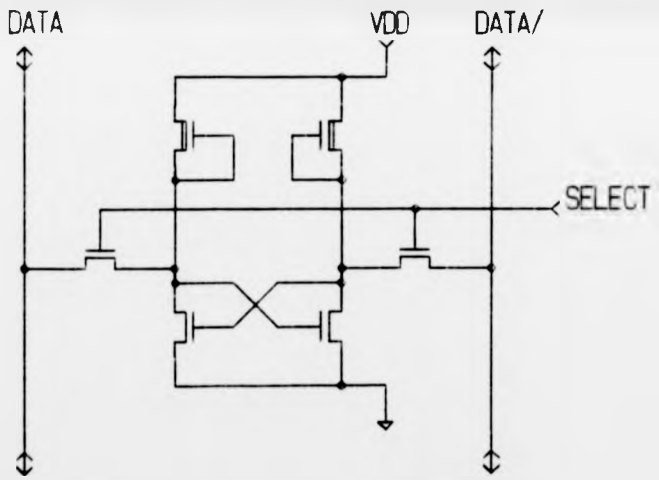


Fig. 6.1 Circuit diagram of a NMOS static RAM cell

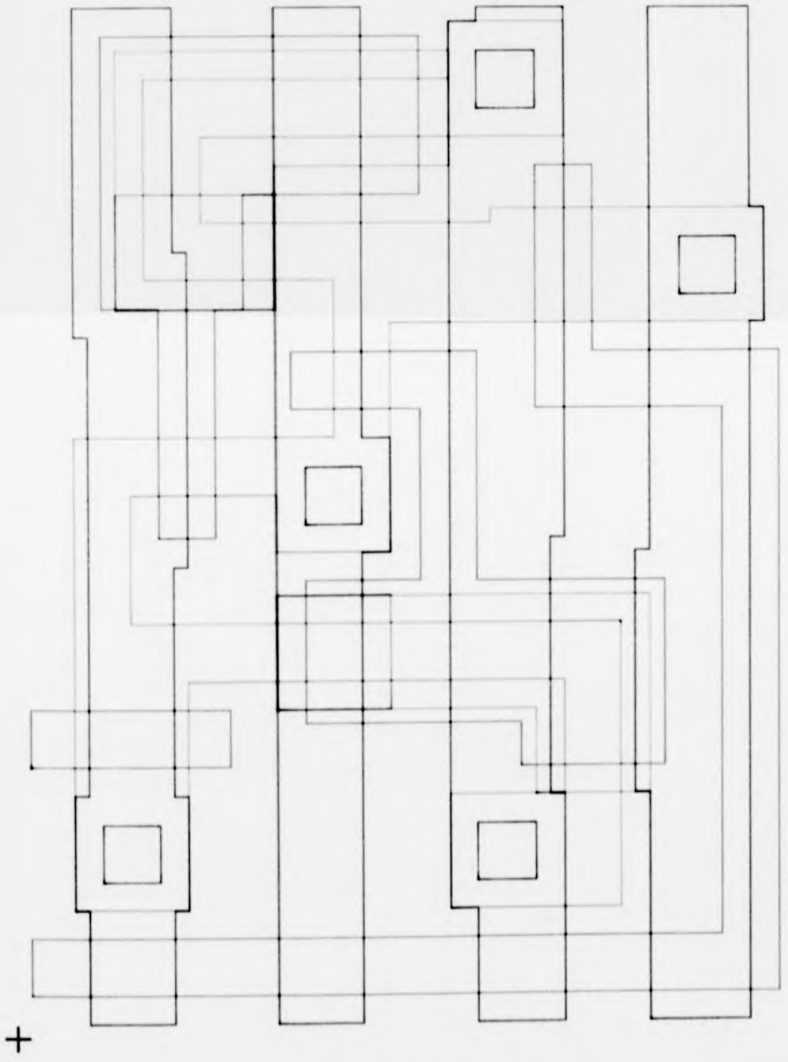


Fig.6.2 Possible layout of a NMOS static RAM cell

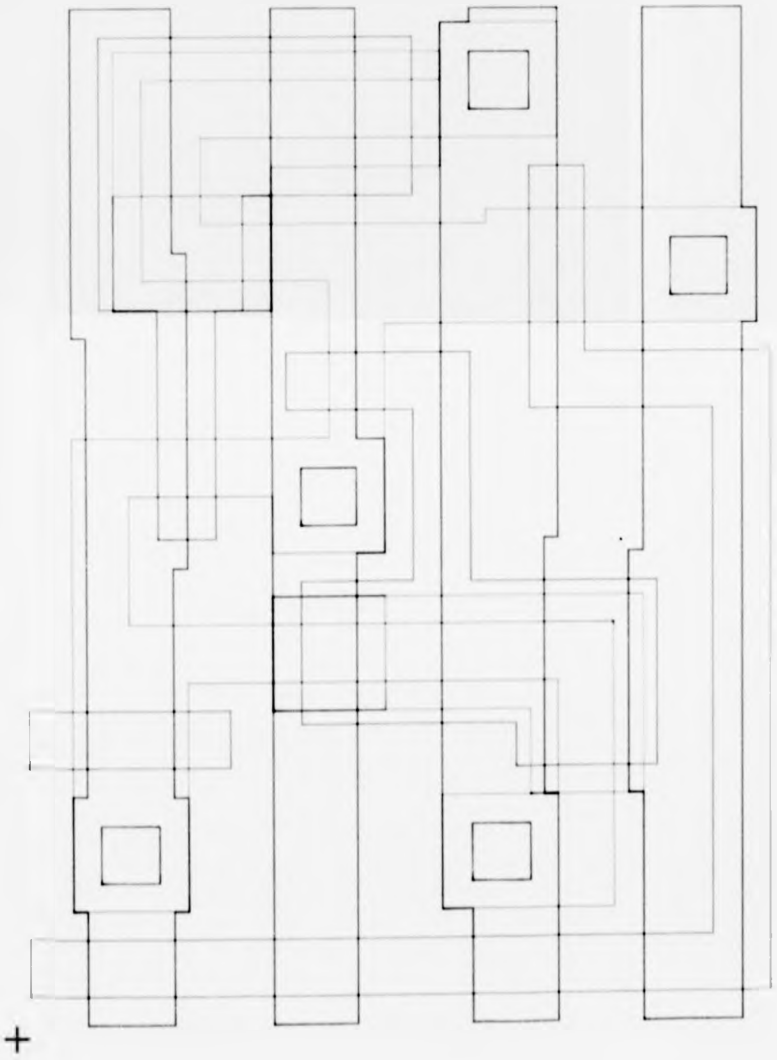


Fig.6.2 Possible layout of a NMOS static RAM cell

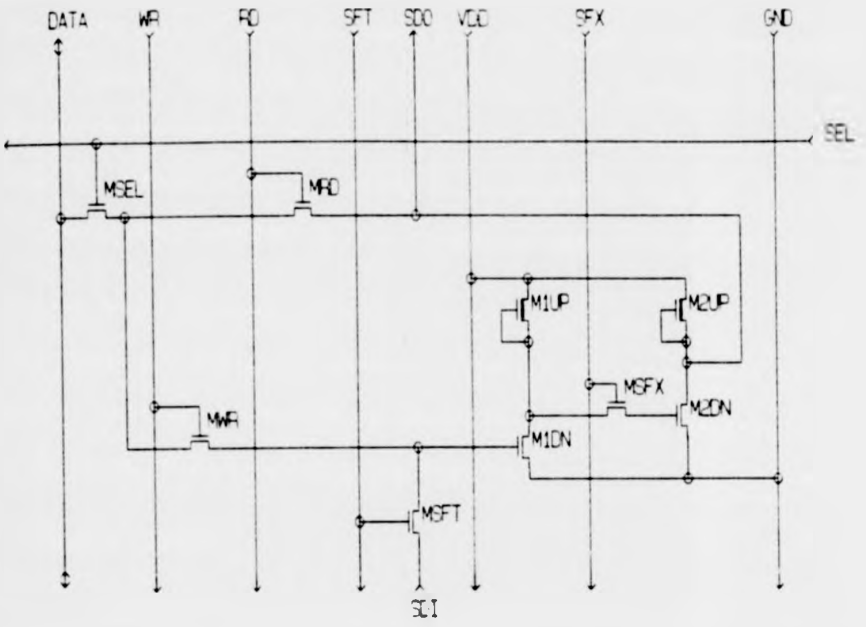


Fig. 6.3 Circuit diagram of a shiftable RAM cell

6.1 MEMORY DESIGN

Each SPE is to incorporate 64 nibbles (4-bit words) of read-write memory. This small quantity of memory is clearly best implemented as static RAM: the additional complex circuitry associated with dynamic memories would outweigh any potential advantages.

The circuit diagram of a 6-transistor NMOS static RAM cell is given in figure 6.1. Detailed descriptions of various kinds of read-write memory can be found in Weste and Eshragian [30 pp348ff] and in Mavor, Jack and Denyer [31 pp127ff]. A possible layout for such a RAM cell is given in figure 6.2; the colour code used for all layout diagrams in this thesis is that used by Mead and Conway [25 p64]. The cell measures 78.0 by 106.5 micron, therefore a memory consisting of 256 such cells (64 by 4) would occupy 2.13 square mm.

6.1.1 Design of a Shiftable Memory

As discussed in section 4.5, it was decided to investigate the possibility of providing the z^{-1} operation in one cycle by enabling the entire memory to be shifted by one location in a single cycle. This clearly cannot be achieved using the standard memory data lines but implies some local intelligence in the memory. After some consideration it was realised that it was impractical to base a memory cell with such capability on a simple static RAM cell. Instead, a cell based on a modification of a semi-static register [25 p70] was developed, and the circuit diagram is given in figure 6.3 with a possible layout in figure 6.4. This cell supports several modes of operation. The cell is based around two inverters formed by transistors M1DN, M1UP M2DN and M2UP, and a few pass transistors. When the memory is inactive, not reading, writing or shifting, data is maintained in the cell by the feedback path provided by transistors MSFX, MRD and M&R. Data is written using transistors MSEL and M&R and read using MSEL and MRD. The gate of MSEL is connected to the horizontal select line and is used to select the appropriate row of the memory array for reading or writing. The shifting of memory data by one location is achieved using transistors MSFT and MSFX. With M&R and MRD help off, these allow an entire column of the memory to be connected as a shift register.

This memory cell consists of 9 transistors rather than 6 for a standard RAM cell, and what is more important requires 8 vertical metal tracks through each cell rather than 4. The size of the layout of figure 6.4 is 174 by 82.5 micron, therefore a memory consisting of 256 such cells would occupy 3.67 square mm.

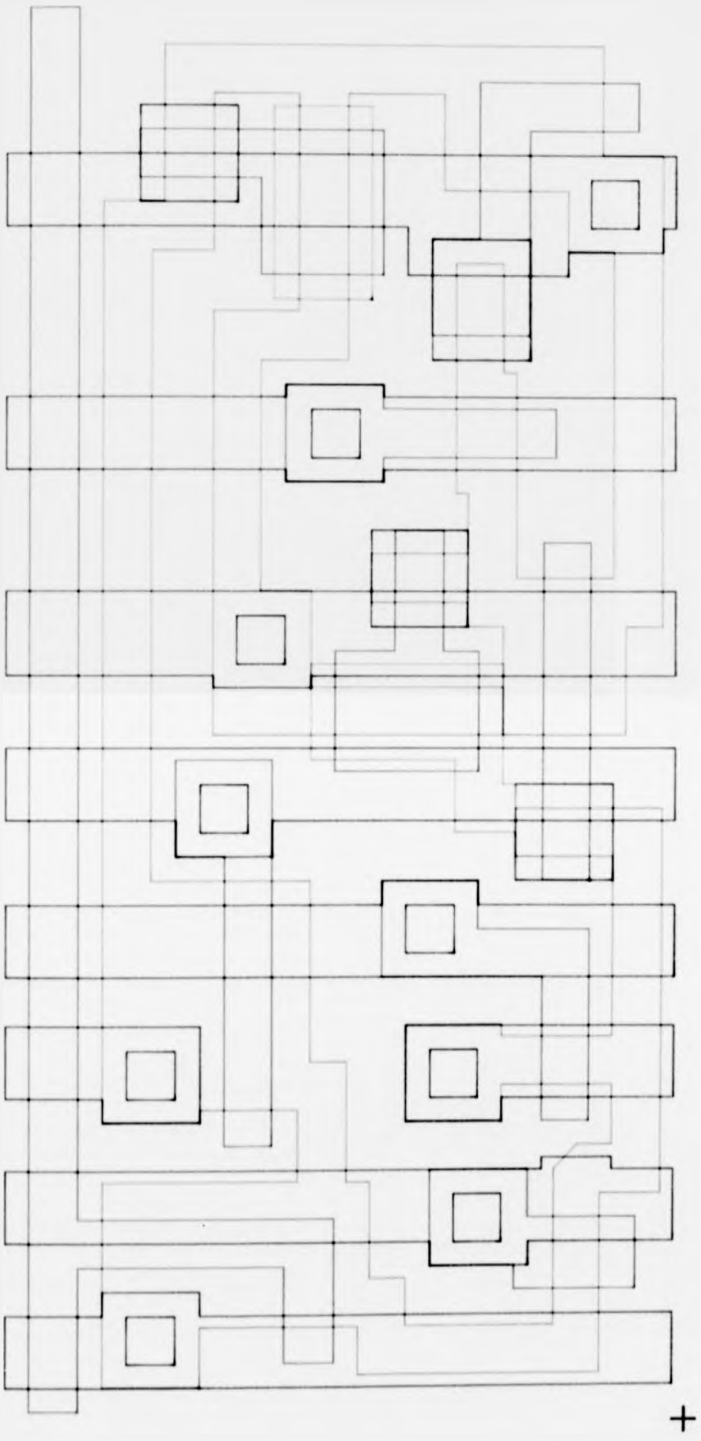


Fig.6.4 Layout of a shiftable RAM cell

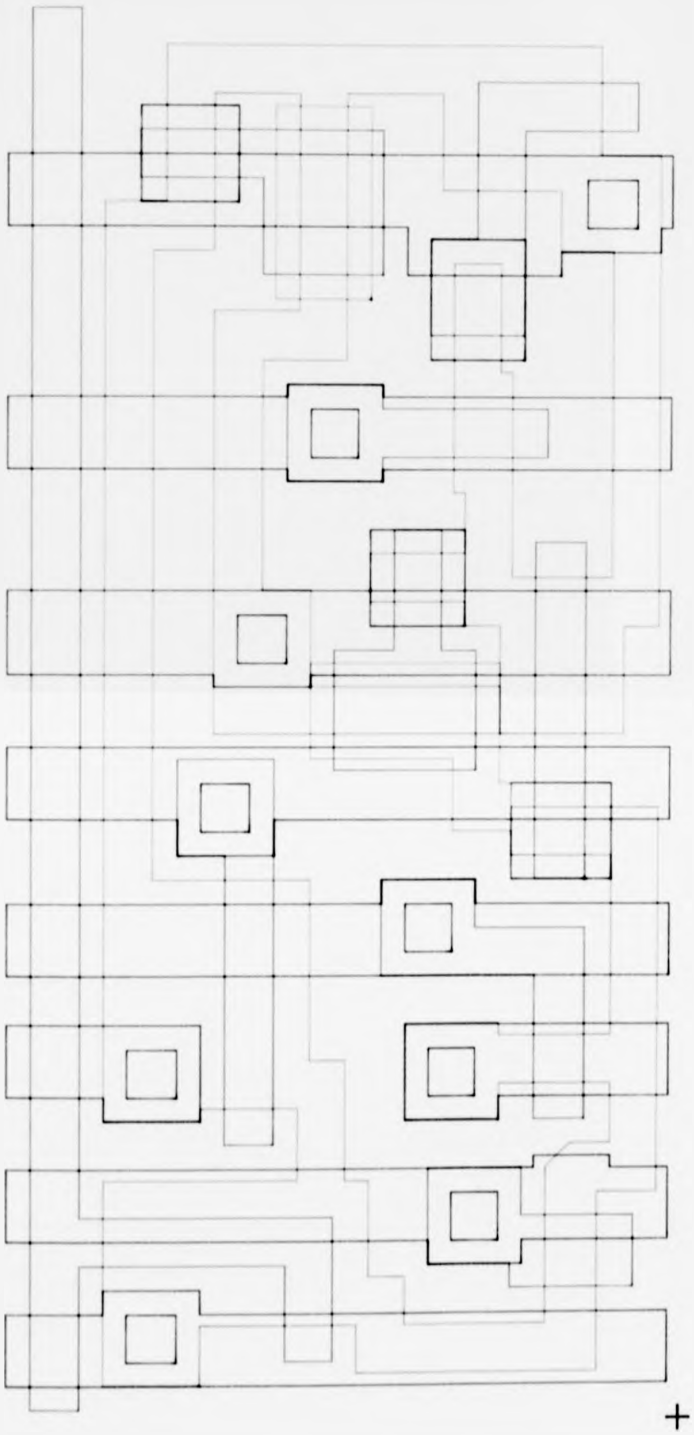


Fig.6.4 Layout of a shiftable RAM cell

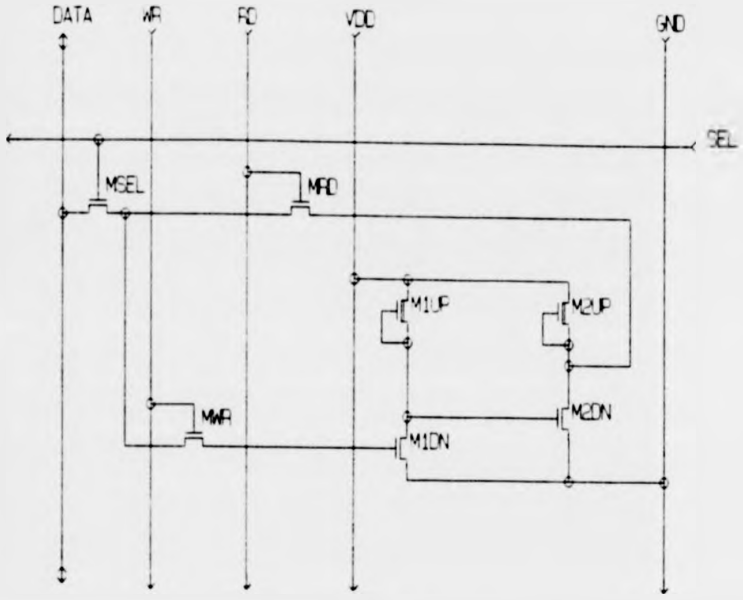


Fig. 6.5 Circuit diagram of a non-shiftable RAM cell

It has been decided (qv section 4.7) only to provide this feature for half the memory, on the grounds that some of the memory should remain in place for the storage of fixed data such as coefficients. A memory consisting of both types of cells described above would be difficult to implement due to the different means of controlling each. It was therefore decided to design a simpler, non-shiftable version of the shiftable memory cell which would share the same control methodology. A circuit diagram for such a cell is given in figure 6.5 with a possible layout in figure 6.6. This cell is naturally smaller than the shiftable cell, occupying an area of 133.5 by 82.5 micron, although it is still larger than a conventional static RAM cell. The height of this cell and of the shiftable cell are identical, which is essential to the sensible construction of the memory cell array. The area of a cell array consisting of 128 shiftable

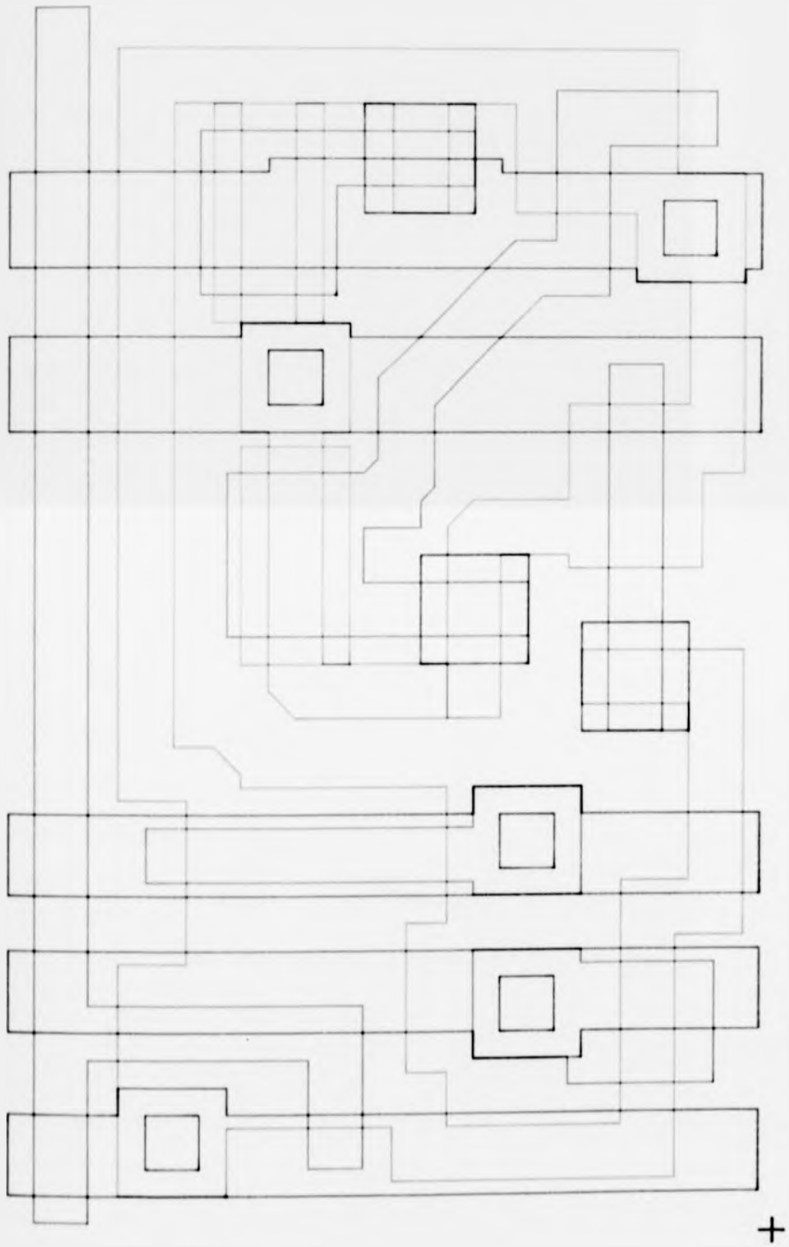


Fig.6.6 Layout of a non-shiftable RAM cell

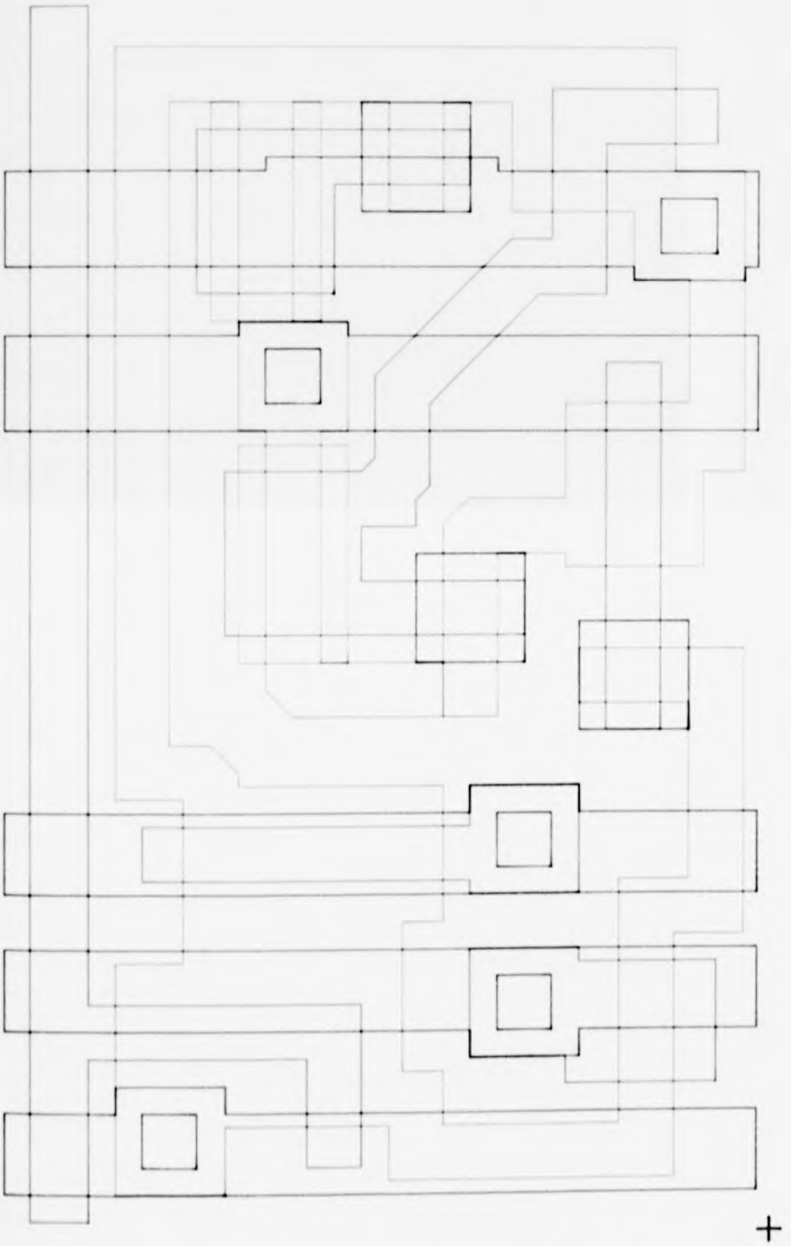


Fig.6.6 Layout of a non-shiftable RAM cell

cells and 128 non-shiftable cells will be 3.25 square mm.

6.1.2 Memory Peripheral Circuitry

Unfortunately, the design of a memory does not only consist of the design of the cells. They not only need to be put together in an array, but also peripheral circuitry such as address decoders and data drivers are required. In this case, fortunately, only one level of column multiplexing will be required: conceptually the memory is 64 by 4 bits and a reasonable aspect ratio for the memory can be achieved by using 32 cells by 8 cells. Minimising the column multiplexing was taken into account when choosing the aspect ratio for the cells themselves; 82.5 microns was found to be the smallest height consistent with a small overall area. In fact, the column multiplexers will be used to choose between the shiftable and non-shiftable halves of the memory. It would not make any sense to put shiftable and non-shiftable cells in one column due to their different size in the x-direction. However, having all the shiftable cells for each bit in the same column avoids awkward routing of data from the top of one column to the bottom of the next. This means that 32 by 8 is the ideal configuration for our purposes.

6.1.2.1 Row address decoding and select line driving

The most complicated memory peripheral circuitry is associated with the decoding of row addresses to drive the select lines, also known as word lines, which run across the memory array. In this case, 5 bits of the memory address are decoded by the row decoders to drive the 32 select lines. It was decided to use a simple NOR-gate decoder with a super-buffer [25 p17] driver. The capacitance of the

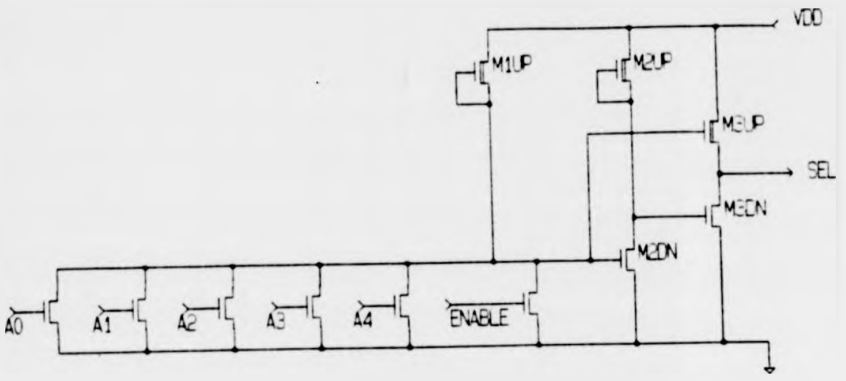


Fig. 6.7 Circuit of address decoder and select line driver

select lines is quite high, due to the long polysilicon word line and the eight transistor gates to which it is connected. However the resistance of the line, being in polysilicon rather than metal, was the limiting factor in speed rather than the power of the select line driver. The circuit for a single address decoder is given in figure 6.7 with an example layout in figure 6.8. Note that the height of this cell is identical to the height of the memory cells, for pitch-matching purposes.

Address line drivers are also required to provide the true and inverse version of each of the five address bits, with sufficient drive capability for 16 address decoders. It is desirable for these drivers to pitch-match with the address decoders to avoid routing, but this was not found to be practical. Once again, super-buffers were chosen to provide this function.

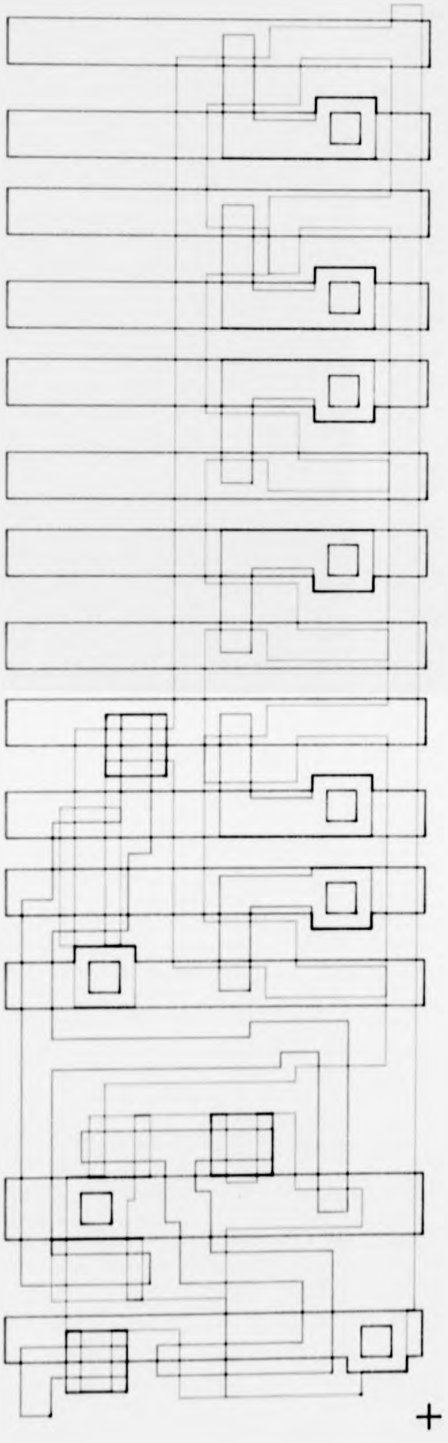


Fig.6.8 Example layout of address decoder and select line driver

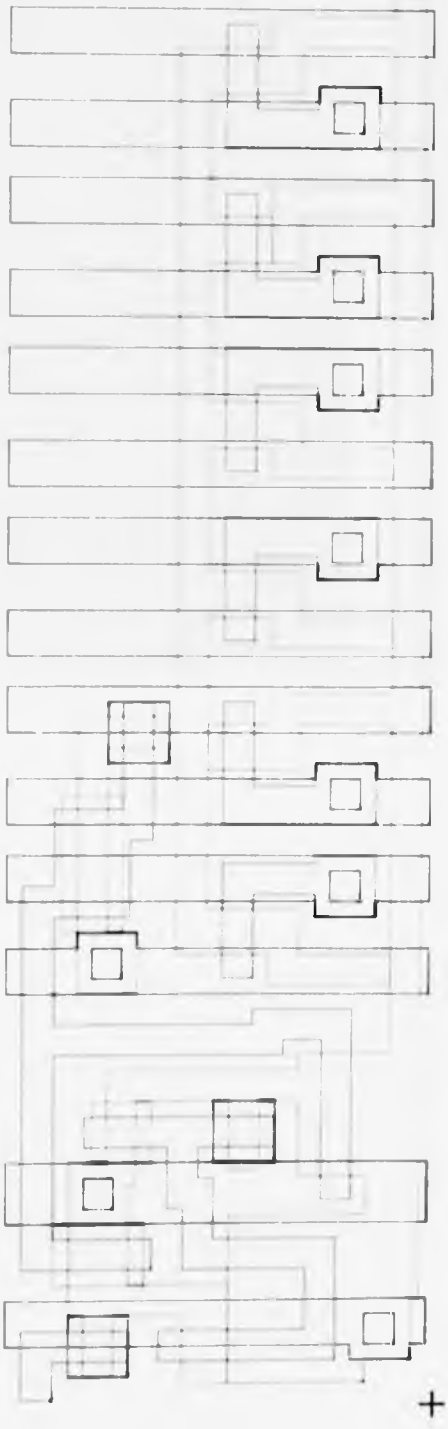


Fig.6.8 Example layout of address decoder and select line driver

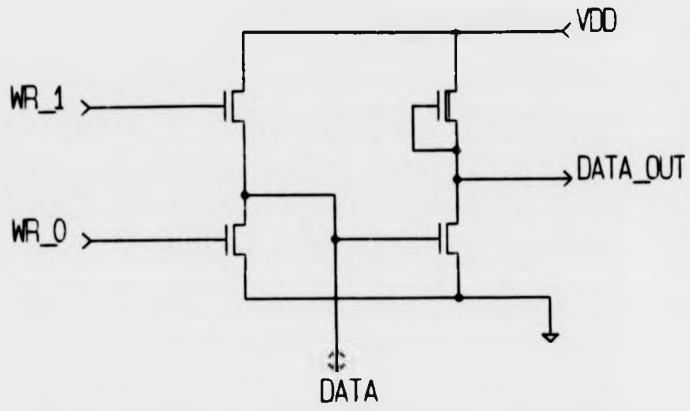


Fig. 6.9 Circuit of bidirectional data line driver

6.1.2.2 Data line driver circuit

The data lines are bidirectional, being used both to write data into the memory and read it out. Hence bidirectional drivers are necessary for the data lines. To provide bidirectionality, it is necessary to use enhancement mode devices both to pull up and pull down the data lines. The depletion mode pull-up devices normally used in NMOS cannot be turned off and are therefore unsuitable. The use of enhancement-mode pull-ups limits the voltage which can be realised on the data lines and reduces the pull-up speed.

The circuit of the data line driver can be found in figure 6.9, with the layout in figure 6.10. During a memory write cycle, either the pull-down or pull-up device will be turned on, writing a 0 or 1 respectively into the selected memory cell. During a read cycle, both these devices will be turned off, and the value on the data line will be that stored in the selected cell. In fact, prior to the read, the data lines are pre-charged to an intermediate voltage by turning on both devices, to decrease the response time. A simple

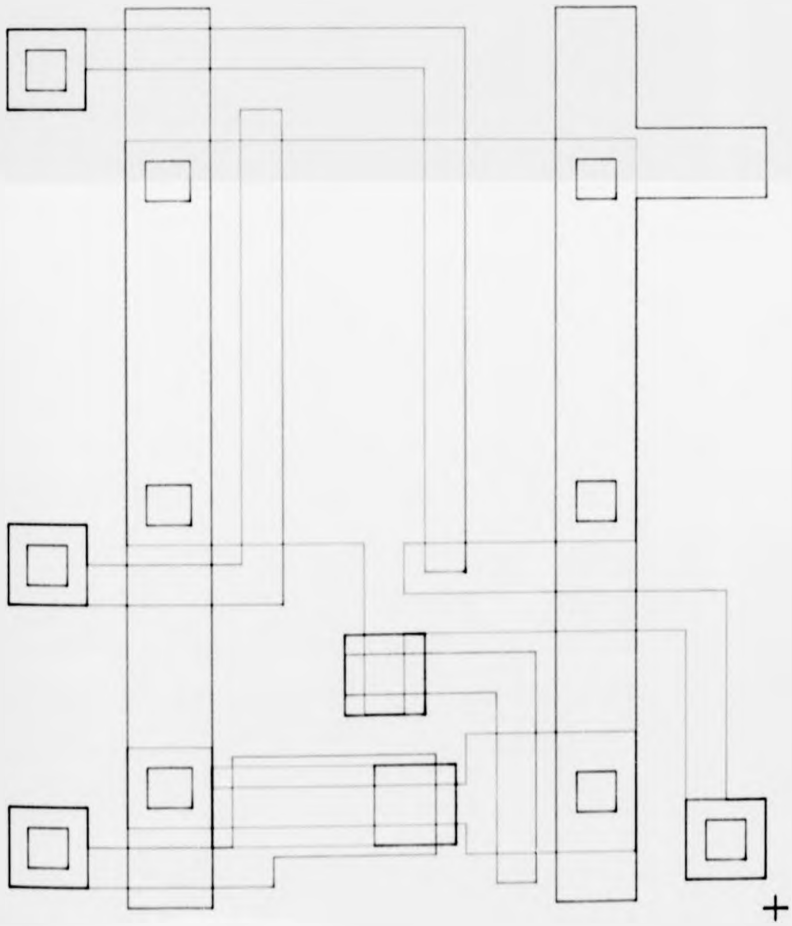


Fig.6.10 Layout of bidirectional line driver

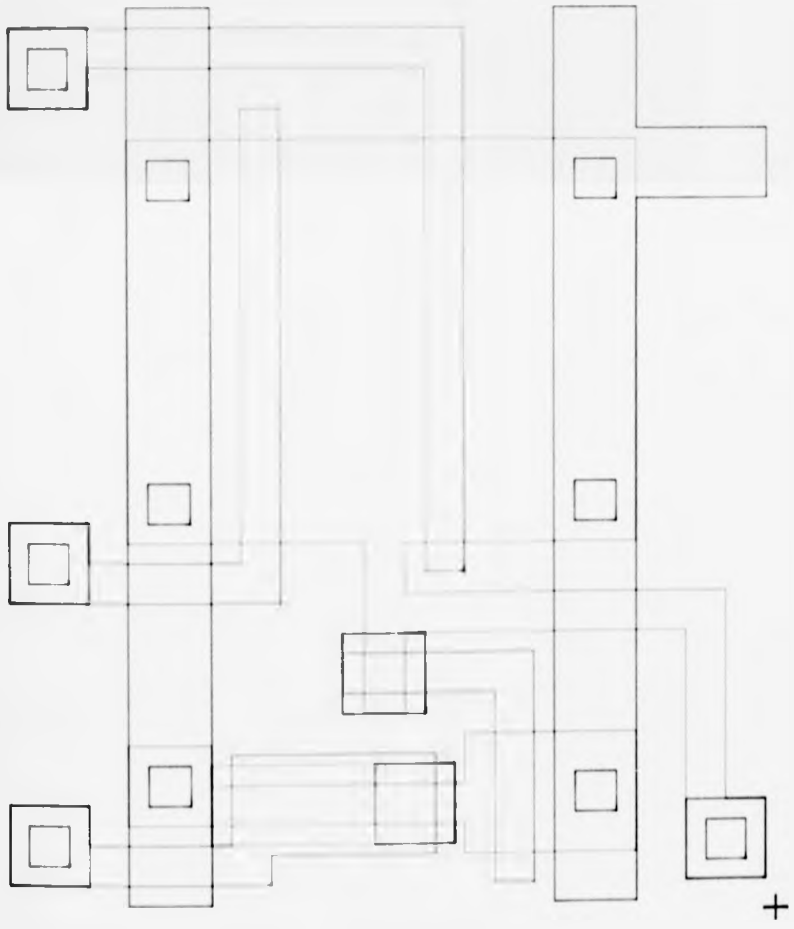


Fig.6.10 Layout of bidirectional line driver

inverter is used to buffer the value on the data line to the outside world. A more complex sense amplifier could be used in place of this inverter to further reduce the response time. This was not implemented on the test chip due to simulation problems and lack of time. A suitable circuit for a sense amplifier, intended for a dynamic RAM but which could be adapted for use in this case, can be found in Mavor, Jack and Denyer [31 p134].

It was later realised that the method adopted for the precharging of the data lines to an intermediate value was not ideal. This would be much better achieved by using the crossover point of the sense amplifier to establish the precharge voltage. In this case, where the sense amplifier is a simple inverter, this can be done by connecting its input and output together. This voltage would be applied to the data lines through a further pass transistor.

6.1.3 Circuit Level Simulation of the RAM

Circuit level simulations of various parts of the RAM have been performed using the circuit simulator SPICE2G.5 developed at the University of California, Berkeley. These simulations verify the functionality of the cells and provide performance estimations for the memory as a whole. It would be impracticable to include results from all the simulations within this thesis, however the results from an attempt to realistically simulate a model of the entire RAM are included.

It is worth pointing out at this point the extreme difficulty found by the author in obtaining results from the SPICE simulator. More often than not, the simulator would fail to converge on a result

either during the initial dc analysis of the circuit or at some time during the transient analysis. This experience is common to all those involved in MOS IC design at UMIST and to many elsewhere known to the author, and indeed appears to be significantly worse for CMOS circuits than the NMOS circuits considered here. Some proprietary simulator products are available which claim to have solved this problem, however they were not available to the author. Consequently much time was wasted in trying to get simulation results, and the results obtained were not as comprehensive as would be desirable. The problems were particularly acute with larger simulations, with the effect that the author had to be content in the main with simula-

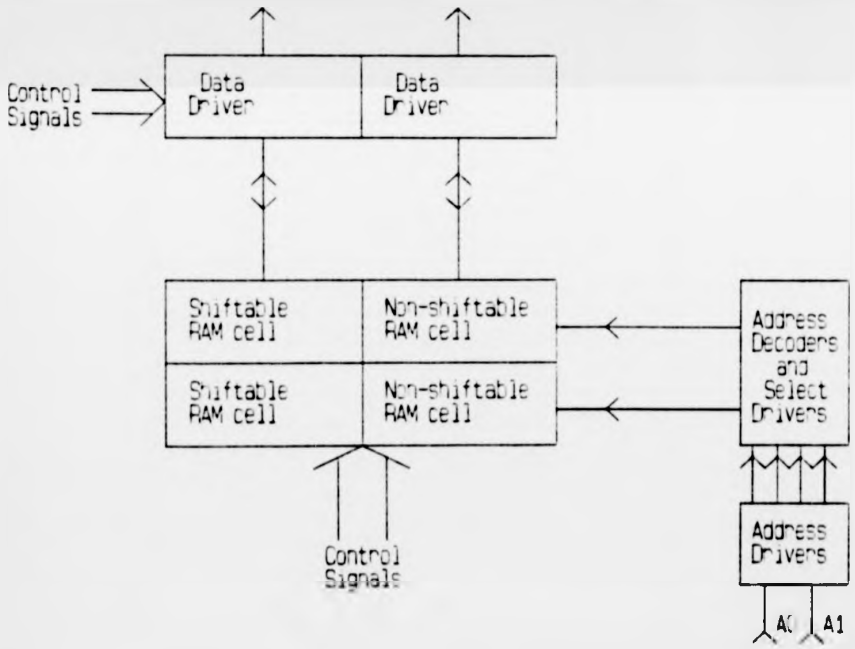


Fig. 6.11 Block diagram of model of RAM circuit simulated

tions of circuits consisting of only a few transistors.

Even were the simulator reliable, it would not be reasonable to expect to simulate the entire RAM at once. The computer time

involved in circuit simulation on such a large scale would be prohibitive. A model was therefore devised which would test out each individual cell of the RAM, including capacitive and resistive loads to represent the rest of the circuitry. A block diagram of this model can be found in figure 6.11. The simulation was built around four memory cells - two of the shiftable variety and two non-shiftable, in a two by two arrangement. Thus two address decode and select line drive blocks were required along with two data line drivers. There is therefore one data line for the shiftable cells and one for the non-shiftable. Address line drivers were included for the active address lines A0 and A1. The cells being simulated occupied locations 0 and 1, therefore these address lines give the possibility of selecting either or neither used location.

This circuit, although considerably simplified from the real RAM circuit, proved still very difficult to simulate. The author spent many fruitless hours and days making small modifications to the circuit and to some parameters to SPICE in attempts to obtain results. Eventually a result was obtained, thanks to advice from the SPICE support staff at the SERC Rutherford Appleton Laboratory, but this was several months after the first attempt to simulate the circuit! The results of this simulation are presented graphically in figures 6.12 and 6.13. The memory operates in two clock phases, in common with the rest of the SPE. The first is used for reading and writing, and the second for refreshing and shifting. The simulations presented in the diagrams represent seven clock cycles. The first two are write cycles for addresses 0 and 1 respectively and the subsequent two are read cycles for the same addresses. The fifth cycle is a shift cycle and the final two are further read cycles for the

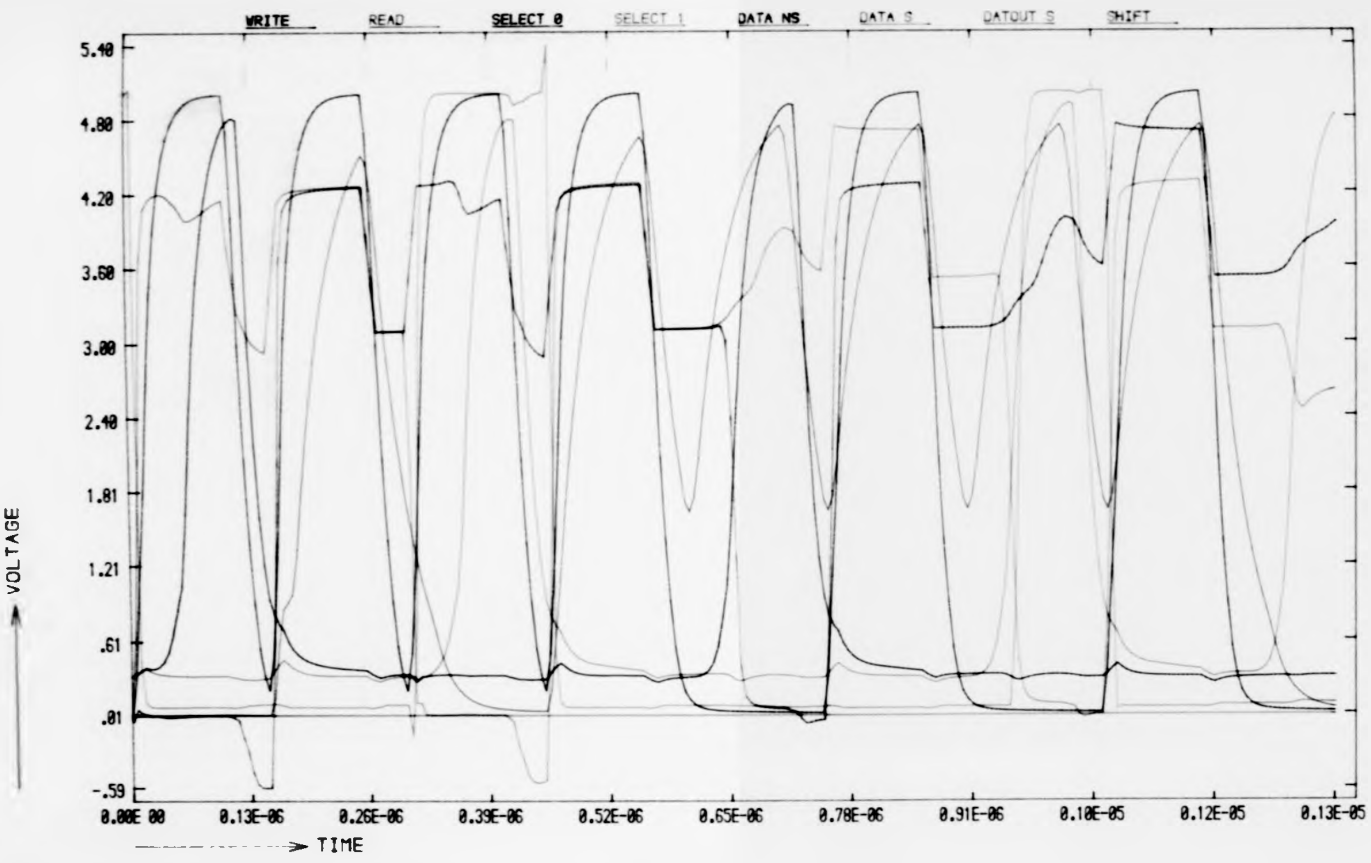


Fig 6.12 SPICE simulation results for RAM - part 1

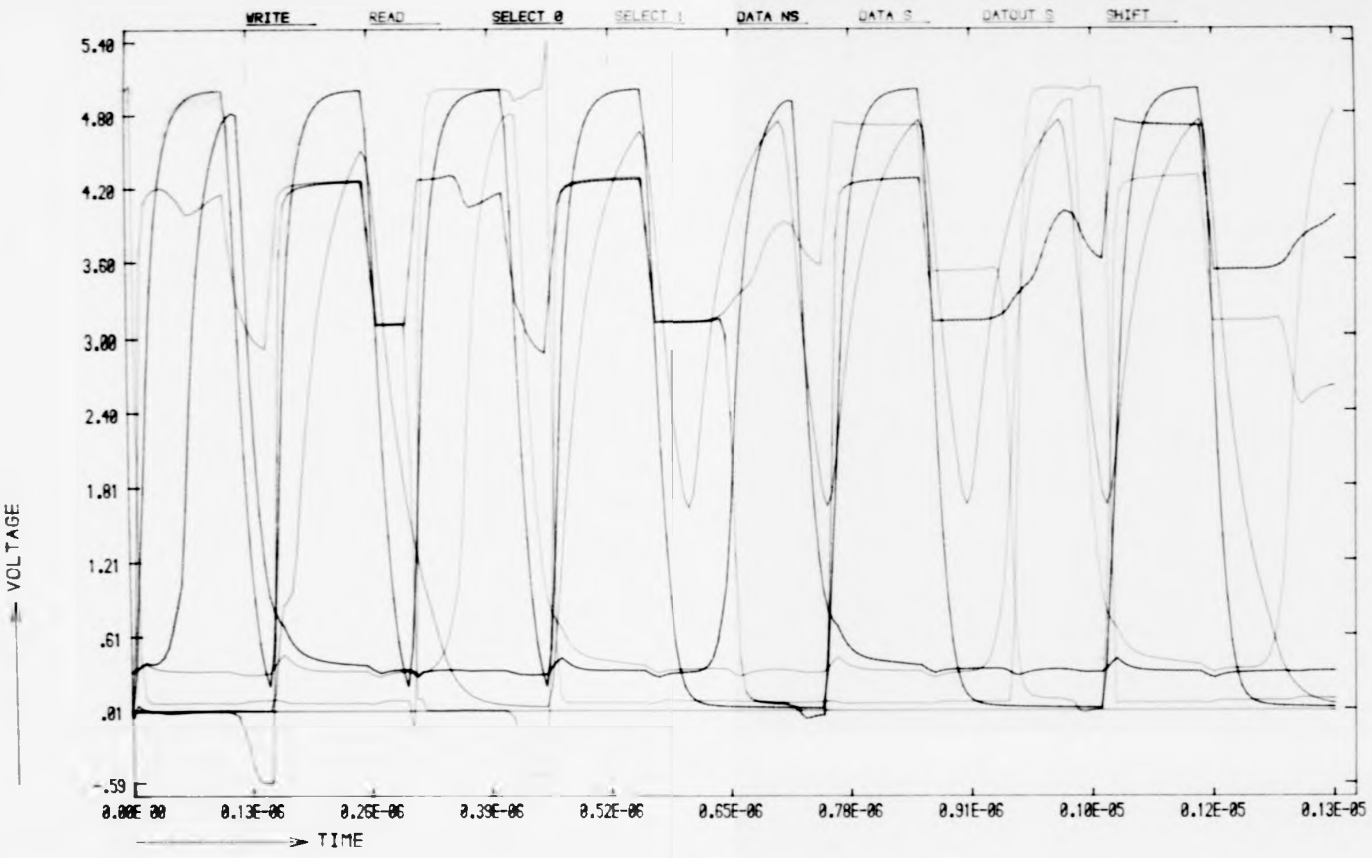


Fig 6.12 SPICE simulation results for RAM - part 1

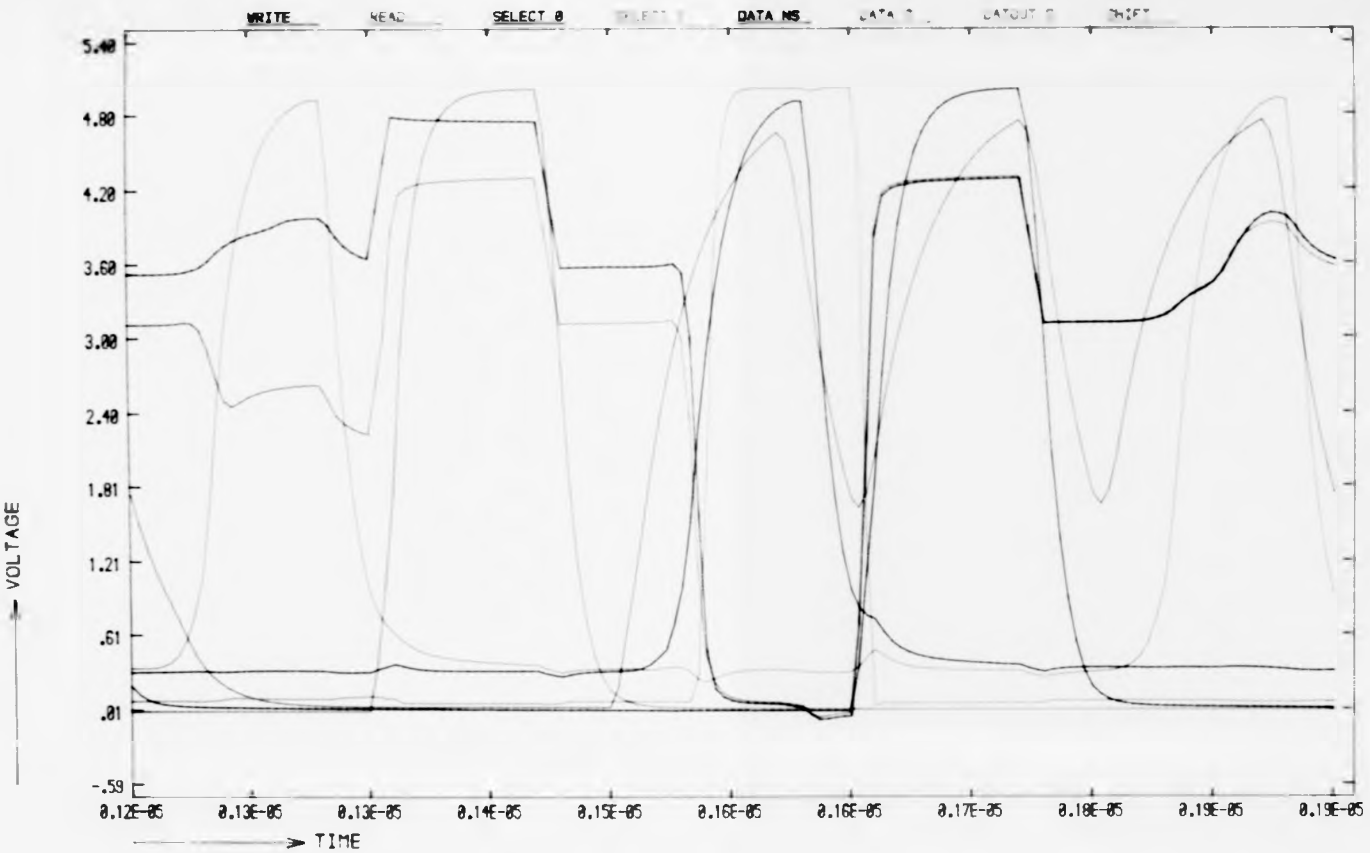


Fig 6.13 SPICE simulations results for RAM - part 2

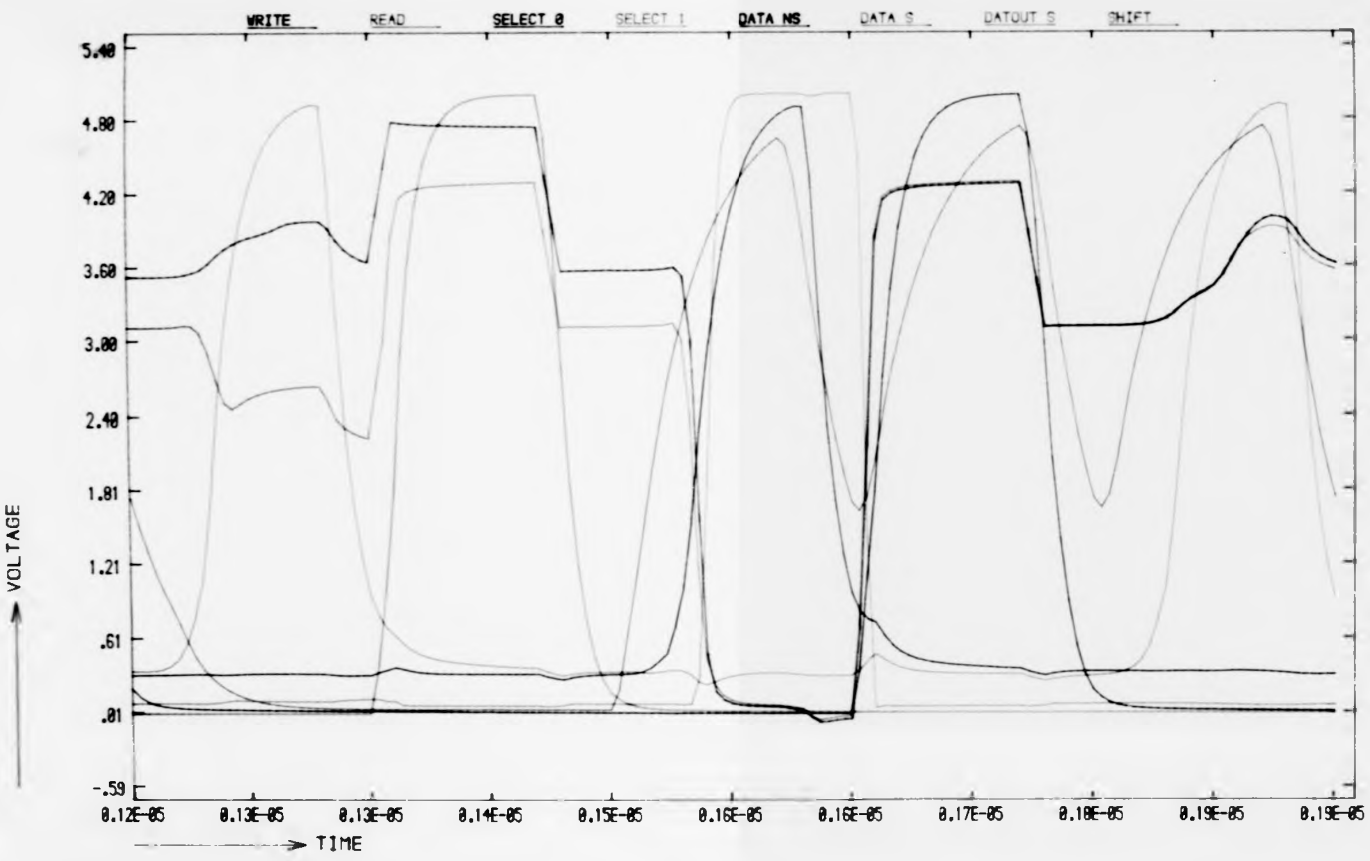


Fig 6.13 SPICE simulations results for RAM - part 2

two addresses. This simulation clearly demonstrates that data can be read and written to cells in the memory and verifies the correct operation of the memory shift. One interesting point to notice is the way the inverter on the data lines cleans up the data line waveforms. The waveform marked "DATA S" represents the actual data line through the shiftable RAM cells and that marked "DATOUT S" represents the output from the inverter whose input is "DATA S". Whereas the waveform "DATA S" is very messy, that of "DATOUT S" shows clean transitions from high to low voltages. This simulation also shows that the limiting factor in the speed of the memory is the delay in the select lines. This is mainly due to the high resistance of the polysilicon word lines, although it also depends on the delay through the address decoders and select line drivers themselves. The overall delay observed in the select lines is about 50ns., with very little delay due to the changeover of the data output line for a read cycle.

6.1.4 Layout of a Test Chip

As has been explained, separate test chips were fabricated for the memory and data path sections of an SPE. The control portion was never fabricated. Although this subdivision arose initially due to lack of time because of the fixed deadline for the fabrication run, in many ways it was an advantage. In particular, it eased the problem of testing the fabricated chips.

It was decided to fabricate as simple a chip as possible to give the best chance of obtaining working parts; that is it was decided to include as little circuitry as possible whose purpose was purely for the test chip and would have no use in a real SPE. In view of this,

the three connections to each data drive block were taken straight to the outside world via the appropriate drive or protection circuitry. These consist of two inputs, the write 0 and write 1 signals and one output. The address lines were fed directly from the input pads to the address line drivers. The main read, write, shift and other control signals were taken straight from the input pads to the memory cell array. This was later realised to be a mistake, as the high capacitance associated with these lines in conjunction with the resistance imposed by the input protection circuitry (about 3 kiloha), could compromise the operating speed of the memory.

In view of the small scale possible, a full colour plot of the test chip was not considered appropriate. However, a monochrome version can be seen in figure 6.14, and some salient features can be picked out. Eight columns of RAM cells are visible, alternating between shiftable cells and the narrower non-shiftable cells. The address decoders can be seen at one side of the memory array. At one end are the data line drivers and at the other end, displaced to one side are the address line drivers. The plot serves at least to give some impression of the space taken up by the different parts of the memory.

6.1.5 Testing the Fabricated RAM Chip

Having been able to arrange fabrication of the RAM test chip, it was possible to perform some tests. The main concern was to establish that the memory did indeed function as intended and to verify the performance indicated by the SPICE simulations. However, it must be emphasised that the main reason for performing the layout was not to obtain fabricated chips but data on the silicon area occupied by

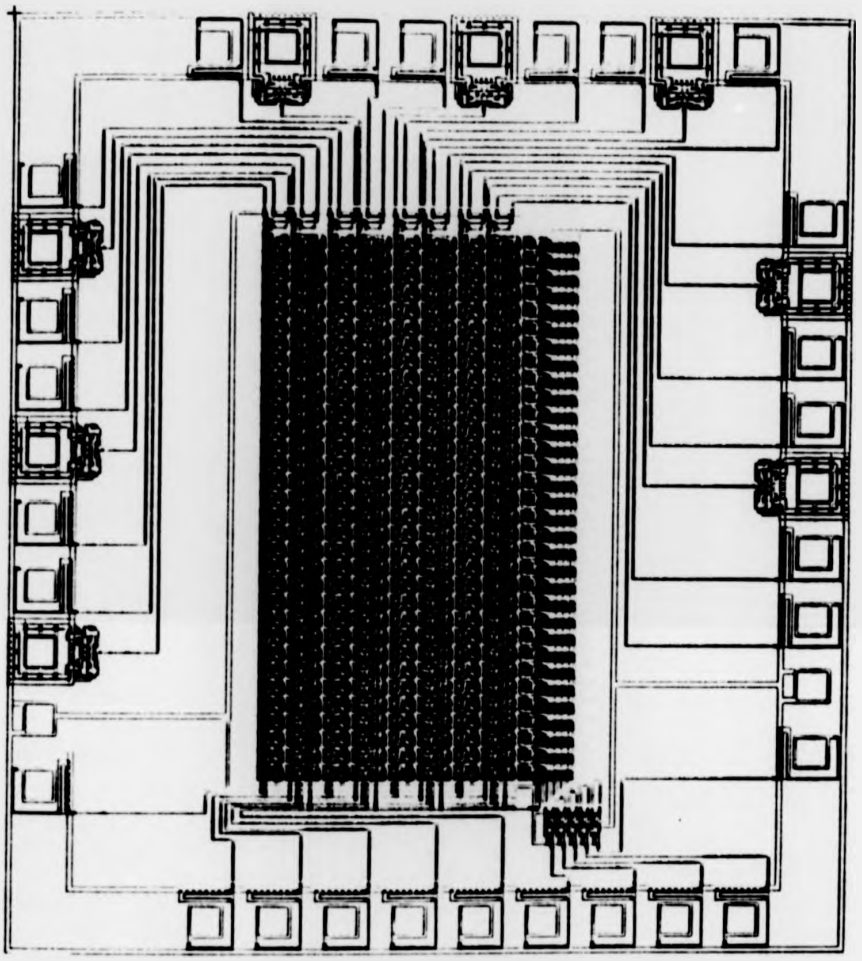


Fig. 6.14 Layout of RAM test chip

various different parts of the overall design.

While establishing the test which would be required, it was realised that a faulty connection had inadvertently been made. The SFX line to all the shiftable RAM cells (see figure 6.3 for identification of this line) was connected to the ENABLE line of the select line drivers. In the operation of the RAM, the SFX line is supposed to be high during ø1 and low during ø2. The select lines should be

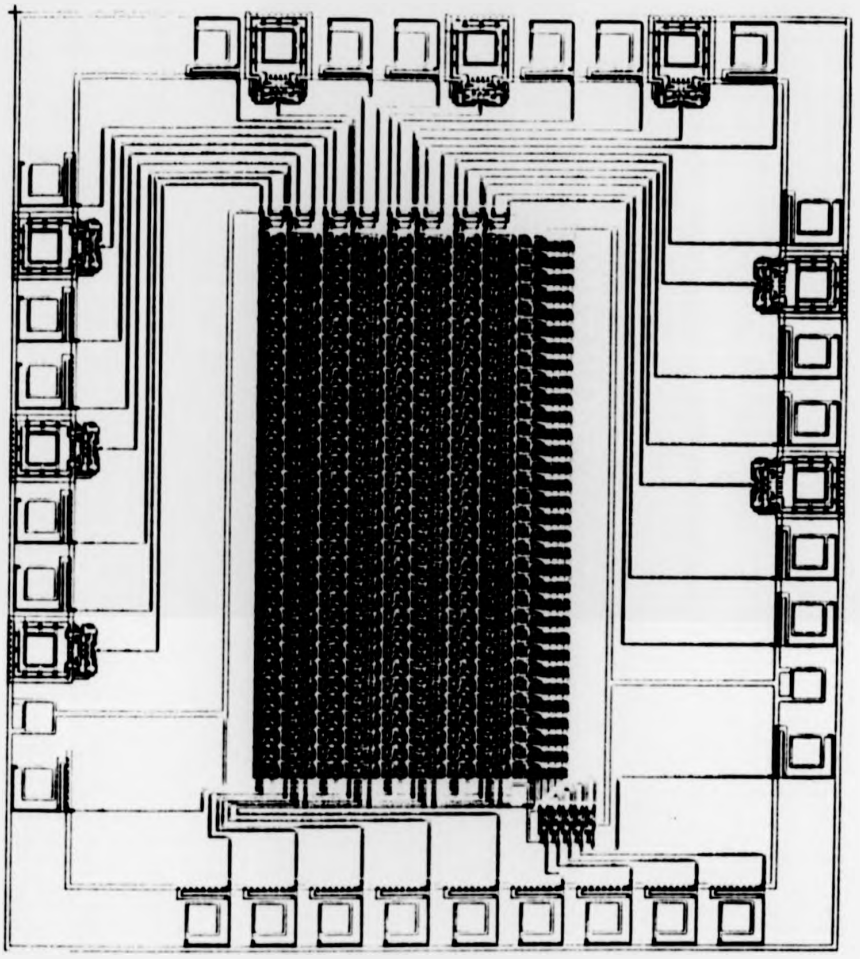


Fig. 6.14 Layout of RAM test chip

various different parts of the overall design.

While establishing the test which would be required, it was realised that a faulty connection had inadvertently been made. The SFX line to all the shiftable RAM cells (see figure 6.3 for identification of this line) was connected to the ENABLE line of the select line drivers. In the operation of the RAM, the SFX line is supposed to be high during Ø1 and low during Ø2. The select lines should be

enabled during Ø1 and disabled during Ø2, however the sense of the ENABLE line is inverse, meaning that SELECT should be the inverse of SFX, rather than identical to it. A workaround is possible by adding a third phase between Ø1 and Ø2 during which SFX can be brought high. In this case SFX can be kept high during Ø2 without risk of corrupting the stored data, which would otherwise be a possibility. During shift cycles, SFX must still be kept low during Ø2, however the resultant enabling of the select lines will not interfere with memory operation as it would in a refresh cycle. As always envisaged, the data lines will be pre-charged to an intermediate voltage during Ø2 in anticipation of a possible read operation during the next cycle.

The test results obtained were disappointing: basically the chips did not work. Very few of the data lines on any of the chips showed any change in output, most being stuck at high or low. Those few which did actually change state did not seem to do so in any sensible manner. During a write cycle at least, one could expect the data lines to show a one or a zero depending on whether a one or a zero was being written. This however was not the case. Indeed, the data lines which seemed closest to working showed a logic zero whenever data was being written and a logic one whenever data was being read. In such circumstances, further testing was impractical. The variations in behaviour between chips and between data lines within a particular chip would seem to indicate processing problems, although it would be unfair to attribute blame without further investigations.

This experience serves to highlight the problem of testing chips which apparently do not work at all. It is extremely difficult to distinguish between processing problems and some fundamental design error, as access to internal nodes is impossible. However, although

the negative test results are disappointing, the essential data from the design exercise, namely the silicon area used, is still valid. It is hardly likely that any design errors could be so fundamental as to significantly affect the silicon area.

6.2 DATAPATH DESIGN

The second major part of an SPE is the datapath. This consists of three main sections: an arithmetic unit capable of performing addition and subtraction, a 4-bit barrel shifter, and some registers. The floorplan of the datapath is traditional, consisting of four identical bit slices connected together by abutment. The only exception to this is the barrel shifter, although the only difference between the bit slices of the barrel shifter is the position of one contact. All the individual cells which go together to form the datapath are therefore the same height. In general, the control signals run vertically across all the bitslices in the datapath and the data signals horizontally along each bit slice. There are obvious exceptions to this such as the carry signal in the arithmetic unit and some of the data in the barrel shifter, both of which must run vertically.

6.2.1 Design of an Arithmetic Unit

The basic requirement for the arithmetic unit is to perform addition and subtraction, taking its inputs from the accumulator and register RB and returning the results to the accumulator. This is easily achieved by providing an adder, either of whose inputs can be inverted, set to zero or used as they are. Such an arithmetic unit can perform all the arithmetic instructions listed in Appendix C

except those which alter the value of the B register. These will have to be provided elsewhere in the datapath.

The adder implementation decided upon was based on the Manchester carry chain [25 p150]. This uses a single pass transistor in each adder to effect the carry propagation and therefore is much faster then a ripple carry adder based on conventional logic gates. The complexity of carry look-ahead logic is avoided, although for long word-length adders there is a very simple carry look-ahead scheme which can be implemented in conjunction with a Manchester carry chain [30 p325]. This is, of course, unnecessary for a 4-bit adder for which a simple Manchester carry adder is ideally suited. The adder is a dynamic circuit, all the carry signals being pre-charged during one of the clock phases, in this case Ø2. During Ø1, the evaluate phase, two signals KILL and PROPAGATE are established from the adder inputs which respectively cause the carry output to be discharged to ground or the propagation of the carry input signal to the carry output. If neither of these signals is generated, then the carry output remains at the high level to which it is precharged. As it happens, the sum output of the adder can be formed from the exclusive-or of the PROPAGATE signal and the carry input.

The logic necessary for the generation of the KILL and PROPAGATE signals can be used in this case to control the function of the arithmetic unit. If a general function block [25 p151] is used to generate these signals, then they can be made to be produce any logic function of the adder inputs under the influence of external control lines. By this method, the adder can be made to perform addition, subtraction and all the other operations required of it and is transformed from a simple adder into an arithmetic unit.

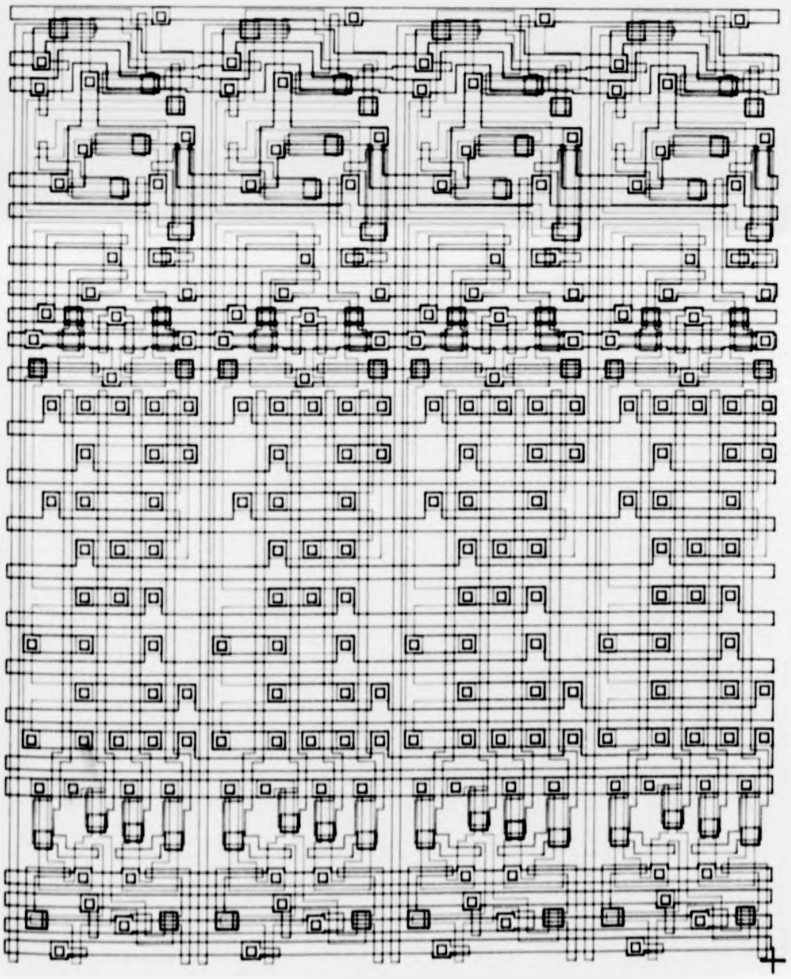


Fig.5.15 Layout of a 4-bit arithmetic unit

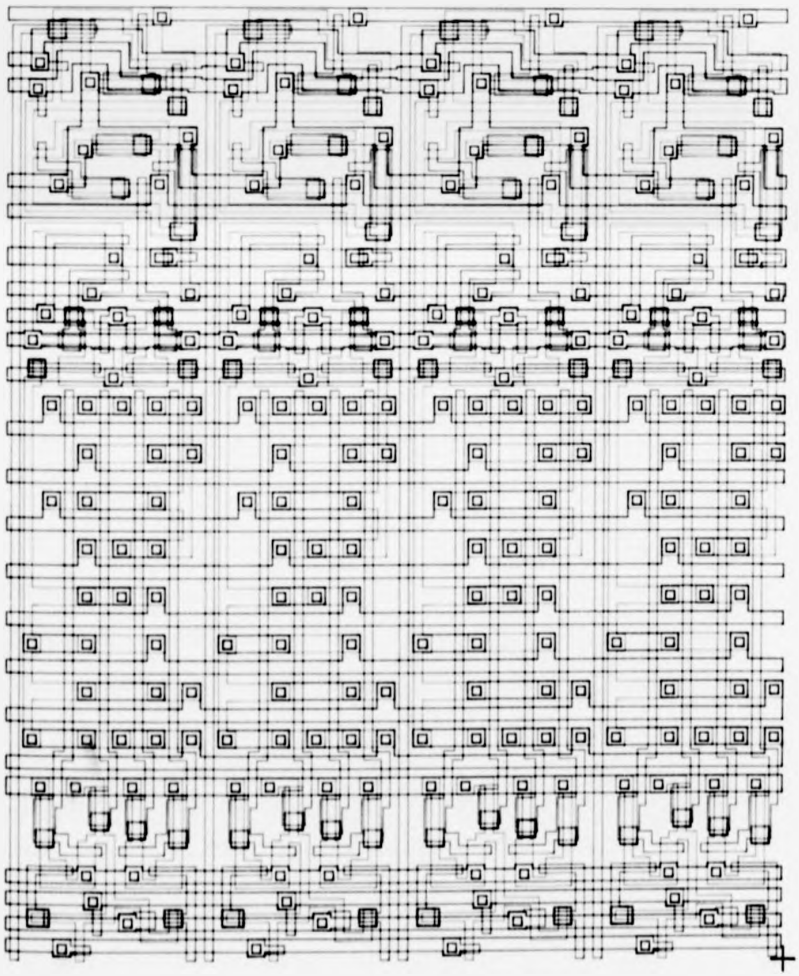


Fig.5.15 Layout of a 4-bit arithmetic unit

The layout of the 4-bit arithmetic unit can be seen in figure 6.15. The key question in the layout of a Manchester carry chain adder is to keep the carry propagation delay as short as possible. The main technique available to achieve this is to reduce the capacitance of the carry line. It is therefore run in metal, keeping the active area and polysilicon connected to it down to a minimum. The height of the adder cell, and hence the whole bit slice, is also kept as small as is compatible with good overall usage of silicon area. The only transistor gate connected to the carry input lines is of minimum size, again to reduce capacitance. This transistor forms the pull-down device of an inverter whose output is used in the generation of the sum signal. The transistors used to effect the propagate and kill functions are 12 micron wide, rather than the minimum 6 micron, to reduce their on-resistance and also reduce the carry propagation delay. They could be made wider still, and this would probably be useful in an adder of longer word-length.

6.2.2 Design of a Barrel Shifter

The barrel shifter is based on a design in Mead and Conway [25 p157]. It is based on pass transistor logic, and one of its main strengths is that no signal passes through more than one transistor, thus keeping delays down to a minimum. The design has been modified slightly to allow for the different functionality required in this case. The layout of the 4-bit barrel shifter is given in figure 6.16. Data from the output register of the arithmetic unit is fed back into the barrel shifter on the long polysilicon tracks at the top of each bit slice. Data is also fed into the barrel shifter perpendicular to the normal data flow. This data will be from the left

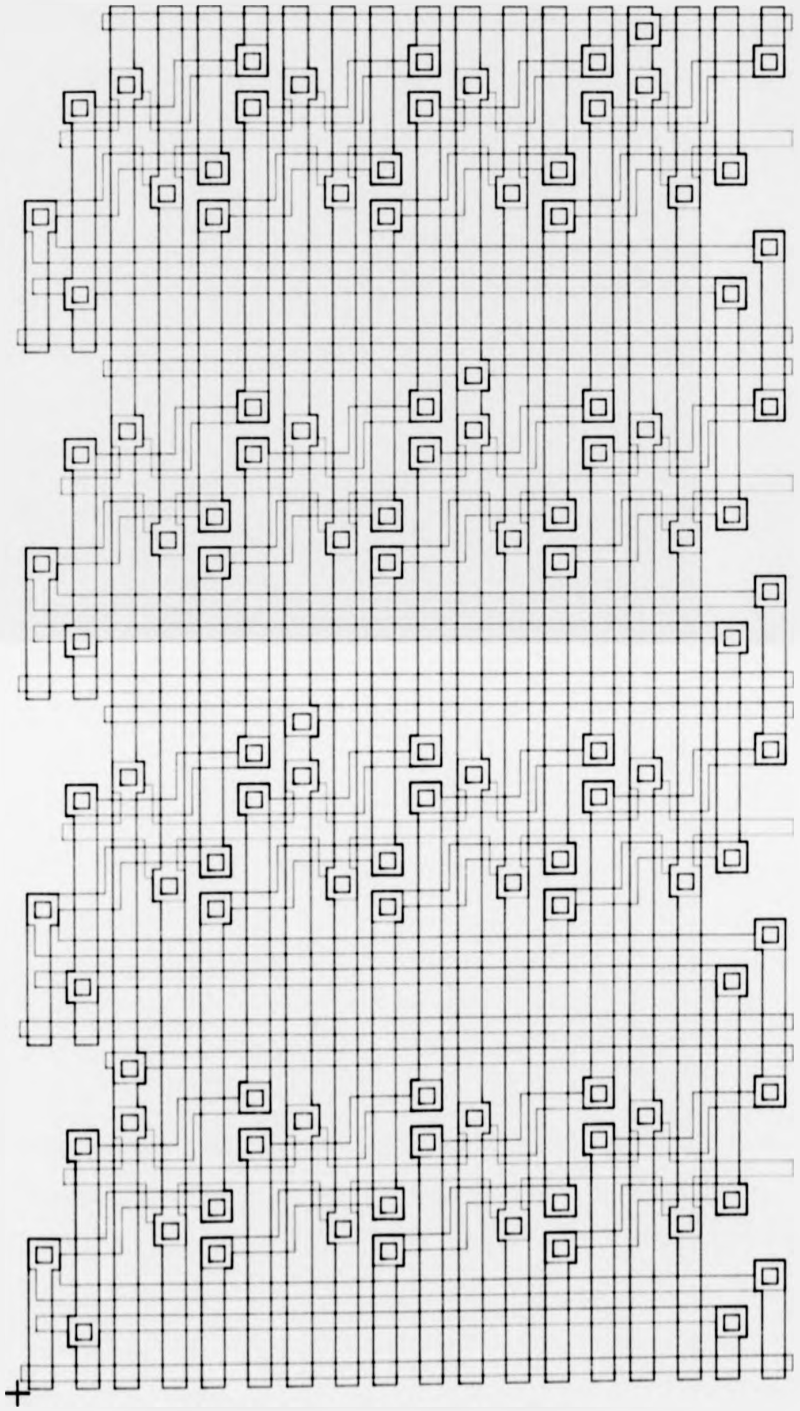


Fig.6.16 Layout of a 4-bit barrel shifter

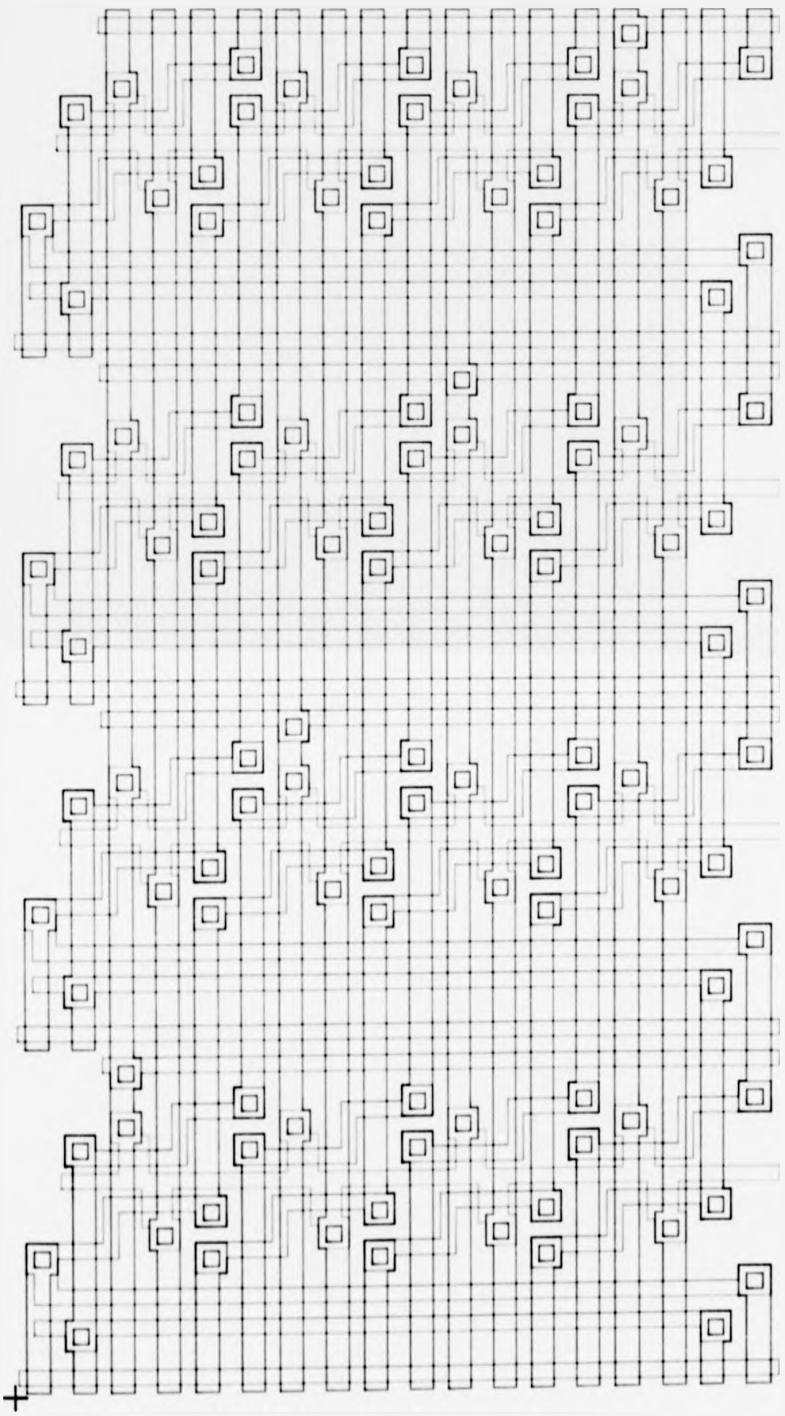


Fig.6.16 Layout of a 4-bit barrel shifter

or right data port of the SPE as appropriate to the direction of the shift operation. Control signals are provided to determine the direction and distance of the shift operation.

6.2.3 Register Designs

There are four registers incorporated in the datapath of the SPE, each with specific functions. They are the accumulator, the RB register, the multiplier register and the output register of the arithmetic unit. They are all based on the two-phase semi-static register described by Mead and Conway [25 p163], with modifications depending on the possible sources and destinations for their data. Their locations can be seen on the layout of the whole datapath which is presented in figure 6.17. The multiplier register is at the extreme left of each bit slice, the RB register being between it and the barrel shifter. The accumulator may be found between the barrel shifter and the arithmetic unit, with the arithmetic unit output register being at the extreme right of the bit slice.

6.2.4 Circuit Level Simulation of the Datapath

Circuit level simulations of the individual cells which make up the datapath were performed using the SPICE2G.5 simulator. These verified the functionality of the cells. More complex simulations would be required to estimate the performance of the datapath. The longest delay is likely to be associated with the arithmetic unit, and in particular the carry chain. Firstly, the carry chain itself was simulated, but without realistic load capacitances or input waveforms. As soon as any more circuitry was added to produce a model of the arithmetic unit similar to that used for the RAM, the

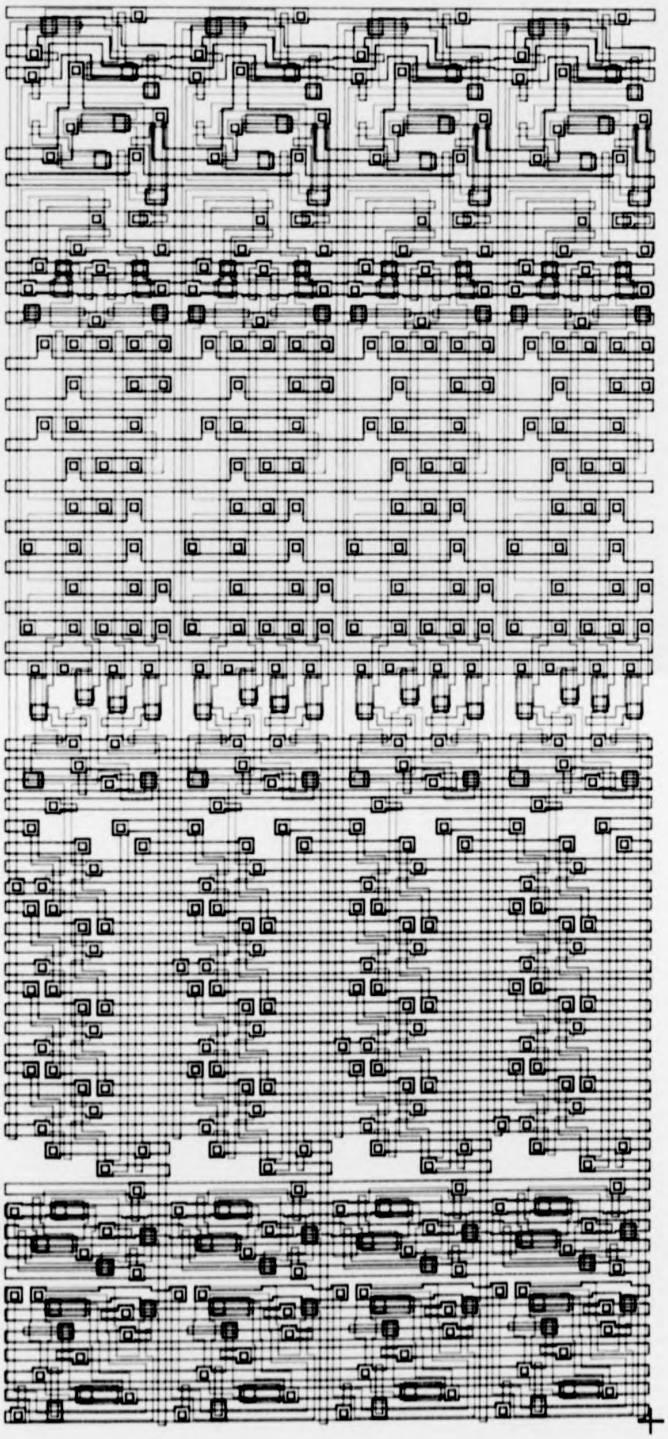


Fig.6.17 Layout of the entire datapath

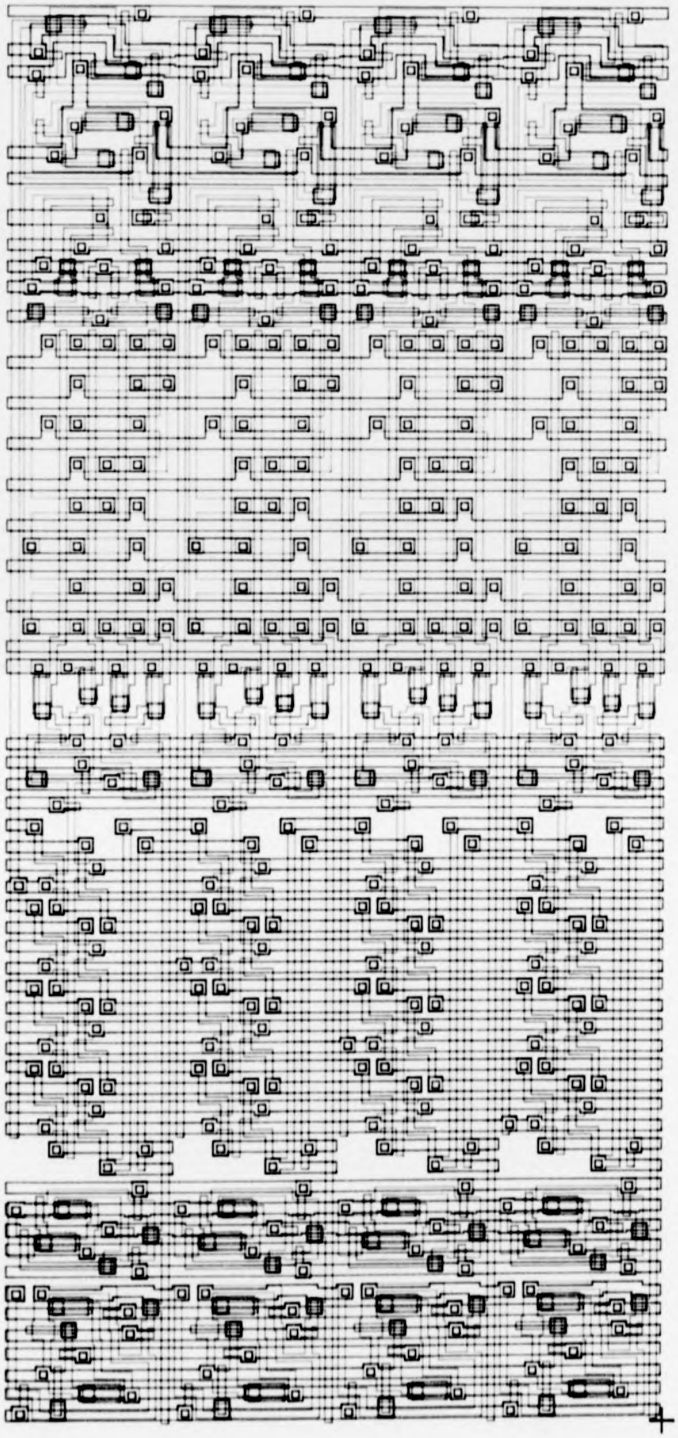


Fig.6.17 Layout of the entire datapath

simulations failed to converge. After spending several days on this problem, it was abandoned following the experience with the RAM chip simulations. In fact, it is likely that the RAM access time will be the limiting factor in terms of speed rather than delays in the datapath, especially as the length of the carry chain is only four bits.

These experiences with SPICE demonstrate very clearly the need for an improved circuit simulation tool for MOS IC design. The arithmetic unit is simple enough that it ought to have been possible to simulate it in its entirety and obtain accurate results, without resorting to try to devise a simplified model, whose simulations failed to converge in any case.

6.2.5 Layout of a Test Chip

With the designs completed as above, the datapath could be fabricated immediately. The only problem was to decide on which signals to connect to the outside world to keep the total number of pins below 40, as it would be possible to make many more connections than this. A number of control lines were identified which should always be enabled either during ø1 or ø2 in normal operation, and so these were connected together to the appropriate clock phase input. This having been done, it was possible to connect all the remaining control lines to input pins along with the carry input signal. The data inputs to the barrel shifter which would eventually be connected to either the left or right data port of an SPE were also connected to input pins as a means of reading data into the system. The four bus lines, along with the carry output line were all connected to output pins. This provided a chip with many inputs but not many outputs, which is something of a disadvantage in terms of testing. It would

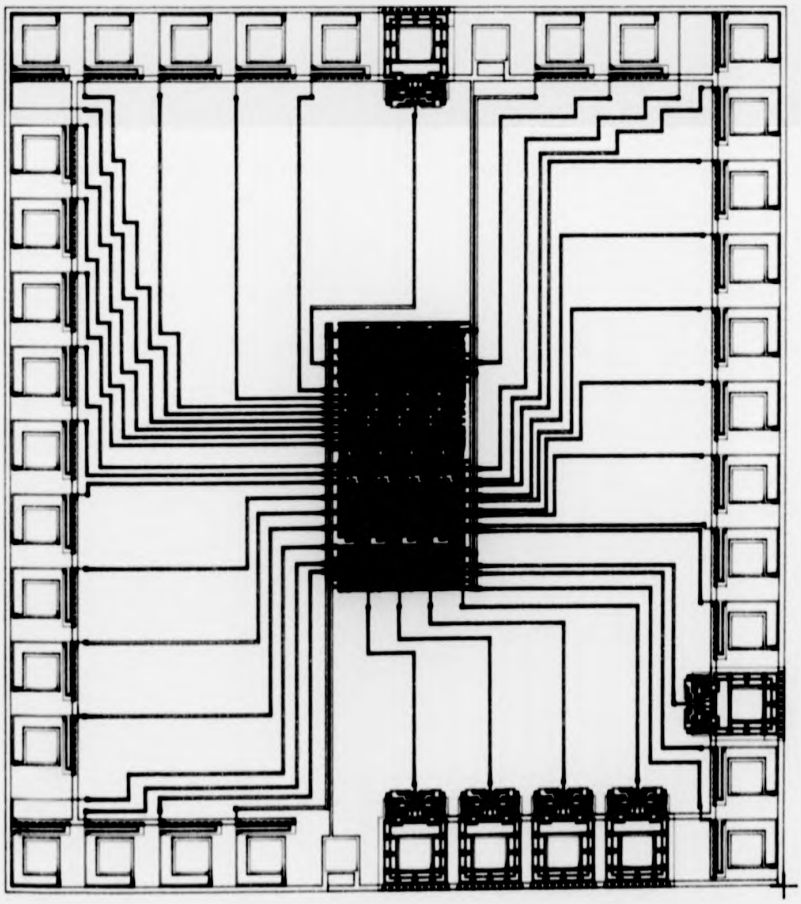


Fig.6.18 Layout of the datapath test chip

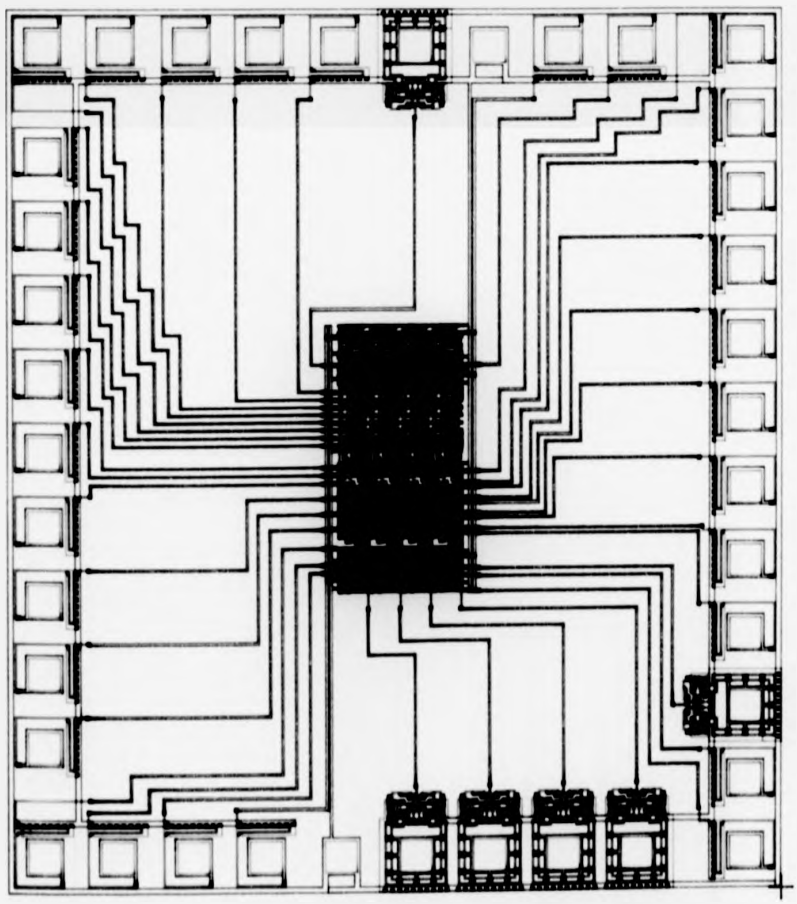


Fig.6.18 Layout of the datapath test chip

have been useful also to connect the outputs from the accumulator output register to output pins, but these pins were simply not available. However, the configuration adopted would be able to verify the operation of the datapath if everything worked properly; it would be a little more difficult to locate the source of a problem in the case of failure.

6.2.6 Testing the Fabricated Datapath Chip

The testing of the datapath chip was somewhat more successful than that of the RAM chip. Altogether six packaged chips were available, but unfortunately none of them performed correctly. The most promising chip showed some problem associated with the top (most significant) bit; all the others showed more fundamental problems.

The first test simply consisted of writing a value into the accumulator and then reading it out again. The data was input through the barrel shifter and read out on the bus. These are the only ways of getting data in and out of the test chip as it had been connected up. Various patterns of data were tried, and all tests were successful except that the top bit was always read as a zero. The remaining test chips were rejected at this stage as only one or two bit positions could be modified on each of them, the others being permanently stuck at one or zero.

The next test was on the RB register. In this case, data was written into the accumulator, transferred to the RB register and then read out again. As the transfer to the RB register was necessarily performed on the bus, a different value was written into the accumulator and read out again between transferring the data into the RB

register and reading it out again, in order to verify that reading the RB register actually altered the value on the bus. Once again, this test was successful except for the top bit sticking at zero.

The testing of the multiplier register was not so successful. There is no way of reading the data out of this register in parallel, however the register is designed as a parallel-in, serial-out register, so data written in parallel should be possible to read out serially. As the serial data output of the multiplier had been connected to an output pad of the test chip, this was indeed tried. However, the output proved to be permanently stuck at zero.

The arithmetic unit can be programmed to perform various different functions in accordance with the values on the control lines to the propagate and kill logic function blocks. One basic procedure was adopted with only a few modifications to test out these various functions. Firstly, the accumulator and the RB register were loaded with suitable data in the manner described above. Then the arithmetic unit was operated for one cycle, and the output loaded back into the accumulator. This value was then read out onto the bus. The carry input and output lines were also available for control and observation respectively. Within the constraint of the non-functional top bit, all the operations required by the SPE: addition, subtraction, negation of the accumulator, etc., were all verified. The effect of the value of the carry input was clearly seen. The carry output bit could only be observed in the subtraction operation; the top bit being zero meant that the carry output was always zero during addition.

The barrel shifter had already been largely checked out by the previous tests. As already stated, the only way to get data into the test chip was through the barrel shifter, and data routed from the output of the arithmetic unit to the accumulator also goes through the barrel shifter. These basically verify shifts of zero and four bits distance, four bit shifts left and right being identical as far as the datapath is concerned. The test for the arithmetic unit was easily modified to test other shift distances when the output from the arithmetic unit was fed back into the accumulator. The effect of the sticking top bit was rather a nuisance in this case, but as far as could be verified the outputs were as expected.

One of the main reasons for producing test chips was to verify the likely speed of operation. This partly working chip gave that possibility. In a finished SPE, many of the control signals would be ANDed or ORed with one of the clock phases, whereas in this case they were directly connected to the tester. This meant that four tester clock cycles had to be used for each SPE clock cycle: two for the phases ø1 and ø2 and two for the inter-clock gaps to ensure non-overlapping. In a real system, the gaps would naturally be much shorter than the phases themselves, which might not in fact be the same length. In all the tests above, the clock frequency of the tester was increased until the output began to change. In each case, this happened at about 5MHz, meaning that ø1 and ø2 were each of 200ns duration. The gaps could probably be reduced to about 50ns each, giving an overall clock period of 500ns. It therefore appears that the SPE could be expected to operate at a clock frequency of about 2MHz. This is not wholly unreasonable for a 6-micron NMOS technology, and could be expected to be about ten times faster for a

truly VLSI CMOS process.

6.3 CONTROL LOGIC DESIGN

The third major part of an SPE is the control logic. This could not, unfortunately, be implemented in time for the fabrication run used for the memory and datapath chips. It was therefore decided to perform a design study of the control logic sufficient to give approximate estimates of the area occupied and the timing. In view of the problems previously encountered with SPICE previously, and the inordinate amount of time expended trying to obtain results, it was also decided not to spend a great deal of time on SPICE simulations, but to concentrate on realistic estimates of delays in the circuit.

The major part of the control logic is concerned with providing the control signals to the datapath. Other parts are required to perform the decoding associated with the COND instruction provided for the implementation of multiplication (qv section 4.6.3) and for the communication of data to and from the right and left data ports.

6.3.1 DATAPATH CONTROL

The control signals for the datapath are formed from combinatorial logic functions of the instruction registers IR and IO and the clock signals. There are two different classes of control signal: those which are active during ø1 and are derived from the IR register, and those which are active during ø2 and are derived from the IO register. The former group consists of the Arithmetic Unit (AU) control signals and those which control the writing of the multiplier (MIER) and RB registers. The latter group consists of the barrel

shifter control signals and those which control the refreshing and shifting of the MIER register.

The obvious choice for implementing the logic to derive these control signals is the Programmed Logic Array (PLA) [25 p79ff], however other possibilities were explored before settling on the final arrangement. There are clearly no common product terms between those signals which depend on IR and those which depend on IO, and hence separate logic for these, whether in the form of PLAs or otherwise, is sensible. As far as the AU control signals are concerned, there are many common product terms among the control signals, and there is no real sensible alternative to a PLA. It was also found that the other two control signals dependent upon IR also fitted well into this PLA. The layout of the PLA is given in figure 6.19. It occupies an area of 730 by 550 micron, or 0.40 mm^2.

The IO dependent signals are rather different. There are no product terms common between signals, and a number of signals are derived from only one product term. Indeed, there are only two signals derived from multiple product terms, namely those which cause no shift or a 4-bit shift through the barrel shifter. These two signals are known as S0 and SRL4 respectively. For those signals derived from only one product term each, a distributed NOR gate, essentially identical to the AND plane of a typical NOR-NOR PLA, is adequate and probably the best solution. For the S0 and SRL4 signals, the possibility of using individual complex AND-OR-INVERT or OR-AND-INVERT gates was considered. However, preliminary investigations indicated that this was not a practical approach. The number of series transistors involved in such gates in either form were such that, for reasonable sized pull-down transistors, the pull-up transistor would

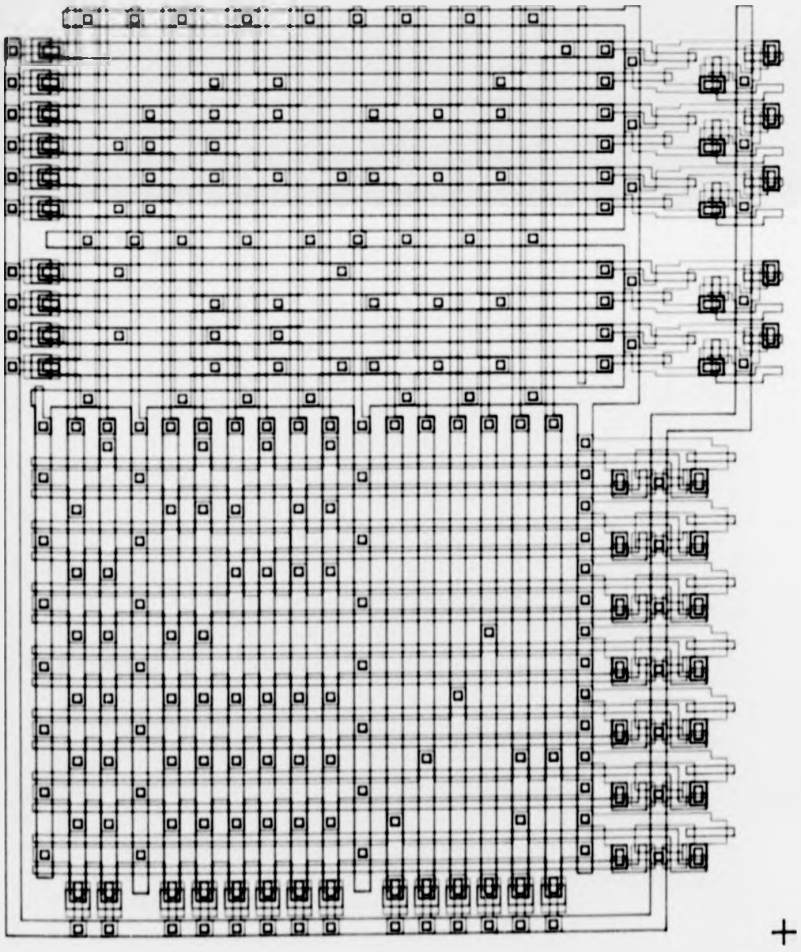


Fig.6.19 PLA layout for IR dependent control signals

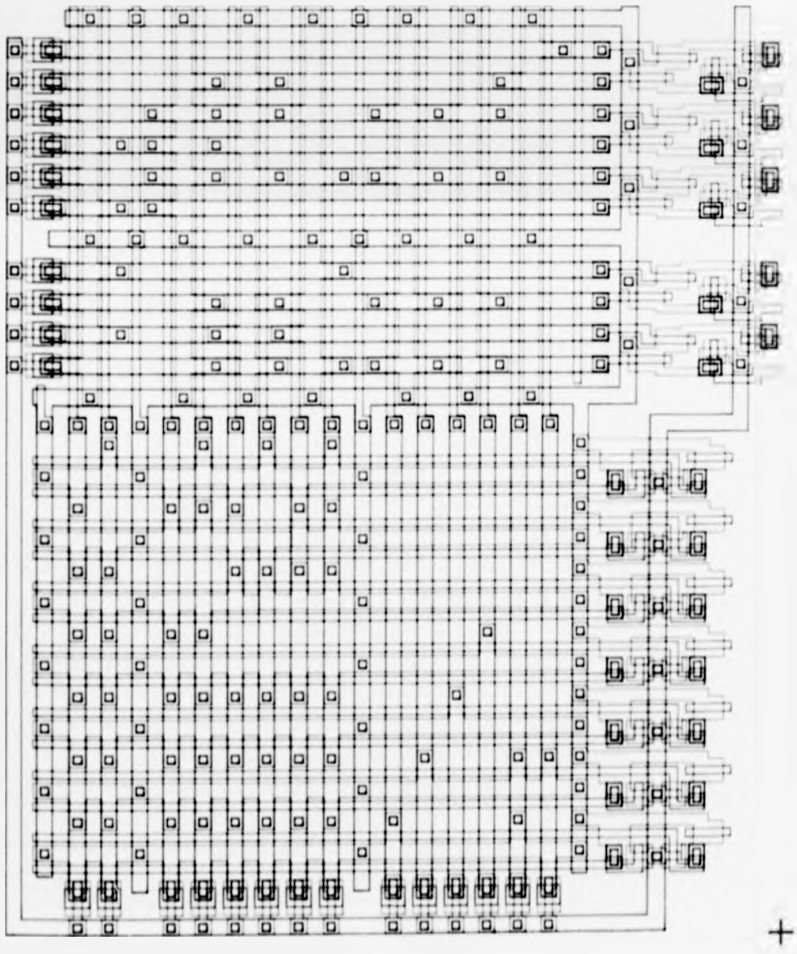


Fig.6.19 PLA layout for IR dependent control signals

have to be very long. This resulted in a rise-time for the gates in excess of 100ns being obtained from SPICE simulations, when loaded with a minimum size transistor. It was therefore decided that a PLA would once again be the best solution for these two signals. This PLA has fewer product terms and outputs than the previously described one, and is therefore smaller and faster. It occupies an area of 400 by 430 micron, or 0.17 mm^2.

In view of the difficulties involved in SPICE simulations in the previous designs, only a very simple simulation of the PLAs has been performed. As the first PLA described is larger and therefore slower, it alone has been simulated. The capacitances of the long tracks have been estimated. The AND and OR planes have each been represented as an inverter with appropriately sized transistors. The input and output buffers have also been simulated. Each section was initially simulated separately, and then they were all connected together. Somewhat to the surprise of the author, this simulation worked first time. The results are presented graphically in figure 6.20. These results show the overall delay through the PLA to be of the order of 30ns. This is unlikely to degrade the performance of the overall system. The main timing requirement is that the output should be valid well before the end of the clock phase in which it is required.

The control logic described so far only performs the derivation of the logic signals as functions of the instruction registers. Those which control the loading of registers, as opposed to the function of the barrel shifter and AU, must also be ANDed with the appropriate clock signal. All the signals will need appropriate buffering to drive the datapath control lines. As the datapath is

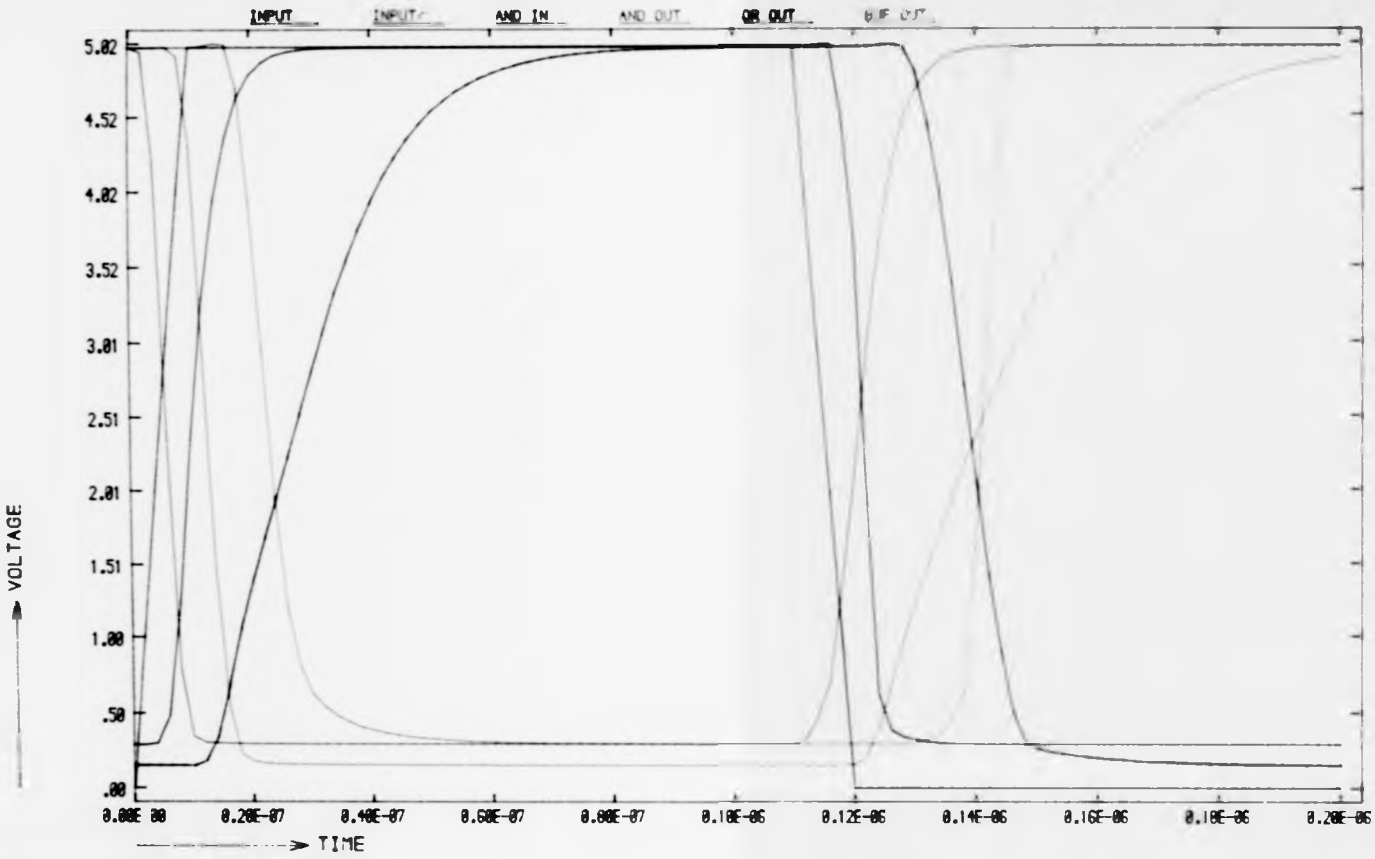


Fig 6.20 SPICE simulation of control PLA

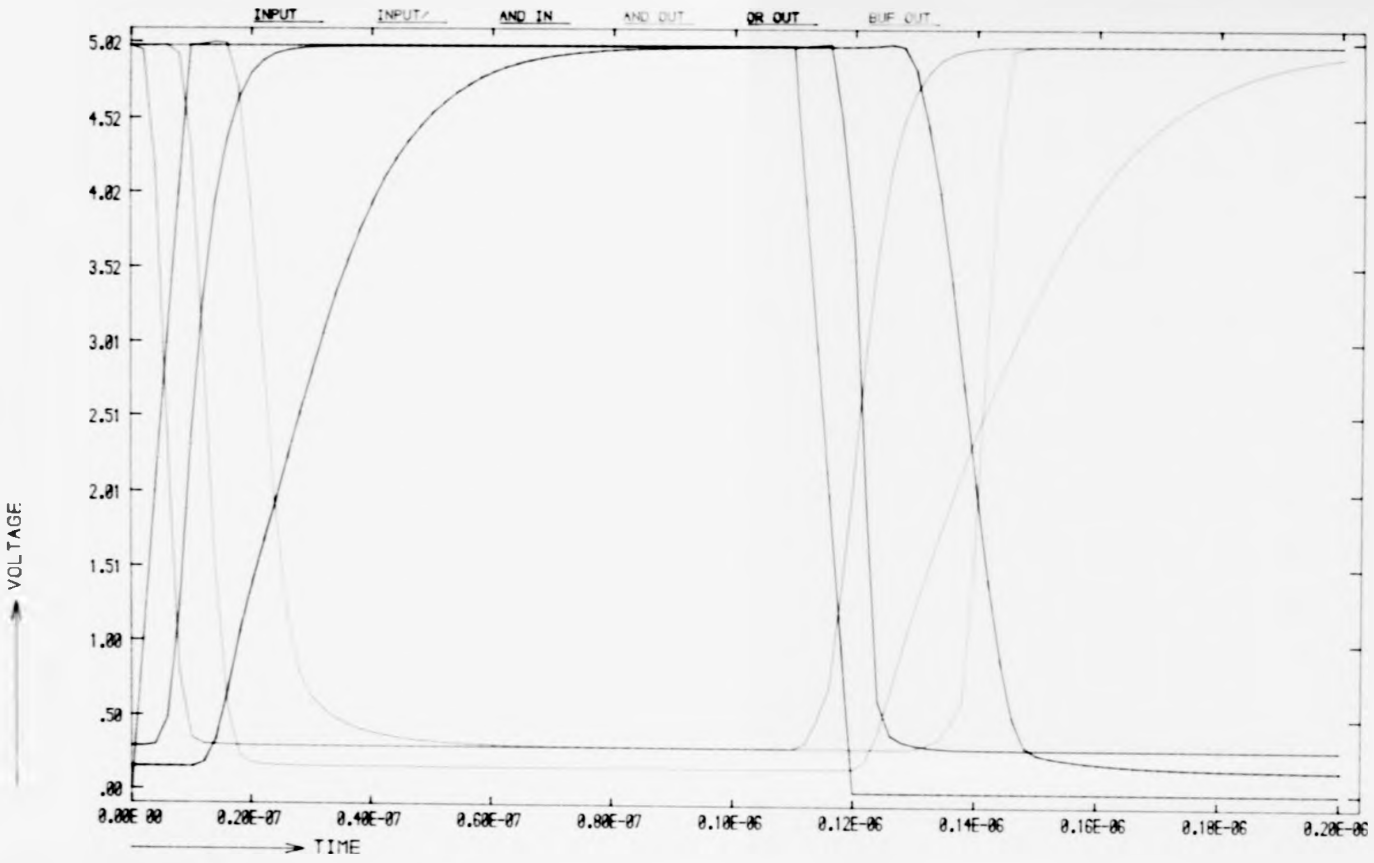


Fig 6.20 SPICE simulation of control PLA

relatively small, these will not need to be very sophisticated.

6.3.2 GENERAL CONTROL LOGIC AND WIRING

The rest of the SPE would be made up with the other small pieces of control logic referred to above, and with wiring. The logic involved is relatively simple, and its implementation would to a large extent depend on wiring considerations. It was not considered necessary to proceed any further with detailed design, as it would not significantly contribute to estimates of performance or area. In particular, the problem of the provision of program memory, considered in the next chapter, shows that items of detail in the control logic are not really relevant to the overall assessment of efficiency in use of silicon.

CHAPTER SEVEN

CONCLUSIONS

7.1 Evaluation of Project

The aim of the project was to design a programmable signal processor with variable word length, capable of performing difference equation calculations, and which could be replicated many times on a single VLSI integrated circuit. This has been achieved. The architecture has been verified using the HILO functional simulator. Salient parts of the architecture have been subject to custom integrated circuit design, some of which has been fabricated. Some circuit simulation has also been performed. These permit estimates to be made of system performance in terms of speed and silicon area. The results must now be evaluated and compared with other approaches to assess the viability of the approach adopted, and recommendations made for further research.

7.1.1 SPE Achievements

The aim of developing a programmable signal processor with variable word length has been achieved with the nibble-serial architecture adopted. A processor is formed from the appropriate number of nibble-wide Signal Processing Elements (SPEs). Data is input from the left into the most significant SPE and instructions from the right into the least significant SPE. This system poses severe constraints on the processors. Each instruction must be executed in precisely one clock cycle, and no branching instructions are permissible. This would clearly be unacceptable in a general purpose

microprocessor, but the determinism of difference equation calculations makes it acceptable. Indeed the Intel 2920, which was used as a reference point for much of the design, works under similar constraints. The register transfer level simulations described in chapter five were used to verify the operation of the architecture as described, and a real filter was implemented.

The custom IC layout undertaken, and described in chapter six, allows an estimate to be made of the silicon area occupied by an SPE. The data memory occupied 3.25 mm^2, the datapath 0.62 mm^2, and the control PLAs together 0.57 mm^2. This gives a total of 4.44 mm^2. Allowing reasonable area for the wiring and remaining control logic, it is unlikely that an SPE could occupy less that about 6 mm^2. Program memory would need to be separate. Before estimating how many SPEs could be integrated onto a single chip using a VLSI process, it is necessary to consider in more detail the provision of program memory.

7.1.2 VLSI implementation of SPE-based processors

7.1.2.1 The provision of program memory

So far, the provision of program memory for SPE based processors has not been considered. If many SPEs are to be fabricated on a single VLSI chip, then clearly both the SPEs and their program memory must be integrated. The SPEs themselves do not have any provision for sequencing the program memory, and therefore simple sequencers must also be provided. As no instructions can change the program flow, this would simply consist of a program counter, which would be reset after a preset count had been reached.

The program memory could be provided in three forms: as read only memory (ROM), erasable programmable ROM (EPROM) or as read-write memory (RAM). In this last case, the program memory would need to be loaded from external ROM or EPROM after power-up. Despite this drawback, this approach has many attractions. ROM would certainly provide the smallest solution: the area of a ROM cell designed for the Edinburgh 6 micron NMOS process measures 18 by 24 micron, or 432 square micron, whereas the RAM cell described in section 6.1 measured 78 by 106.5 micron, or 8307 square micron. A ROM implementation, however, would wipe out many of the advantage of a programmable system. An EPROM cell would be smaller than a RAM cell although larger than pure ROM, and would seem to be ideal except for the relatively long access times of EPROM devices. This would be likely to seriously reduce the possible clock frequency. RAM therefore appears to be the main contender. It would be possible to produce two versions, one with RAM for program memory and the other with ROM. The RAM version could then be used for development purposes and the ROM version for production. The use of off-chip program memory even for development purposes is clearly not possible when many processors are implemented on one chip.

The most serious problem in the provision of program memory is that it is not known which SPEs will require program memory. Word-length is variable by using a variable number of SPEs in a single processor, and program memory is only required at the least significant SPE. The configuration of the SPE-based processors will need to be performed using memory bits dedicated to this purpose, which will need programming in a similar manner to the main program memory. As four to six SPEs would be used in each processor in most cases,

representing 16 to 24 bit wordlengths, this could result in throwing away between 75% and 83% of the program memory. This is clearly unacceptable, and some way must be made to share program memory between SPEs in a flexible manner.

Allied to this problem is the question of how much program memory is required. One way to assess this is to consider the ratio between the clock frequency and the sampling frequency. Hopefully a clock frequency of about 20MHz could be achieved with a VLSI process. A sampling frequency of about 10kHz is probably the lowest which could be contemplated, this corresponding to telephone quality speech. This provides 2000 clock cycles per sample. As there are no branching or looping instructions in the SPE, and each instruction is executed in precisely one cycle, this would suggest the provision of 2000 program memory locations per processor. Another, probably more realistic approach is found by considering the filter program of Figure 5.12. This utilises 63 instructions, and occupies 7 memory locations. It could be supposed that a program of about 500 instructions would use up almost all the 64 locations of data memory in the SPE. In any case, it is undesirable to provide excessive program memory, as this will reduce the efficiency of use of silicon. If it is further assumed that each bank of program memory will be shared between four SPEs at least, this suggests providing 128 locations per SPE. Once again, using the RAM cell previously designed, the cell array part of such a memory would occupy 8.5 mm^2. This is significantly more than the data memory, datapath and control logic described in chapter 6 put together. With the required memory peripheral circuitry, the area occupied would be more like 10 mm^2. If ROM were to be used, the area would be much smaller. 128 bytes of

ROM would occupy only 0.44 mm^2 for the central array, or probably about 1 mm^2 to include peripheral circuitry. It may well be that the RAM cell layout of figure 6.1 could be improved. This is particularly so if a process tailored to the implementation of memory were used. This would not be inappropriate, as memory forms a large proportion of the chip in any case.

It has been seen to be necessary to share program memory between SPEs to avoid excessive waste. The silicon area calculations above only serve to emphasise this point. One possibility would be to provide a block of program memory to any one of four SPEs. This would mean wasting 33% of program memory for 24-bit wordlengths and 50% for 32-bit wordlengths. Wordlengths of less that 16 bits would not be available, although this is not likely to be a limitation in practice. The only other alternative would be to provide some reconfigurable way of organising program memory into blocks. The result of this would be that processors with longer wordlengths would have more program memory available. There is no reason that they should require more, as, using the guidelines above they have the same number of data memory locations. Although it would be possible to devise some such reconfiguration scheme, it is likely to have a not insignificant overhead in silicon area and possibly performance. The best compromise seems to be to provide a block of program memory, together with a simple sequencer, for every four SPEs. It is very unfortunate, but it seems that there is no way that the flexibility afforded by the SPE-based approach can be extended to the configuration of program memory.

There is now sufficient data to estimate the number of SPEs which could be integrated on one VLSI chip. The area of a single SPE has

been estimated at 6 mm^2 and the area of program memory at 10 mm^2 or a total of 16 mm^2. This probably ought to be rounded up to 20 mm^2 for further wiring, etc. However, this area relates to a 6 micron process. A VLSI process with feature sizes as low as, say, 0.5 micron is not hard to envisage. In this case, the total area per SPE would be reduced to 0.2 mm^2, assuming all dimensions scale proportionately. A VLSI chip having 100 mm^2 available for the main circuitry (peripheral circuitry being extra) seems reasonable. This would indicate that 500 SPEs could be implemented on a single chip. This chip would possess formidable processing power. If ROM were used for program memory in place of RAM, even more could be integrated. In the case of Wafer Scale Integration (WSI), if such ever becomes practical, the processing power available could be phenomenal, although it is doubtful that applications could be found for such power, given that the processor would still essentially only be capable of implementing difference equations.

The programming of a VLSI chip containing about 500 SPEs would not pose too great problems in the configuration envisaged above. With 512 bytes of program memory for every four SPEs, the total program memory size would be about 64Kbytes. It would be possible to make every address location directly accessible by using a 16-bit address bus for programming purposes. This would violate the principle of local communication, but only for programming purposes. It would not necessarily compromise performance during normal operation. An alternative programming method would be to make extensive use of shift registers to move data round the chip. A single shift register could be used to select which memory blocks to program, permitting the programming of a single block, or of many blocks in parallel when

they are to execute the same program. Shift registers could also be used to move the program data around the chip. In conjunction with counters to cycle through program memory addresses in each block, the memory could be efficiently programmed.

7.1.2.2 Testability of SPE-based processors

Testability is a vitally important concern in the design of VLSI circuits, with built-in self test rapidly becoming the only practical way to achieve good testability coupled with acceptable time on the tester. In the case of SPE processor chips, the situation is eased because they consist of a large number of identical and reasonably simple elements. This means that the question can be broken down into two main problems, the testability of the SPEs themselves, and access to their inputs and outputs. This access, and the generation of test patterns and the analysis of outputs could be achieved using Built-In Logic Block Observer (BILBO) elements [37] or similar. The testability of the SPEs themselves should be inherently quite good, as access is available to most components under program control. If adequate test cover could not be achieved in this manner, BILBO techniques could equally well be applied within each SPE. As far as testing the program memory is concerned, this essentially means ensuring it can be read as well as written.

7.1.3 Comparison with Intel 2920

The Intel 2920 has been used as a guide for the design of the SPE. It is therefore interesting to compare their performance. For this reason, the filter of Appendix A has been coded up for the Intel 2920. The program is given in Appendix B. This should be compared

with the SPE program for the same filter, given in figure 5.12. No attempt has been made to include the analogue portions of the 2920, to permit better comparison with the SPE.

The first factor to note is that whereas the SPE program has 65 steps, the 2920 program only has 23, although 7 of the SPE instructions are concerned with the input and output of data which is not covered by the 2920 program. The main reason for the difference is that shifting and adding are two separate operations in the SPE, but one in the 2920. This has to be so to support the nibble-serial operation of SPE-based processors. Another reason for the extra instructions is that all the data memory locations in the 2920 are directly available for arithmetic operations; in the SPE implied registers are used and data must be moved around more. This was done to reduce the width of the instruction word, which is 24 bits in the 2920 but only 8 bits in the SPE. The limitation of shift length to 4 bits per shift in the SPE only caused one minor inconvenience in the SPE program, resulting in two extra instructions being required.

The Intel 2920 was implemented in an NMOS process of the late 1970s, not totally dissimilar to that used to fabricate the SPE test chips. Its maximum clock frequency of 2.5MHz is similar to that predicted for the SPE. The power of the 2920 would be roughly comparable to the power of an SPE-based processor consisting of 6 SPEs and a block of program memory, although due to the requirement for more program steps, the SPE-based solution would be slower for the same clock rate. The 192 words of program memory in the 2920 would provide approximately the same functionality as the 512 words proposed for the SPE, taking into consideration the comparison of programs above. In fact, 512 by 8 (4096 bits) for the SPE is slightly

less memory overall than 192 by 24 (4608 bits) for the 2920. The 2920 employed EPROM for the program memory, which would be much smaller that the RAM proposed for SPE based systems. A chip fabricated in 6 micron technology, incorporating 6 SPEs and one block of program memory implemented in RAM would occupy about 76 mm^2: a large chip, but not impossible. This would be reduced to about 40 mm^2 if ROM were used for the program memory, which provides a more fair comparison with the 2920. The chip area of the 2920 is not known, but it is known to be produced on a single NMOS chip.

7.1.4 Suitability for Signal Processing Algorithms

The SPE has been designed specifically to be efficient in the implementation of difference equation calculations. These are used to implement both recursive and transversal digital filters. Other important signal processing algorithms include adaptive filtering, the Fast Fourier Transform (FFT), matrix arithmetic, and logarithms and exponentiation. These last two are relevant to homomorphic processing (q.v. section 1.1.3). There is no reason in principle why some of these algorithms should not be implemented on SPE based processors, however there would be limitations which would probably prevent their effective use. For example, the FFT butterfly operation could easily be programmed into the SPE. However, the small data and program memories would limit the calculation to FFTs with very few points indeed. Similar considerations would apply to matrix operations. Transcendental functions such as logarithms and exponentiation could once again be programmed into an SPE, permitting the implementation of certain homomorphic processing algorithms. However, such things as calculating the cepstrum require FFT calcula-

tions, and these have been seen to be impractical. It must be pointed out, however, that these same restrictions all apply to the Intel 2920 in just the same way, and yet that has found application, particularly in speech bandwidth telephone circuits. It would have to be accepted that an SPE based processor would probably only find application in similar areas. Alternative architectures would have to be explored for more general application.

It might at first seem that SPE-based processors would be likely candidates for implementing adaptive filters, because these usually use FIR filters with time-varying coefficients to perform the actual filtering. The calculation of the filter coefficients is handled separately. SPEs would be perfectly capable of implementing the filters, and the ability to execute multiplication by a variable, rather than coding the multiplicand into the program, would enable adaptive filtering. The calculation of the filter coefficients, however, is not something which could easily be performed by the SPEs, and would have to be handled by a separate processor. The main drawback would be communicating the filter coefficients to the individual SPE-based processors. The only path available for this data is that used for the signal data itself. All the filter coefficients for each processor in the pipeline would need to be sent to the SPEs during each sample time, unless the program memory sequencer were made much more complex. This would be an unacceptable overhead, seriously reducing performance. It is therefore unlikely that adaptive filtering could sensibly be performed on SPE-based processors without significant modification to the architecture.

The only way SPE based processors can be used for parallel processing is as a pipeline. Once again, this is fine for implementing

digital filters by difference equations. A vector processor would be more appropriate for implementing matrix arithmetic, it is not so clear what form of parallel processing is suited to calculating FFTs. Any flexibility in the interconnection scheme for SPE based processors would greatly increase the complexity. This goes against the fundamental principle of keeping the SPE as simple as possible, so that as many as possible can be fabricated on a single chip.

7.1.5 Incorporation of delay function in memory

The data memory of the SPE has been designed so that half of it is shiftable, permitting the z^{-1} delay to be performed for all data in only one cycle (qv section 4.5). This, of course, incurred some cost. The area of the central cell array of a 64 by 4 bit RAM was found to be 2.13 mm^2 (q.v. section 6.1). The area of the memory incorporating the shifting function into half of it was found to be 3.25 mm^2. This is a significant increase in area, although the function is clearly very useful. It could, however, probably be better implemented by having an index register which would be added to the address in the instruction. This index register could be incremented each time a z^{-1} delay is required. This approach was considered during the design of the SPE, but was rejected as it involved an extra 6-bit addition in each memory access cycle. It was thought that this might reduce the performance of the SPE. In retrospect, it must be admitted that this was rather naive, and the index register would be unlikely to significantly affect the performance of the SPE. The area occupied by the extra register and adder would certainly be smaller than the overhead of providing the shiftable memory.

7.1.6 Silicon Area Efficiency

The main reason for targeting a processor at a particular application area such as signal processing, is to improve the efficiency in use of silicon over general purpose processors. In this case, the aim of making the wordlength configurable was also to improve silicon area efficiency. There can be little doubt that an SPE based processor represents more efficient use of silicon than a general purpose microprocessor for signal processing applications, but it must be doubted whether the configurable wordlength really justifies its cost. In particular, the difficulty of providing program memory in a configurable manner negates many of the advantages of the wordlength configurability. This is particularly so when the dominance of program memory in terms of silicon area is recognised. Other approaches to building signal processors, such as the Texas Instruments TMS320, and the Inmos Transputer, also offer big improvements over standard microprocessors, and achieve a far greater generality of application than SPEs or indeed the Intel 2920.

It has transpired that the datapath of the SPE is small in relation to the data and program memories. The control logic and the program memory can be regarded as a single entity for these purposes, as they are both concerned with the implementation of program control. The Arithmetic Unit itself is even smaller than the datapath, which also includes the registers and barrel shifter, although it must be admitted that the barrel shifter can be regarded as performing arithmetic operations too. In any case, this relatively small area of silicon which is actually doing the work is unfortunate, as the main aim is to make as much arithmetic processing power as possible available. It must be admitted in the SPE's favour that the

situation would be several times worse in a general purpose microprocessor. The only way around this seems to be to integrate more dedicated functions in hardware, but still under program control. A parallel multiplier would be an obvious suggestion. This would both increase the silicon area used for real arithmetic processing, and reduce the area used for program memory, which would no longer be full of as many shift and add instructions. This approach could be extended to complex multiplication, and such things as the FFT butterfly operation, although this begins once again to make the processor very specialised.

This kind of approach would mean abandoning the configurability of wordlength, although it has already been seen that the advantages of this are dubious. In designing the SPE it was recognised that the nibble serial approach was essentially incompatible with parallel multiplication. It might be possible to implement 4 by 4 bit multiplication in each SPE, but this would certainly require a major redesign. The present scheme essentially permits 1 by 4 bit multiplication. A similar scheme could be adopted, whereby the multiplier would be circulated to each SPE in any processor, 4 bits at a time. Every 4 bits of the multiplicand, which would remain fixed in the appropriate SPE, would be multiplied by every 4 bits of the multiplier. The 8-bit sub-products resulting from each multiplication would need to be added together to produce the final product. This would be achieved by passing the lower 4 bits of each sub-product to the next SPE on the right, and adding both nibbles into the product. The product itself would be shifted right by four bits after each 4 by 4 bit multiplication. This corresponds to the single bit shift employed at present. As has been stated, this would involve a major

redesign, but if the SPE were thought to have enough potential, could well be worthwhile.

Another way in which the efficiency of SPE based solutions could be improved would be by increasing the number of bits in each SPE. Without fundamentally altering the architecture, 8 bit SPEs could be built in place of 4 bit. The reduced wordlength flexibility might not prove too much of a problem, but increasing the number of bits much beyond 8 would probably take away most of the advantage of providing configurable wordlength. This seems attractive at first, as it reduces the overhead of control logic. However, it has been seen that the program memory takes up much more area than the control logic, and therefore the gain would be much less than would appear at first. Essentially, each SPE would almost double in size, but the program memory would only be shared by two SPEs in place of four. The program memory overhead would be the same in each case. The only gain would therefore be from the fact that an 8 bit SPE would be less than twice the size of a 4 bit SPE. This, in fact, would depend on other decisions, too. If 8 bit shifts were permitted, an 8 bit barrel shifter is roughly four times the size of a 4 bit shifter. This ratio would also apply if a 4 by 4 bit multiplier as proposed above were replaced by an 8 by 8 bit multiplier. These last two possibilities could improve the performance for SPE based processors at the expense of extra area.

7.2 Recommendations for Further Research

It has to be admitted that taking all these things into consideration, SPE based systems do not seem to have a real future as a means of providing programmable digital signal processors. Their

application area is very limited, and the compromises necessary to achieve configurable wordlength have reduced the efficiency in use of silicon to too great an extent. However, the principle of designing special purpose processors as a means of obtaining efficient use of silicon without sacrificing the flexibility and ease of implementation of programmable systems is still vitally important. It would probably be more practical to produce a family of processors of differing wordlength but executing basically the same instruction set than to attempt to provide configurable wordlength. Ease of performing multi-precision arithmetic could be used to good advantage as an alternative to configurable wordlength, say on a 16-bit processor. Extra parallelism would have to be exploited to achieve the equivalent throughput. A family of processors could indeed differ in more than just wordlength, for example some could include floating point arithmetic and others just fixed point. Two new processors recently announced by AT&T, the WE DSP 16 and WE DSP 32 [26] differ in this way, although only advance information was available to the author, so it is not clear how similar they are in other ways. However, what is clear is that the DSP 16 provides very high performance 16 bit fixed point arithmetic, whereas the DSP 32 includes a 32-bit floating point processor.

7.2.1 Programmable Systems

As is well known, a large number of programmable signal processors have come onto the market during the execution of this project. None of them uses an approach remotely resembling the SPE based approach, which is probably just as well for the originality of this thesis, but also serves to confirm the conclusions above that it is

not the best way to proceed. The Texas TMS320 family is certainly one of the most popular, but many of the the others are not too dissimilar. The approach which has been successful in the TMS320 is to make each arithmetic cycle as fast as possible, and to tailor the instruction set to signal processing applications. The provision of a hardware multiplier has been seen above to provide increased silicon efficiency, and this is included in the TMS 320. Indeed an adder is also included which operates in parallel with the multiplier to enable one multiply-accumulate operation to be performed in each machine cycle. Clever design of the instruction set means that in that same cycle, memory addresses can be incremented to access the next data for the multiply-accumulate operation, enabling a multiply-accumulate operation to be performed every machine cycle. Furthermore, the data can be moved in memory, also in the same cycle, to implement the z^{-1} delay operation, albeit only on one data item at a time. Another nice feature is the implementation of bit-reversed memory addressing, which greatly eases the problem of data reordering for FFT calculation. The use of separate program and data memory effectively increases the processor-memory bandwidth, as indeed it does with the SPE, so that an instruction can be executed in only one machine cycle. The latest version of the TMS320, the 320C25, which has a cycle time of only 50ns can implement 20 multiply-accumulate operations per microsecond. It is claimed that it can implement a 1024 point complex FFT in only 7.1 ms.

Two areas of research in programmable systems seem to offer significant reward. The first is the connection of processors into networks, and the second is the establishment of functions which can usefully be hardwired into a programmable processor, or made particu-

larly easy to implement by clever instruction set design. The Inmos transputer shows great potential for the implementation of networks of processors. This is, after all, the key concept behind the transputer. Inter-processor communications can go on at the same time as arithmetic operations in the processors, providing yet another form of parallelism. Especially in view of the latest transputers incorporating floating point processors, the potential for signal processing applications is quite clear. Combining this with some of the clever instruction set design of the TMS320 would seem to show great promise.

The incorporation of hardware multipliers, multiplier-accumulators and even floating point processors has been seen to be very successful in improving the efficiency of programmable signal processors. These are, of course, still very general purpose tools, and indeed the signal processors described often also find application in general scientific calculations. One way to increase throughput further would be to increase the provision of hardwired functions. An obvious example would be complex multiplication, or even a whole FFT butterfly. It is now quite possible to integrate several hardware multipliers on a single chip, making such possibilities worth exploring. The drawback is, of course, that such functions could be totally useless and therefore wasted in many applications. It might, however, be quite possible to provide several multiplier-accumulators within a single processor, which could either be used to perform pre-configured operations such as the FFT butterfly or programmed by the user for parallel use. This certainly is an area of research worth pursuing. It may be decided, of course, that the best approach is simply to integrate several processors on

on chip or wafer, in a similar way to that envisaged for SPE based processors. In this case, very careful attention will need to be paid to the provision of the right amount of memory, and to inter-processor communications. The exploitation of VLSI complexity and beyond for programmable digital signal processing systems is a great challenge to ingenuity.

7.2.2 Hardwired Systems

As far as can be envisaged, there will be a continuing requirement for custom designed hardware to implement digital signal processing systems. Programmable systems may provide a solution for most audio frequency applications, and indeed possibly well above audio bandwidths. However, applications such as digital video and broadband communications systems will remain beyond the scope of programmable systems for the foreseeable future.

Recent hardwired signal processing systems have tended to use components such as multipliers and multiplier-accumulators, typically up to 24 by 24 bits, and having clock frequencies in excess of 10MHz. However, the days of such an approach are limited. It is now possible to integrate several multiplier-accumulators on a single chip, operating at similar frequencies, utilising advanced CMOS technology. The Inmos A100 [7] chip is an example. This implements a 32-stage FIR filter, and can process 16-bit data with 4-bit coefficients at a data rate of 10Mhz using a 20MHz clock. Coefficient wordlengths of 8, 12 and 16 bits take proportionately longer. The chip therefore effectively includes 32 multipliers of 4 by 16 bits. The output of the multiplier-accumulator array is 36 bits wide, and the chip output is 24 bits wide, selected from the 36 available. These chips will

on chip or wafer, in a similar way to that envisaged for SPE based processors. In this case, very careful attention will need to be paid to the provision of the right amount of memory, and to inter-processor communications. The exploitation of VLSI complexity and beyond for programmable digital signal processing systems is a great challenge to ingenuity.

7.2.2 Hardwired Systems

As far as can be envisaged, there will be a continuing requirement for custom designed hardware to implement digital signal processing systems. Programmable systems may provide a solution for most audio frequency applications, and indeed possibly well above audio bandwidths. However, applications such as digital video and broadband communications systems will remain beyond the scope of programmable systems for the foreseeable future.

Recent hardwired signal processing systems have tended to use components such as multipliers and multiplier-accumulators, typically up to 24 by 24 bits, and having clock frequencies in excess of 10MHz. However, the days of such an approach are limited. It is now possible to integrate several multiplier-accumulators on a single chip, operating at similar frequencies, utilising advanced CMOS technology. The Inmos A100 [7] chip is an example. This implements a 32-stage FIR filter, and can process 16-bit data with 4-bit coefficients at a data rate of 10Mhz using a 20MHz clock. Coefficient wordlengths of 8, 12 and 16 bits take proportionately longer. The chip therefore effectively includes 32 multipliers of 4 by 16 bits. The output of the multiplier-accumulator array is 36 bits wide, and the chip output is 24 bits wide, selected from the 36 available. These chips will

find easy application in video, and possibly radar, signal processing.

It becomes increasingly difficult to establish general purpose components as levels of integration increase, and in many cases it is desirable to implement the system in one chip or a very few chips. It is these considerations which have brought Application Specific Integrated Circuits (ASICs) to the fore in many application areas. Until recently, ASICs were not generally suited to signal processing applications, it making much more sense to utilise standard multiplier-accumulator chips, albeit possibly with gate arrays implementing some of the random logic. However, it is looking increasingly likely that ASICs will become more and more important for signal processing systems, and this seems to be a very promising area of research.

Traditional gate arrays and standard cell systems will not be a great deal of use in such systems. Some more advanced tools will be required. The FIRST silicon compiler [10] is an example of such a system, designed at the University of Edinburgh. This implements bit-serial signal processing systems, producing silicon layout directly from a textual specification. This specification is at a fairly low level, thus the designer has to tailor his design to the basic functions provided by the FIRST system. Nonetheless, this system represents a very important step forward in custom VLSI for signal processing.

There is currently much interest in the ASIC world in general cell systems, such as the Plessey Megacell system [4]. In a standard cell system, cells are all the same height and placed in rows on a

chip for easy routing. Although cells of varying width can be accommodated, the most complex cells available are similar to MSI TTL chips: counters, adders, etc. In general cell systems, cells of varying height and width can be accommodated. Thus larger subsystems such as RAM, ROM, PLA, and multipliers can be provided as cells. Moreover, these cells can often be parameterised, that is they can be produced automatically by software to a specification provided by the designer. This means he can have available multipliers of any desired configuration, for example 12 by 20 bits or whatever he may require. He may also be able to trade silicon area for performance in some cases. Such systems are clearly of interest to the designer of signal processing systems, as they could potentially provide him with just the building blocks he requires. Indeed he would be better off designing such systems based on parameterised cells, than designing with standard components, as these cannot be tailored precisely to his needs.

With such a general cell system available, it would be possible also to create higher level design tools, perhaps deserving of the name of silicon compiler. Such tools could take a specification from the designer, select an appropriate architecture, and generate the required cell specifications for the general cell system. For example, an FFT compiler could be conceived, where the designer would specify the number of points and performance required, and the compiler could select the appropriate level of parallelism and generate the chip automatically. This kind of tool could also be provided for other classes of function, such as digital filtering. In this case, the compiler could choose, or assist the designer in the choice, between recursive or transversal filtering, and in the latter case

between direct implementation or fast convolution via the FFT. For this kind of system, some kind of expert system will be required, for example to assist in the selection of architecture. The limited range of such architectures for signal processing suggest that real success could be achieved in developing such systems. This is the area in which the author would like to concentrate his research following the completion of this thesis.

APPENDIX A

FILTER USED FOR COMPARISON OF PROCESSORS

A simple filter design was chosen to be implemented on all the processors considered, in order to provide a means of comparing their performance. The filter selected was a 2nd. order Chebyshev low pass filter with the following parameters:

Filter Order: 2

Max. Ripple in Passband: 0.1

Cut-off frequency: 1 rad s^{-1}

Transfer Function: $H(s) = \dfrac{1}{s^2 + 1.522041s + 1.6583122}$

This filter was translated into the z-plane for implementation as a digital filter, using the bilinear z transform. The sampling frequency used was 50 times the cut-off frequency, so that sine waves of a reasonably smooth appearance could be observed at frequencies up to 5 times the cut-off frequency. It should be noted that the cut-off frequency of the filter need not necessarily be 1 Hz, but will always be $\frac{1}{50}$ times the sampling frequency. It is common practice to design digital filters with unity cut-off frequency in this way. The transfer function of the filter in the z-plane is:

$$H(z) = \frac{1 + 2z^{-1} + z^{-2}}{278.48711\ (1 - 1.7946004z^{-1} + 0.82625952z^{-2})}$$

which can be implemented as:

$$y_n = \frac{x_n + 2x_{n-1} + x_{n-2}}{278.48711} + 1.7946004y_{n-1} - 0.82625952y_{n-2}$$

In practice, it is not necessary to have the gain precisely correct, thus x_n can be divided by a suitable power of 2, in this case 1024 (2^{10}), to ensure that the filter gain is less than unity and to prevent arithmetic overflow.

BINARY VALUES OF COEFFICIENTS

$$1.7946004_{10} = 1.11001011011010101110111_2$$

$$0.82625952_{10} = 0.11010011100001011011111001_2$$

APPENDIX B

INTEL 2920 PROGRAM FOR TEST FILTER

```
 1      LDA     Y,Y1,L1
 2      SUB     Y,Y1,R2
 3      ADD     Y,Y1,R4
 4      SUB     Y,Y1,R5
 5      ADD     Y,Y1,R6
 6      SUB     Y,Y1,R8
 7      ADD     Y,Y1,R9
 8      SUB     Y,Y1,R11
 9      ADD     Y,Y1,R12
10      SUB     Y,Y1,R13
11      SUB     Y,Y2
12      ADD     Y,Y2,R2
13      SUB     Y,Y2,R3
14      ADD     Y,Y2,R4
15      SUB     Y,Y2,R6
16      ADD     Y,Y2,R9
17      ADD     Y,X,R10
18      ADD     Y,X1,R9
19      ADD     Y,X2,R10
20      LDA     Y2,Y1
21      LDA     Y1,Y
22      LDA     X2,X1
23      LDA     X1,X
```

APPENDIX C

SPE INSTRUCTION SET

In the following instruction descriptions, "aaaaaa" represents a 6-bit address, "ll" represents a 2-bit shift length where "ll" is one less than the shift length and "m" represents a bit determining whether or not the multiplier register is to be shifted.

Memory Operations

01aaaaaa	SACC	Store Accumulator
10aaaaaa	LMIER	Load Multiplier Register
11aaaaaa	LRB	Load Register B

Shift and I/O Operations

00m01011	SL	Shift Left
00m10011	SR	Shift Right
00m10100	XI	External Input
00m11000	XO	External Output
00m11100	XIO	External Input and Output

Arithmetic Instructions

00m00000	NOP	No Operation
00m00001	NEG	Negate Accumulator
00m00010	A0	Set Accumulator to Zero
00m00011	B0	Set Register B to Zero
00m00100	AB	Load Accumulator from Register B
00m00101	BA	Load Register B from Accumulator
00m00110	ADD	Add Register B to Accumulator
00m00111	SUB	Subtract Register B from Accumulator

Special Instructions

00m01100	COND	ADD, SUB or NOP conditional upon multiplier bits
00m01101	SMEM	Move memory by one location

APPENDIX D

HILO DESCRIPTION OF AN SPE

```
* Circuit header and external connections

    CCT SPE(MI,CO,LDP[3:0],IO[7:0],MIERX[1],CI,RDP[3:0],II[7:0],
            DSI,DSO,MSPE,PHI1,PHI2);


* Tri-state buffers for left and right data ports

    BUFIF1(1,1)
        LDPB[3:0](LDP[3:0],LDPX2[3:0],LDPEN)
        RDPB[3:0](RDP[3:0],AO[3:0],RDPEN);

    INPUT MI CI II[7:0] MSPE PHI1 PHI2;

    TRI LDP[3:0] RDP[3:0];


* Instruction registers including COND instruction decode

    REGISTER(1,1) IR[7:0]=VALCASE II[7:0],
        00?01100=VALCASE MIER[1:0],
            01={II[7:5],BIN 00110},
            10={II[7:5],BIN 00111},
            DEFAULT={II[7:5],BIN 00000) ENDCASE,
        DEFAULT=II[7:0] ENDCASE LOADIF1 PHI2;

    REGISTER(1,1) IO[7:0]=IR LOADIF1 PHI1;


* Arithmetic unit

    REGISTER(1,1) INA[3:0]=VALCASE IR[7:0],
        00?00001=NOT ACC[3:0],
        00?00010:00?00100=BIN 0000,
        DEFAULT=ACC[3:0] ENDCASE LOADIF0 BIN 0;

    REGISTER(1,1) INB[3:0]=VALCASE IR[7:0],
        00?00100:00?00110=BUS[3:0],
        00?00111=NOT BUS[3:0],
        DEFAULT=BIN 0000 ENDCASE LOADIF0 BIN 0;

    REGISTER(1,1) INCI=VALCASE IR[7:0],
        00?00001:00?00111=NOT CI,
        DEFAULT=CI ENDCASE LOADIF1 PHI2;

    REGISTER(1,1) CO=VALCASE IR[7:0],
        00?00001:00?00111=NOT AO[4],
        DEFAULT=AO[4] ENDCASE LOADIF1 PHI1;
```

```
* AU output register

    REGISTER(1,1) AO[4:0]=(BIN 0,INA[3:0])+(BIN 0,INB[3:0])+
            (BIN 0000,INCI) LOADIF1 PHI1;


* Single cycle delay for left data port

    REGISTER(1,1) LDPX1[3:0]=AO[3:0] LOADIF1 PHI2;

    REGISTER(1,1) LDPX2[3:0]=LDPX1 LOADIF1 PHI1;

    REGISTER(1,1) LDPEX1=AND((IO[7:6],IO[4:2]) EQV BIN 00010)
            LOADIF1 PHI2;

    REGISTER(1,1) LDPEX2=LDPEX1 LOADIF1 PHI1;


* Enable signal for left and right data port tri-state buffers

    REGISTER(1,1) LDPEN=LOADCASE (PHI1,PHI2),
        01=LDPEX2,
        10=0 ENDCASE;

    REGISTER(1,1) RDPEN=LOADCASE (PHI1,PHI2),
        01=NAND((IO[7:6],IO[4:2]) EQV BIN 00010),
                                        10=0 ENDCASE;


* Multiplier register

    REGISTER(1,1) MIERX[4:0] = VALCASE IR[7:6],
        10 = (BUS[3:0],BIN 0),
        DEFAULT = MIER[4:0] ENDCASE LOADIF1 PHI1;

    REGISTER(1,1) MIER[4:0] = VALCASE IO[7:5],
        001 = (MI,MIERX[4:1]),
        DEFAULT = MIERX[4:0] ENDCASE LOADIF1 PHI2;


* RB Register

    REGISTER(1,1) RB[3:0]=LOADCASE (PHI1,IR[7:6],IR[4:0]),
        111?????:10000011:10000101=BUS[3:0] ENDCASE;


* Data memory

    RAM(0:63) MEM[3:0];


* Main communications bus

    REGISTER(1,1) BUS[3:0]=LOADCASE (PHI1,PHI2,IR[7:6],IR[4:0]),
        01???????=RB[3:0],
        1001?????:100000011:100000101=ACC[3:0],
        101??????=MEM[IR[5:0]] ENDCASE;
```

```
* Accumulator incorporating barrel shifter operation

    REGISTER(1,1) ACC[3:0]=VALCASE (IO[7:0],MSPE),
        00?01000?=(AO[2:0],RDP[3]),
        00?01001?=(AO[1:0],RDP[3:2]),
        00?01010?=(AO[0],RDP[3:1]),
        00?01011?=RDP[3:0],
        00?100000=(LDP[0],AO[3:1]),
        00?100010=(LDP[1:0],AO[3:2]),
        00?100100=(LDP[2:0],AO[3]),
        00?100110=LDP[3:0],
        00?100001=(AO[3],AO[3:1]),
        00?100011=(AO[3],AO[3],AO[3:2]),
        00?1001?1=(AO[3],AO[3],AO[3],AO[3]),
        00?10100?:00?11?00?=LDP[3:0],
        DEFAULT=AO[3:0] ENDCASE LOADIF1 PHI2;


* Data Strobe In and Data Strobe Out:
* - used only to support test simulations

    REGISTER(1,1) DSI=LOADCASE (PHI2,IO[7,6,4,2]),
        10011=1,
        0????=0 ENDCASE;

    REGISTER(1,1) DSO=LOADCASE (PHI2,IO[7,6,4,3]),
        10011=1,
        0????=0 ENDCASE;


* Program for writing and shifting memory

    REGISTER(1,1) XADR[5:0];

    WHEN PHI1(1 TO 0) DO CASE IR[7:0],
        01?????? - MEM[IR[5:0]]=BUS,
        00?01101 - EVENT X1 ENDCASE;

    WHEN X1 DO XADR=63 EVENT X2;

    WHEN X2 WAIT 1 DO IF XADR >= 32 THEN
            MEM[XADR]=MEM[XADR-1] EVENT X3 ENDIF;

    WHEN X3 WAIT 1 DO XADR=XADR-1 EVENT X2.
```

```
* Accumulator incorporating barrel shifter operation

    REGISTER(1,1) ACC[3:0]=VALCASE (IO[7:0],MSPE),
        00?01000?={AO[2:0],RDP[3]},
        00?01001?={AO[1:0],RDP[3:2]},
        00?01010?={AO[0],RDP[3:1]},
        00?01011?=RDP[3:0],
        00?100000={LDP[0],AO[3:1]},
        00?100010={LDP[1:0],AO[3:2]},
        00?100100={LDP[2:0],AO[3]},
        00?100110=LDP[3:0],
        00?100001={AO[3],AO[3:1]},
        00?100011={AO[3],AO[3],AO[3:2]},
        00?1001?1={AO[3],AO[3],AO[3],AO[3]},
        00?10100?:00?11?00?=LDP[3:0],
        DEFAULT=AO[3:0] ENDCASE LOADIF1 PHI2;


* Data Strobe In and Data Strobe Out:
* - used only to support test simulations

    REGISTER(1,1) DSI=LOADCASE (PHI2,IO[7,6,4,2]),
        10011=1,
        0????=0 ENDCASE;

    REGISTER(1,1) DSO=LOADCASE (PHI2,IO[7,6,4,3]),
        10011=1,
        0????=0 ENDCASE;


* Program for writing and shifting memory

    REGISTER(1,1) XADR[5:0];

    WHEN PHI1(1 TO 0) DO CASE IR[7:0],
        01?????? - MEM[IR[5:0]]=BUS,
        00?01101 - EVENT X1 ENDCASE;

    WHEN X1 DO XADR=63 EVENT X2;

    WHEN X2 WAIT 1 DO IF XADR >= 32 THEN
            MEM[XADR]=MEM[XADR-1] EVENT X3 ENDIF;

    WHEN X3 WAIT 1 DO XADR=XADR-1 EVENT X2.
```

REFERENCES

1. G.A. Anderson and E.D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics and Examples", Computing Surveys, vol 7 no. 4, December 1975

2. Intel, "2920-10 Signal Processor", Data sheet, 1979.

3. A.W. Burks, H.H. Goldstine and J. Von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument", Part 1, Vol I, Report prepared for US Army Ordnance Dept, 1946 in J. Von Neumann, "Collected Works vol V: Design of computers, theory of automata and numerical analysis", ed A.H. Taub, Pergamon press, Oxford, 1963.

4. J.S. Brothers, J.W. Tomkins and J.S. Williams, "The Megacell Concept: an approach to painless custom design", Proc. IEE pt. E, 132, pp91-98, 1985.

5. H. Kurokawa et al., "The architecture and performance of Image Pipeline Processor", Proc VLSI 83 Conference, North-Holland, 1983, pp275-284.

6. B.A. Bowen and W.R. Brown, "VLSI Systems Design for digital signal processing", Prentice-Hall, 1982.

7. Inmos, "IMSA100 Cascadable Signal Processor", preliminary data sheet, June 1986.

8. M.G.H. Katevenis et al., "The RISC-II Micro-architecture", Proc. VLSI 83 Conference, North-Holland, 1983.

9. R.F. Lyon, "A Bit-Serial VLSI Architectural Methodology for Signal Processing", Proc VLSI 81 Conference, Academic Press, 1981, pp131-140.

10. P.B. Denyer, D. Renshaw and N. Bergmann, "A Silicon Compiler for VLSI Signal Processors", Proc ESSCIRC 82, pp215-218.

11. A. Peled and B. Liu, "Digital Signal Processing, Theory, Design and Implementation", Wiley, 1976

12. J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Mathematics of Computation, vol 19, no 90, 1965 pp297-301

13. A.V. Oppenheim, ed., "Applications of Digital Signal Processing", Prentice-Hall, 1978

14. Barnes, et al., "The ILLIAC-IV Computer", IEEE Transactions on Computers, vol C-17 no 8, August 1968, pp 746-757.

15. Watson, W.J., "The TI ASC - A Highly Modular and Flexible Super Computer Architecture", Proc AFIPS, 1972 FJCC, Vol 41 pp221-228, AFIPS Press, Montvale, NJ, 1972

16. Gold, B., et al., "The FDP, A Fast Programmable Signal Processor", IEEE Transactions on Computers, vol C-20, pp33-38.

17. R.W. Blasco, "V-MOS chip joins microprocessor to handle signals in real time", Electronics magazine, 30 August 1979.

18. S.S. Magar and D.A. Robinson, "Microprogrammable Arithmetic Element and its Application to Digital Signal Processing, IEE Proc., vol 127 part F no 2, April 1980, pp99-106.

19. W.K. Luk and H.F. Li, "Microcomputer-based real-time/online fft processor", IEE Proc., vol 127 part E no 1, January 1980, pp18-23.

20. E.E. Swartzlander and D.J. Heath, "A Routing Algorithm for Signal Processing Networks", IEEE Transactions on Computers vol C-28 no 8, August 1979 pp567-572.

21. S.I. Kartashev and S.P. Kartashev, "A Multicomputer System with Dynamic Architecture", IEEE Transactions on Computers, vol C-28 no. 10, October 1979, pp704-721.

22. Intel, "MSC-85 User's manual", June 1977.

23. Intel, "The 8086 Family User's Manual", October 1979.

24. I-Ngo Chen and Robert Willoner, "An O(n) Parallel Multiplier with Bit-Sequential Input and Output", IEE Trans. Comput., vol C-28 no. 10, pp721-727, October 1979.

25. Carver Mead and Lynn Conway, "Introduction to VLSI Systems", Addison-Wesley, 1980.

26. AT&T, "WE DSP16" and "WE DSP32", Product Descriptions, 1986.

27. GenRad, "HILO-2 User Manual", 1984.

28. L.W. Nagel, D.O. Pederson, "Simulation program with integrated circuit emphasis", Proc 16th Midwest Symp. Circ. Theory, Waterloo, Canada, April 1973.

29. A. Matthews, "Digital Processes Group Microcode System User Manual", UMIST internal report, 1981.

30. Neil Weste and Kamran Eshraghian, "Principles of CMOS VLSI Design", Addison-Wesley, 1985.

31. J. Mavor, M.A. Jack and P.B. Denyer, "Introduction to MOS LSI Design", Addison-Wesley, 1983.

32. D. Linden, "A Discussion of sampling theorems", Proc. IRE, vol 47, pp1219-1226, July 1959.

33. M.M. Sondhi, "An Adaptive Echo Canceller", Bell System Tech. J., vol 46, no 3, March 1967, pp497-511.

34. A.V. Oppenheim and R.W. Schaffer, "Homomorphic Analysis of Speech", IEEE Trans. Audio Electroacoustics, vol. AU-16, 1968, pp221-226.

35. Texas Instruments, "Digital Signal Processor: TMS320 Product Description", 1984.

36. Iann Barron, et al., "Transputer does 5 or more MIPs even when not used in parallel", Electronics magazine, vol 17, 1983, pp109-115.

37. B. Konemann, J. Mucha and G. Zwiehoff, "Built-In Logic Block Observation technique", Proc. 1979 IEEE Test Conference, pp37-41.