## *Design of microprocessor-based hardware for number theoretic transform implementation*

Shamim, Anwar Ahmed

# DESIGN OF MICROPROCESSOR-BASED HARDWARE FOR NUMBER THEORETIC TRANSFORM IMPLEMENTATION

by

Anwar Ahmed Shamim B.Sc., M.Sc.



A thesis submitted in accordance with the regulations for the degree of Doctor of Philosophy at the University of Durham, Department of Applied Physics and Electronics.

1983

# Design of Microprocessor-Based Hardware for Number Theoretic Transform Implementation

Anwar Ahmed Shamim

## ABSTRACT

Number Theoretic Transforms (NTTs) are defined in a finite ring of integers $Z_M$, where M is the modulus. All the arithmetic operations are carried out modulo M. NTTs are similar in structure to DFTs, hence fast FFT type algorithms may be used to compute NTTs efficiently. A major advantage of the NTT is that it can be used to compute error free convolutions, unlike the FFT it is not subject to round off and truncation errors.

In 1976 Winograd proposed a set of short length DFT algorithms using a fewer number of multiplications and approximately the same number of additions as the Cooley-Tukey FFT algorithm. This saving is accomplished at the expense of increased algorithm complexity. These short length DFT algorithms may be combined to perform longer transforms.

The Winograd Fourier Transform Algorithm (WFTA) was implemented on a TMS9900 microprocessor to compute NTTs. Since multiplication conducted modulo M is very time consuming a special purpose external hardware modular multiplier was designed, constructed and interfaced with the TMS9900 microprocessor. This external hardware modular multiplier allowed an improvement in the transform execution time.

Computation time may further be reduced by employing several microprocessors. Taking advantage of the inherent parallelism of the WFTA, a dedicated parallel microprocessor system was designed and constructed to implement a 15-point WFTA in parallel. Benchmark programs were written to choose a suitable microprocessor for the parallel microprocessor system. A master or a host microprocessor is used to control the parallel microprocessor system and provides an interface to the outside world. An analogue to digital (A/D) and a digital to analogue (D/A) converter allows real time digital signal processing.

## ACKNOWLEDGEMENTS

I owe my unbound gratitude to Dr. B. J. Stanier for his guidance, constructive criticism, invaluable suggestions and kindly help throughout the period of this project.

I should like to thank Prof. G. G. Roberts for allowing me to use the facilities of the Department of Applied Physics and Electronics. I am grateful to my colleagues for valuable discussions. Furthermore, I am also thankful to the members of the workshop for providing the necessary and technical assistance.

I should also like to thank the computer unit and the library staff for their co-operation.

I greatly acknowledge the moral and financial support I received from my parents, brothers, sisters, Mr. A. Khan and family and H. Khan. I also extend my appreciation to Prof. N. A. Khan and other friends for their moral support.

My special thanks are due to Dr. M. Ahmed and the South Fields Trust, London for providing the financial support.

Dedicated To My Affectionate Parents

Who Inspired Me To Higher Ideals Of Life

CONTENTS

# CHAPTER 1

## Introduction

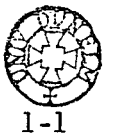
The aim of this work was to design hardware to facilitate the implementation of the Winograd Fourier Transform Algorithm (WFTA) to compute Number Theoretic Transforms (NTTs) on microprocessors.

Microprocessors are easy to implement and provide cheap integer processing power. In recent years there has been a major breakthrough in the solid state technology, which is responsible for providing highly reliable hardware.

Cooley and Tukey (8), described a fast and efficient method to compute the Discrete Fourier Transform (DFT) via the Fast Fourier Transform (FFT) algorithm (2). The FFT is subject to truncation and round off errors, since it involves multiplications with complex irrational roots of unity, which cannot be represented accurately on a finite precision machine.

Number Theoretic Transforms on the other hand have a similar structure to DFTs, and are defined in a finite ring of integers $Z_M$, where M is the modulus. All the arithmetic operations are carried out modulo M. Fast FFT type algorithms may also be used to compute NTTs without round off errors (9) - (15), (20), (80). The results thus obtained are exact.

Winograd (3), (4), proposed short length DFT algorithms which show improvement over the conventional FFT algorithm. The



WFTA requires fewer multiplications, and roughly the same number of additions as the Cooley-Tukey FFT algorithm. In the FFT the transform length is restricted to powers of 2, but in the WFTA the transform length is the product of several mutually prime factors. These mutually prime factors are chosen from the short length (small-N) WFTA. Transform lengths from 2 to 5040 may be implemented. Implementation of the WFTA requires some constants to be precomputed and stored in the memory which requires more memory than the comparable length FFT (51). The WFTA requires less multiplications, but at the expense of increased algorithm complexity and more data transfers (52).

Martin (5), (6), carried out a search for a suitable modulus M for 16-bit arithmetic on the lines described by Bailey (53), and found that M = 65521 is suitable for NTT implementation. Agarwal and Burrus (9), have shown that the transform lengths are subject to certain constraints.

1- N must divide O(M), where O(M) is greatest common divisor (g.c.d) of the set of prime divisors $(p_i - 1)$ of M.

$$O(M) = \text{g.c.d}\ (p_i - 1)$$

2- An element $\alpha$ of order N must exist such that

$$\alpha^N \equiv 1 \bmod M, \alpha^r \not\equiv 1 \bmod M, \quad \forall\, r < N.$$

3- $N^{-1}$ must exist in the ring $Z_M$. If M is not prime, then $N^{-1}$ may or may not exist. $N \cdot N^{-1} \equiv 1 \bmod M$.

4- N must be well factored for fast transform algorithms to exist.

5- To implement fast and simple arithmetic mod M, M and $\alpha$ must have simple binary representation.

No attempt has been made to compare the WFTA and the FFT nor to derive any of the algorithms. Martin (5), have discussed these topics in detail. Here we will emphasise more the hardware design and implementation to compute NTT via WFTA. McClellan and Rader (7), provide good references for the NTT and the WFTA.

In chapter 2 basic number theory and Number Theoretic Transforms, and some fundamental concepts about rings, fields, and modular arithmetic are described. A brief discussion about Mersenne Number Transforms (MNTs) and Fermat Number Transforms (FNTs) is also presented.

Chapter 3 describes different algorithms for signed and unsigned multiplication suitable for microprocessors. Multiplication using ROM lookup is also described, this method provides a fast way of multiplying two numbers. However, the applications may be limited since the size of the ROM increases rapidly as the size of the input numbers increase. Fast multiplier chips are now available which may replace several discrete components. Finally 16-bit modular arithmetic operations for a microprocessor are described.

Chapter 4 describes a step by step approach towards the implementation of the WFTA to compute the NTT. The WFTA was implemented on the TMS9900 microprocessor (54), (55), using Assembler and FORTH (56), (57), languages. The WFTA was also implemented on the MC6809 microprocessor (78), (79), using Assembler language, and in FORTRAN and Assembler on IBM mainframe computers (370/168 and 370/4341).

The total transform execution time on a processor depends upon the number of operations and the time required to execute each operation. Ordinary microprocessors do not have hardware multiplication, even microprocessors with hardware multiply require a considerable amount of time for multiplication. Modular arithmetic operations and in particular modular multiplication, are very slow. Chapter 5 describes a special purpose (16 x 16-bit) external hardware modular multiplier (mod 65521) interfaced with the TMS9900 microprocessor. This modular multiplier behaves as an intelligent memory mapped peripheral. We shall use the term **modular** for the results reduced **modulo M.** This external modular multiplier uses multiplier chips and ROM lookup techniques to generate the modular product. Finally comparison of timings for the implementation of WFTA with and without using the external hardware modular multiplier are discussed.

Chapter 6 provides prerequisite information and describes some of the basic concepts of parallel and multi processor systems. In addition inter processor communication, array processors and processor to memory interconnection is also

described.

The difficulties involved in the uni processor implementation of the WFTA is that it requires more data transfers and indexing in the memory to acquire data (52). Since the WFTA exhibits parallelism in its structure, the possibility of parallel implementation of the WFTA was investigated. Chapter 7 describes design and construction of a parallel microprocessor system to implement a 15-point WFTA.

Benchmark programs were written to choose a suitable microprocessor for the design of a parallel microprocessor system. Motorola's MC6809 microprocessor gave an optimum choice among several microprocessors. To investigate the principle of data exchange between the two microprocessors, a two microprocessor system (using MC6809) was designed and tested. The TMS9900 microprocessor was used as a host processor.

Since the modular multiplication is the most time consuming operation, the parallel microprocessor system was designed such that each of the microprocessor is loaded equally during the modular multiplication. A control or a master microprocessor is used to control the parallel structure. The control microprocessor provides communication between the parallel microprocessor system and the outside world. Inter microprocessor communication is through dedicated latches. The system configuration is that of a master and slave, all the input/output (I/O) data is through the master microprocessor.

The system design is described, and the timings for parallel and uni processor implementation of the 15-point WFTA are discussed. Finally a 15-point convolution was also implemented on the parallel microprocessor system. The software development is the bottleneck of the parallel microprocessor system.

It was found that the execution time of a 15-point WFTA on the parallel microprocessor system is comparable with the execution time on IBM mainframe computers.

Software routines are listed in appendix-A to appendix-D. Appendix-E contains backplane wiring connections for the parallel microprocessor system.

---

Fully documented program listings appearing in the appendices A - D are available in a separate folder.

---

# CHAPTER 2

## Elementary Number Theory and Number Theoretic Transforms

### 2.1 Introduction

The Discrete Fourier Transform (DFT) of a sequence x(n) is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W^{nk} \qquad (2.1)$$

where k = 0,1,2,...,N-1. The Inverse Discrete Fourier Transform (IDFT) is given by:

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\, W^{-nk} \qquad (2.2)$$

where n = 0,1,2,...,N-1, and $W = e^{-j2\pi/N}$, $j = \sqrt{-1}$.

$W_N$ (usually written as W) is the principal root of unity such that $W^N \equiv 1 \bmod N$, where N is the sequence length.

Direct computation of equation (2.1) requires $N^2$ complex operations. A complex operation is a multiplication followed by an addition. On a digital computer multiplication of two numbers requires more computation time than the addition of two numbers. The multiplication time depends entirely on the software and the hardware available. To improve the efficiency and to compute equation (2.1) faster, the number of multiplications must be reduced. Various algorithms are available which are more efficient than the direct computation of equation (2.1).

In 1965 Cooley and Tukey (8), presented their FFT (Fast Fourier Transform) algorithm. This algorithm efficiently computes DFT given by equation (2.1). The number of complex operations are reduced from $N^2$ to $N\log_2 N$. This fractional saving of $N/\log_2 N$ becomes quite appreciable for sequence lengths greater than N = 32. It is required by the algorithm for N to be highly composite and a power of 2, such that $N = 2^m$, where m is a positive integer. Reference (2), provides theoretical development of the FFT algorithm in detail.

The Fourier Transforms are complex in general. The computation of equation (2.1) using the FFT requires multiplications with complex irrational roots of unity. These irrational roots cannot be represented accurately on a finite precision machine. The FFT is subject to cumulative roundoff and truncation errors. This gives rise to noise at the output of digital signal processing system, thus deteriorating the signal-to-noise ratio.

## 2.2 Discrete Fourier Transform and the Convolution

A common problem in digital signal processing is the implementation of convolution which is defined by:

$$y(n) = \sum_{i=0}^{N-1} x(i)\, h(n-i) \qquad (2.3)$$

where n = 0,1,2,...,N-1, y(n) is the convolution of two sequences x(n) and h(n). Direct implementation of convolution by using

equation (2.3) is not efficient. However, the Discrete Fourier Transform (DFT) can be used to compute convolution efficiently. Certain transform possess the Cyclic Convolution Property (CCP), which may be represented as follows:

$$T(y) = T(h) \cdot T(x) \qquad (2.4)$$

where '.' denotes pointwise multiplication. The inverse of equation (2.4) is given by:

$$y = T^{-1}\left[T(h) \cdot T(x)\right] \qquad (2.5)$$

So a cyclic (circular) convolution may be performed by taking the inverse transform ($T^{-1}$) of the product of the transforms of the two sequences to be convolved.

Let X(k) and H(k) be the Fourier transforms of the sequences x(i) and h(i) respectively. Then from equation (2.5) we have:

$$y(n) = N^{-1} \sum_{k=0}^{N-1} H(k)\, X(k)\, W^{-nk} \qquad (2.6)$$

Substituting value of X(k) in equation (2.6) we get,

$$y(n) = N^{-1} \sum_{k=0}^{N-1} H(k) \sum_{i=0}^{N-1} x(i)\, W^{ik}\, W^{-nk}$$

$$= \sum_{i=0}^{N-1} x(i)\, N^{-1} \sum_{k=0}^{N-1} H(k)\, W^{-k(n-i)}$$

$$= \sum_{i=0}^{N-1} x(i)\, h(n-i)$$

To obtain an N point circular convolution of the sequence h(n-i), if the sequence length is less than N it must be periodically extended to have a period of N. Hence

$$y(n) = \sum_{i=0}^{N-1} x(i)\, h(n-i \bmod N) \qquad (2.7)$$

$$= x(i) * h(i)$$

where * denotes convolution.

Equation (2.7) shows circular convolution, it is so called since it evaluates y(n) as if the input sequence were periodically extended outside the range $\left[0 \text{ to } N-1\right]$. This may also be stated as that for cyclic convolution the indices are evaluated mod N. If zeros are appended to the sequence so as to avoid aliasing or overlapping, the cyclic convolution gives the same results as conventional convolution. Convolution computed via equation (2.5) is computationally efficient when the sequence length is highly composite, so that FFT type algorithms can be applied to it.

## 2.3 Congruence

Consider two elements a,b of a set. Then for b a positive integer, if b is a factor of a we can write

$$a = qb + r \qquad \text{for } 0<r<q \tag{2.8}$$

where q represents the quotient and r the remainder. Equation (2.8) basically represents a division operation. If the remainder r = 0 then we say that b divides a and is represented as $b \mid a$. For all integers in the set there are at least two divisors for each element, either $1 \mid a$ or $a \mid a$. This condition indicates that a is a prime, with no divisors except 1 and itself. If r = 0 then we say that a is composite a=qb. Either q or b or both can be prime or composite. For q and b composite we can further factorise until we get prime factor factorisation which is written as:

$$a = \prod_i p_i^{r_i}$$

where $p_i$ is a prime and $r_i$ is an integer exponent. In equation (2.8) if b is a fixed number then it is called the modulus. Then for infinitely large number of values of a we can have the same value of the remainder r. All these values of a which give the same value of r are said to be congruent and are denoted by $\equiv$. The remainder r is called the residue mod b, or simply the residue. For example, let b = 5. Then $7 \equiv 2 \bmod 5$, $12 \equiv 2 \bmod 5$, and $17 \equiv 2 \bmod 5$. Numbers 7, 12, 17 are congruent mod 5. In general we can write

$$a \equiv r \bmod b$$

or $$b \mid (a-r)$$

also if $a \equiv 0 \bmod b$ then $b \mid a$. Some notations also use angle

brackets to represent the modulus, for example:

$$\langle 12 \rangle_5 \quad \text{and} \quad \langle 13 + 8 \rangle_5$$

The following conditions hold for congruence

$$\langle l + m \rangle_b \equiv \langle \langle l \rangle_b + \langle m \rangle_b \rangle_b$$

$$\langle l - m \rangle_b \equiv \langle \langle l \rangle_b - \langle m \rangle_b \rangle_b$$

$$\langle lm \rangle_b \equiv \langle \langle l \rangle_b \langle m \rangle_b \rangle_b$$

The largest number which can divide a and b is called the greatest common divisor (g.c.d). If the two numbers a and b are mutually prime i.e. they have no common factors then they are represented as (a,b) = 1, or a and b have a common factor of 1, for example (3,4) = 1, and (3,5) = 1, etc. However, if there is a common divisor then (8,10) = 2.

## 2.4 Chinese Remainder Theorem (CRT)

If the residue is known for several mutually prime moduli then with the help of the Chinese Remainder Theorem (CRT) these residues can be combined to give the result modulo the product of all the mutually prime factors.

Let a set of simultaneous congruences be given for which each of the moduli $m_i$ are relatively prime. For each i, $b_i$ is determined through linear congruences. The solution of the set of congruences is given by:

$$y = a_1 b_1 M/m_1 + a_2 b_2 M/m_2 + \ldots + a_j b_j M/m_j \qquad (2.9)$$

where $y \equiv a_i \bmod m_i$, and composite modulus M is given by:

$$M = \prod_i m_i \qquad (2.10)$$

provided that $m_i$ are relatively prime, $b_i$ are defined such that:

$$b_i\ (M/m_i) \equiv 1 \bmod m_i$$

For example, let $x \equiv 2 \bmod 3$ , $x \equiv 2 \bmod 5$, $x \equiv 4 \bmod 7$. To solve these simultaneous congruences first we get the product of mutually prime factors according to (2.10). Hence

$$M = 3 \cdot 5 \cdot 7 = 105$$

Now from (2.9)

$$x = 2\ b_1\ 105/3 + 2\ b_2\ 105/5 + 4\ b_3\ 105/7$$
$$= 2 \cdot 35 \cdot b_1 + 2 \cdot 21 \cdot b_2 + 4 \cdot 15 \cdot b_3 \qquad (2.11)$$

Now to determine $b_1$, $b_2$, $b_3$ such that

$$35 \cdot b_1 \equiv 1 \bmod 3 \implies b_1 = 2$$
$$21 \cdot b_2 \equiv 1 \bmod 5 \implies b_2 = 1$$
$$15 \cdot b_3 \equiv 1 \bmod 7 \implies b_3 = 1$$

substitution of these values in (2.11) gives

$$x = 70 \cdot 2 + 42 \cdot 1 + 60 \cdot 1 = 242 \equiv 32 \bmod 105$$

## 2.5 Groups, Rings and Fields

Recall from the previous section that

$$a = b + Mc \qquad (2.12)$$

where b is the remainder, c is an integer (quotient) and M the modulus. Then (2.12) may be rewritten as

$$a \equiv b \bmod M \quad \forall \quad a,b \in [1, M-1]$$

In a finite set $[a,b,c,\ldots,M-1]$ of integers all the elements are congruent to some integer called the modulus M. Such a set is denoted as $Z_M$. Let there be an operation * defined in $Z_M$, then the following conditions hold.

1- Closure : $a * b \quad \forall \; a,b \in Z_M$

2- Associative : $(a * b) * c = a * (b * c) \quad \forall \; a,b,c \in Z_M$

3- Identity element : $a * I = I * a = a \quad \forall \; a,I \in Z_M$

4- Inverse element : $a * a^{-1} = I \quad \forall \; a,a^{-1} \in Z_M$

5- Commutative : $a * b = b * a \quad \forall \; a,b \in Z_M$

Where I represents an identity element and $a^{-1}$ is the inverse of a. If the operation * is defined as ordinary addition then property 4 represents subtraction, and for ordinary multiplication it represents division.

If these properties hold then the set of integers $Z_M$ is called a group under the operation *. A group which obeys the commutative law is called an abelian group or a commutative group. A group is called a cyclic group if all the elements of the group can be generated from a single element, this element is called a generating function. For example 1 is a generating function under addition mod M. For a group $Z_M$ under ordinary addition '+' and ordinary multiplication '.' operations if the following distributive laws hold,

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$

$$(a + b) \cdot c = a \cdot c + b \cdot c$$

$\forall \; a,b,c \in Z_M$, then the group is called a ring.

Consider some examples of arithmetic mod 11, the elements in the ring $Z_M$ are $\left[0,1,2,\ldots,10\right]$.

1- Addition : $5 + 8 = 13 \equiv 2 \bmod 11$

2- Negation : $-3 = 11 + (-3) \equiv 8 \bmod 11$

3- Subtraction : $3 - 7 = 3 + (11 - 7) = 3 + 4 \equiv 7 \bmod 11$

4- Multiplication : $5 \cdot 4 = 20 \equiv 9 \bmod 11$

5- Multiplicative inverse : $6 \cdot 2 = 12 \equiv 1 \bmod 11$

6 and 2 are multiplicative inverses of each other

or $6^{-1} \equiv 2 \bmod 11$ or $2^{-1} \equiv 6 \bmod 11$

6- Division : a/b is defined if and only if $b^{-1}$ exists,

therefore, $a/b \equiv a \cdot b^{-1} \bmod M$

consider $9/2 = 9 \cdot 6 = 54 \equiv 10 \bmod 11$

from property 5, 6 and 2 are inverses of each other.

The element 2 is an integer root of unity of order 10,

$2^5 \equiv -1 \bmod 11$

$2^{10} \equiv 1 \bmod 11$

## 2.6 Number Theoretic Transforms

One group of transforms having the CCP are those with DFT like structure. Let

$$X(k) = T\, x(n), \quad \text{so} \quad x(n) = T^{-1}\, X(k)$$

$$X(k) = \sum_{n=0}^{N-1} x(n)\, \alpha^{nk} \qquad (2.13)$$

where k = 0,1,2,...,N-1.

The inverse is given by:

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\, \alpha^{-nk} \qquad (2.13a)$$

Where $\alpha$ is an element of order N, and plays the same role as W in equation (2.1). Where N is the least positive integer such that

$\alpha^N \equiv 1 \bmod M$, $\alpha, N \in [0, M-1]$. NTTs use modular arithmetic and possess the CCP.

Euler's function or Euler's totient function is defined as the number of integers in the ring $Z_M$ which are relatively prime to a given modulus M. This function is represented by $\emptyset(M)$. If M is composite then $\emptyset(M)<M$, but if M is prime then the Euler's function $\emptyset(M)= M-1$, for example $\emptyset(6) = 2$, and $\emptyset(7) = 6$.

$$\emptyset(M) = M(1-1/p1)(1-1/p2)...(1-1/pr)$$

where p1,p2,...,pr are different primes dividing M.

Euler's theorem states that for any non zero element a in the ring $Z_M$, which is relatively prime to M, (a,M) = 1, the following congruence holds

$$a^{\emptyset(M)} \equiv 1 \bmod M$$

If M is prime then O(M) = M-1 and the Euler's theorem reduces to Fermat's theorem given by:

$$a^{M-1} \equiv 1 \bmod M$$

The necessary and sufficient condition for the NTT with the CCP to exist is that $N \mid O(M)$, where O(M) is the greatest common divisor (g.c.d) given by:

$$O(M) = \text{g.c.d } (p_1 - 1)(p_2 - 1)...(p_r - 1) \qquad (2.14)$$

Thus the maximum transform length $N_{max} = O(M)$.

When the transforms in equation (2.13) and (2.13a) are defined in a finite ring of integers with the CCP, they are known as Number Theoretic Transforms (NTT) (7), (9) - (15), (80). In NTTs all the arithmetic operations are conducted mod M. There

are several constraints between the modulus M and the transform length N (9). Since the NTTs are similar in structure to the DFTs any algorithm which applies to the DFT can be applied to the NTT. In other words an NTT is a DFT with the CCP defined in a finite ring of integers under addition and multiplication. Such a ring is denoted by $Z_M$. If the modulus M is a composite number then the multiplicative inverses of all the elements do not exist. Hence $Z_M$ is a field if and only if M is prime. If $\alpha$ is of the order of $\emptyset(M)$, (where $\emptyset(M)$ is the Euler's totient function), then $\alpha$ is called the primitive root or the generating function, the non-zero elements of $Z_M$ can be generated by the powers of the primitive root.

The results obtained by NTTs are exact and are not subject to cumulative round off or truncation errors. For computing convolutions using NTTs, the choice of the modulus M has to be made first, then the corresponding N and $\alpha$ may be evaluated.

In a ring of integers $Z_M$, integers may be represented unambiguously if their absolute value is less than M/2. If the two sequences to be convolved x(n) and h(n) are scaled such that y(n) never exceeds M/2, then the convolution in the ring of integers mod M gives the same results as normal arithmetic. In most practical applications the impulse response of a digital system h(n) and the peak amplitude of the input x(n) signal is usually known.

For efficient implementation of convolution using NTTs the algorithm should be computationally efficient. Also N should be highly composite and the modulus large enough to provide a large

dynamic range of numbers. By suitable choice of N, M and $\alpha$ it is possible to define NTTs which can be computed efficiently. If N is chosen to be a power of 2 the efficiency of the FFT algorithm can be applied for computation. Binary representation of $\alpha$ should also be simple, such that the multiplication could be performed with ease. For $\alpha$ = 2 or a power of 2 the multiplications are reduced to bit shifts and add.

Discrete convolution may also be obtained by either Mersenne Number Transform (MNT) or Fermat Number Transform (FNT). These transforms are special cases of Number Theoretic Transforms. The multiplications in MNT and FNT are reduced to circular bit shifts within the word and add (12), (13), (14), (24). On a digital computer most of the computation time is taken by the multiplication. The situation is even worse on a microprocessor because ordinary microprocessors do not have hardware multipliers. Software implementation of the modular multiplication requires more time. External hardware modular multiplier may be implemented to facilitate modular multiplication. So transforms which do not require multiplications at all such as the MNT and FNT are computationally more efficient.

### 2.6.1 Mersenne Number Transforms

If the modulus is chosen to be a Mersenne number ($M_p$), then the transforms defined in a ring with CCP are called Mersenne Number Transforms (MNT). The mersenne numbers are defined as follows:

$$M_p = 2^p - 1$$

where p is prime. Mersenne numbers are of interest only if p is prime.

Rader (12), have described method for computing circular convolution using Mersenne Number Transforms. The arithmetic to compute Mersenne transform requires only additions and circular shifts of bits within the word. Circular convolution is computed in a similar fashion as given by equation (2.5). Mersenne Number transforms provide error free convolution, since quantisation and truncation have no meaning in the field of integers. MNTs are defined in a field under addition and multiplication, also the associative, commutative and distributive laws hold, except that division is not defined therefore some numbers do not have multiplicative inverses mod $M_p$, unless $M_p$ is prime.

Mersenne number transforms are defined in a set of p integers.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, 2^{nk} \mod M_p \qquad (2.15)$$

where k = 0,1,2,...,p-1

Let q be defined as inverse of p such that

$$q = M_p - (M_p - 1)/p$$

we have solution

$$(pq) = 1 \mod M_p$$

if $(M_p - 1)/p$ is an integer

but $M_p - 1 = 2^p - 2$

since $p \mid 2^p - 2$.

It is a special case of Fermat's theorem which states that, for every prime p and every integer q, $p \mid q^p - q$, this proves that is an integer. Since

$$pq = (p-1)\, M_p + 1 = 1$$

thus the inverse transform is given by:

$$x(n) = q \sum_{k=0}^{N-1} X(k)\, 2^{-nk} \mod M_p \qquad (2.16)$$

where n = 0,1,2,...,p-1.

To ease the computations $2^p$ (p is prime) may provide a suitable modulus, but the transform length is restricted to 2p. As 2p is not highly composite, it is not of much interest. Consider modulus $2^k + 1$, the maximum transform length is 2 since $3 \mid 2^k + 1$, hence k must be even (k = pq a composite number). The other choice for the modulus is $2^p - 1$, where p is prime, 2 represents root of unity. This allows addition to be performed by simple 1s complement add. Multiplication mod $M_p$ is done by forming 2 p-bit product of two words, and adding p least significant bits (1s complement addition). However, multiplication by $2^k$ mod $M_p$ is quite simple to implement, requiring bit rotation in a p-bit word. The same is true for the inverse transform except that the results must be multiplied by the inverse q.

### 2.6.2 Fermat Number Transforms

If the modulus is chosen to be a Fermat number, then the transform is called a Fermat Number Transform (FNT). Fermat numbers are defined as:

$$M = F_t = 2^b + 1 \qquad (2.17)$$

where $b = 2^t$, $t = 0,1,2,...$

Fermat numbers $F_o$ - $F_4$ are prime and $F_5$ upwards are composite. Then for FNT to exist

$$N \mid O(F_t)$$

$$O(F_t) = 2^b = N_{max}$$

The largest possible transform length in this case is

$$N = 2^m \qquad m < b$$

If $\alpha = 2$ the FNT can be computed efficiently. The FNT of a sequence is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, \alpha^{nk} \mod F_t \qquad (2.18)$$

where $k = 0,1,2,...,N-1$, and inverse is given by:

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\, \alpha^{-nk} \mod F_t \qquad (2.19)$$

where $n = 0,1,2,...,N-1$, and N is a power of 2, and $\alpha$ is the Nth root of unity, i.e. $\alpha^N \equiv 1 \mod F_t$. In case of the FNT the multiplication is equivalent to bit shifts and add.

One of the constraints in the practical implementation of the FNT is that the wordlength is defined by the transform length (13). For a general $F_t$ ($t>4$) the maximum transform length is

given by $N = 2^{t+2}$. Since $\alpha^2 \equiv 2 \bmod F_t$, $\alpha \equiv \sqrt{2}$, the transform length N = 4 x wordlength. For example arithmetic mod $F_2$ provides us with $6^2 \equiv 2 \bmod 17$, $6 \equiv \sqrt{2} \bmod 17$.

Equation (2.18) can be computed efficiently using FFT type algorithm. In FNT multiplication is equivalent to simple binary word shift followed by subtraction. Leibowitz (14), have used slightly different approach for performing modular arithmetic mod $F_t$. In the Agarwal and Burrus (13), method problems arise due to quantisation when b-bits are used for modular arithmetic. This is due to the fact that $2^b \equiv -1$, hence when -1 is encountered it is either rounded to 0 or 2. This introduces some quantisation error. The method described by Leibowitz (14), uses (b+1)-bits, the extra bit is only used to represent 0.

McClellan (15), have described hardware to implement the FNT. A different number representation is used in which the bits are weighted +1, -1 and not as 0, 1 as in conventional binary representation.

# CHAPTER 3

## Multiplication Techniques for Microprocessors

### 3.1 Introduction

We have seen in the previous chapter that the Number Theoretic Transforms (NTTs) are defined in a finite ring of integers $Z_M$. NTTs provide error free convolution (9), (12), (13). Since in the ring all the numbers are defined precisely, so there is no ambiguity in their representation on a digital computer. In contrast floating point numbers cannot be represented accurately on a digital computer, and floating point arithmetic is subject to roundoff and truncation errors.

Ordinary microprocessors are integer processing machines and are available at much lower prices than the floating point arithmetic processors. A microprocessor provides cheap integer processing power. By appropriately manipulating the carry bit in the condition code register, the microprocessor is capable of performing multi-precision arithmetic, for example an 8-bit microprocessor can perform 16-bit arithmetic operations. It seems logical to investigate the possibilities for implementing NTTs on microprocessors (5), (6). In many microprocessors no hardware multiplier is available since it requires more hardware and chip area. When a hardware multiplier is not available alternative methods may be employed to perform the multiplication in software or by implementing an external hardware multiplier

(18), (31), (41).

For real time digital signal processing applications, multiplication must be carried out efficiently. The multiplication speed can be increased by reducing the total number of additions (of partial products) or by performing high speed addition. Carry Save Adders (CSA) or Carry Look Ahead (CLA) may be used to reduce the carry propagation delay instead of conventional Carry Propagate Adders (CPA) (16), (17), (23).

## 3.2 Clocked Multiplication Algorithms

We can classify multiplication in different ways i.e. serial, parallel, unsigned, signed (twos complement). A brief outline of different algorithms for binary multiplication is presented.

### 3.2.1 Multiplication on a Microprocessor

The simplest form of binary multiplication is multiplication by two or powers of two. This is analogous to multiplication by ten or powers of ten (considering integer arithmetic) in the decimal number system. Multiplication by ten is accomplished by appending a number of zeros equal to the power of ten towards the least significant digit. Similarly in the binary number system, multiplication by two is accomplished by shifting the binary word towards the most significant bit position and filling the vacated places by zeros. The number of shifts is equal to the power of two. Overflow conditions must be detected and dealt with accordingly. It may be mentioned here that division by two in the

binary number system is equivalent to shifting the binary word a number of positions towards the low order significant bits. This is analogous to shifting of the decimal point in the decimal number system towards the high order digit position. However, in the binary number system if the least significant bit was a one prior to division by two, then the result is subject to truncation. This may be circumvented by rounding the binary word prior to shifting, this is done by adding a one to the least significant bit irrespective of the bit value.

In practice it is quite uncommon to encounter multiplications by two or a power of two. Hence some other method must be devised and developed for the implementation of multiplication on a microprocessor.

The most commonly used method to perform multiplication on the microprocessor is the shift and add algorithm. The microprocessor checks the bits in the multiplier one by one and if a one is encountered the multiplicand is added to the partial product. After addition the partial product is shifted towards the least significant bits. If a zero is encountered then no addition takes place and the partial product is simply shifted towards low order bits, which is equivalent to shifting of multiplicand towards the most significant bit position (28). This method is lengthy and quite inefficient for large numbers. If subtract instruction is available then an alternative method may be used. For example a string of ones in the multiplier can be reduced to subtract for the first 1 encountered, shift for each subsequent 1 and addition for the first 0 encountered. A

multiplication by 14 (1110) may be reduced as follows.

$$14 = 2^3 + 2^2 + 2^1$$
$$= 2^4 - 2^1$$
$$= 10000 - 10$$

Since the multiplication time increases with the number of multiplier bits, the above mentioned method may produce results faster than the shift and add algorithm. This algorithm may also be implemented externally in hardware (17), (18).

### 3.2.2 Burk-Goldstine - Von-Neumann Method

This method was developed for twos complement multiplication (21). In this method if the multiplier and the multiplicand are positive no correction of the final result is required. However, if any of the operands is negative (twos complement) then correction must be applied to the final result. This step is necessary since in the twos complement number the sign is embedded in the number itself. This algorithm generates the product in the following manner.

Let X, Y be the multiplicand and the multiplier respectively, where

$$X = -x^0 + X^*$$
$$Y = -y^0 + Y^* \qquad (3.1)$$

$-x^0$ and $-y^0$ represent the sign bit and $X^*$ and $Y^*$ give true value of the numbers. For number representation see Chu (21). The product is obtained as follows

$$X^* Y^* = (X + x^0)(Y + y^0)$$
$$= XY + x^0Y + y^0X + x^0y^0$$

To obtain the correct answer $-(x^0Y + y^0X + x^0y^0)$ must be added to the final product, such that

$$X^* Y^* = X Y$$

If one of the numbers is positive then either $-x^0Y$ or $-y^0X$ have to be added.

### 3.2.3 Robertson's First Method

This method multiplies a signed number X with an unsigned number $Y^* = Y$. When the multiplier is negative, correction term $-y^0X$ must be added. No correction is required when the multiplicand is negative (21).

### 3.2.4 Robertson's Second Method

In this method if the multiplier is negative, then the product of -X and -Y is calculated which yields a positive result, then no correction is required. But if Y = -1 then the result is not correct. The value of Y must be restricted such that $-1 < Y < 1$ (21).

Comparing the two methods, in the first method if the multiplier is negative then it needs correction, but in the second method no correction is required. The hardware only needs to sense the sign bit $y^0$ of the multiplier and to complement the multiplicand X.

### 3.2.5 Booth 's Algorithm

Booth's algorithm is quite extensively used where serial, signed twos complement multiplication has to be implemented (20), (21), (28), (35), (40), (43), (46). This method has an advantage over the previous methods that no prior knowledge of the sign and no correction of the result is required at the end. Also the product is independent of the sign of the multiplier and the multiplicand. Let the multiplier and multiplicand be represented as.

$$X = -x_n 2^n + x_{n-1} 2^{n-1} \ldots\ldots + x_0 2^0$$

$$Y = -y_n 2^n + y_{n-1} 2^{n-1} \ldots\ldots + y_0 2^0$$

In this method two consecutive bits $y_i$ and $y_{i-1}$ of the multiplier are examined simultaneously, starting from the least significant bit. Three possible conditions can arise for $y_i$ and $y_{i-1}$

i) if $y_i$, $y_{i-1}$ are 01, then the multiplicand is added to the partial product. After addition the partial product is shifted by one bit towards the least significant bit position.

ii) if $y_i$, $y_{i-1}$ are 10, then the multiplicand is subtracted from the partial product and the partial product is shifted one bit towards the least significant bit position.

iii) if $y_i$, $y_{i-1}$ are 00 or 11, then no addition or subtraction takes place. However, the partial product is shifted one bit position towards the least significant bit.

### 3.2.6 A Short Cut Multiplication Method

This method involves detection of isolated bits ones or zeroes. If a sequence of ones are detected then multiple addition of the multiplicand into the partial product takes place. Otherwise multiple shifts are performed on the partial product. Additional hardware may be required to detect the sequence of ones or zeroes. For example, if the multiplier is 01000100, then there are only two additions of $2^6$ and $2^2$. Worst case would be if the multiplier had alternating ones and zeroes.

### 3.2.7 Multiple Digit Multiplication Method

This algorithm uses the method of repeated additions of the multiplicand to the partial product. However, there is a subtle difference from the method described previously (Booth's algorithm). In this method two consecutive bits of the multiplier are checked simultaneously. The following four different conditions can arise.

i) if $y_i$, $y_{i-1}$ are 00, then no addition takes place

ii) if $y_i$, $y_{i-1}$ are 01, then the multiplicand is added into the partial product.

iii) if $y_i$, $y_{i-1}$ are 10, then twice the multiplicand is added into the partial product.

iv) if $y_i$, $y_{i-1}$ are 11, then three times the multiplicand is added into the partial product.

Since two consecutive bits are considered only once, the total number of addition steps are thus reduced and hence there is an overall improvement in the speed. It may be noted that the

partial product is shifted two bit positions instead of one after the addition of the multiplicand into the partial product.

Parasuraman (18), have described a variation in this method by inspecting three bits at a time and applying correction. Harman (19), have described a possible method to increase the multiplication speed by examining the number of ones in the multiplier and the multiplicand. The operand which has the least number of ones is chosen as the multiplier. This method may not find a place in practical applications.

## 3.3 Clockless Multiplication

All the different techniques described above use clock signals to generate the shift and the add pulses. Now we consider some algorithms for clockless multiplication which are much faster than the methods described before. Clockless circuits are also referred to as combinatorial circuits, whose outputs entirely depend upon the current input values.

### 3.3.1 Array or Parallel Multiplication

This method is generally used when high speed multiplication is to be performed. All the bits of the multiplier and multiplicand are fed simultaneously into an array of logic gates and full adders. No storage of partial or intermediate products is required. Chu (21), have described a simultaneous multiplier in which the two operands are fed into a two dimensional array structure of logic gates and full adders.

Rabiner and Gold (20), have also discussed a fast parallel multiplier which consists of a two dimensional array of 1-bit adders. The total multiplication time is the sum of the settling time and the propagation delay of the logic used, after the operands are fed into the input. The unit cell is shown in figure (3.1a). These basic cells are cascaded to give a parallel multiplier structure. Figure (3.1b) shows a 3 x 3-bit parallel array multiplier. This arrangement can be extended to an n x n-bit parallel multiplier. A finite amount of time is required for the carry to propagate through different stages of the multiplier. The partial products can be generated as shown in figure (3.2). A problem arises when the partial products have to be added. For small numbers the conventional ripple carry adder (CPA) may be used to add the partial products, but for larger numbers a CLA (Carry Look Ahead) or a CSA (Carry Save Adder) may be used (22), (23). Davies and Fung (31), and Bate and Burkowski (33), have described the interfacing of a high speed combinational array multiplier to a microprocessor.

### 3.4 Read Only Memory (ROM) Multiplier

With the availability of cheap and fast ROMs for storing information lookup techniques may be employed to perform arithmetic operations for a small range of numbers (18), (26), (27), (28). The ROM is programmed such that the products are stored in it in an appropriate manner. The address lines are used as input, and the product is obtained on the data bus. This method is very fast since the output from the ROM entirely depends upon the access time of the ROM and may be of the order

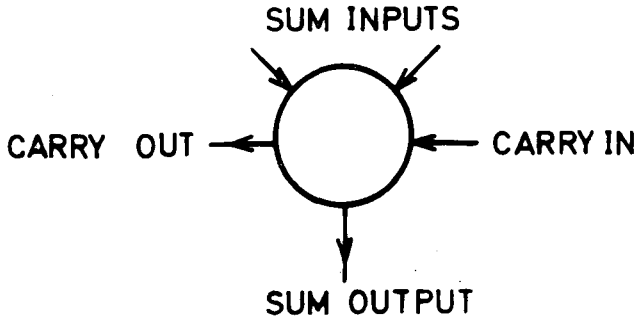



Figure 3.1a: Unit cell (1-bit adder).

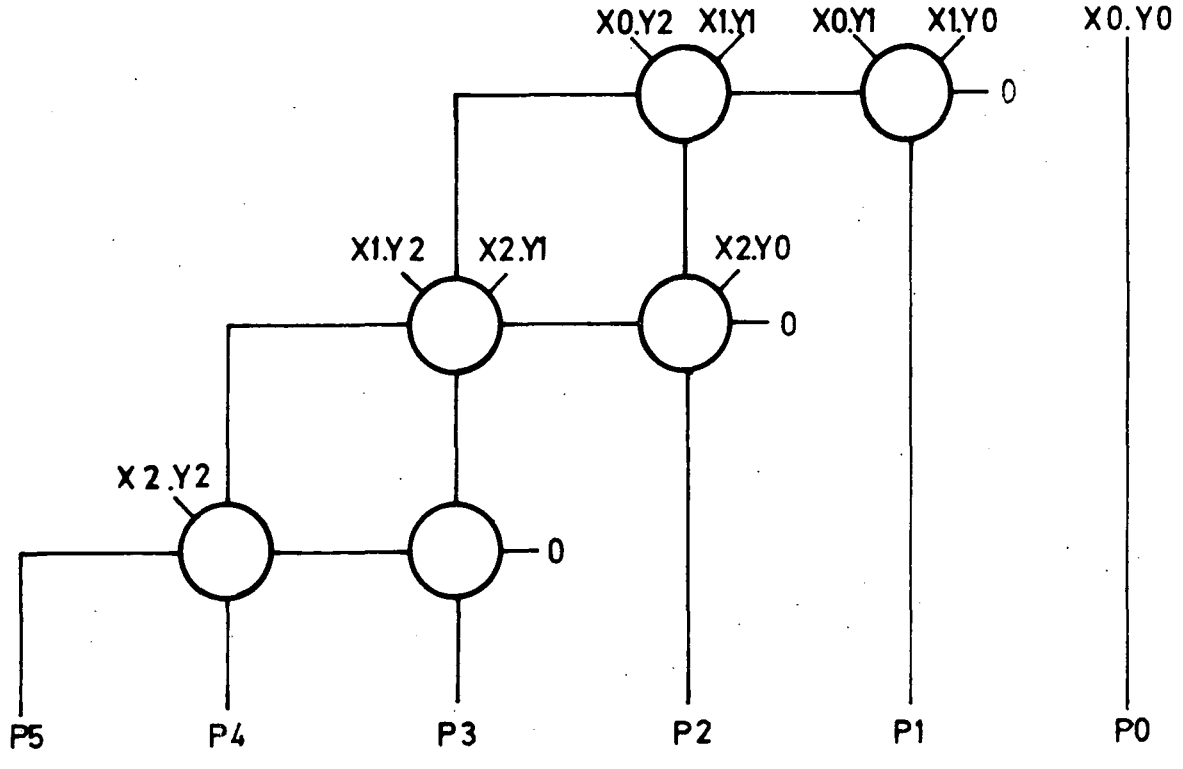



Figure 3.1b: 3 x 3 Parallel array multiplier by combining unit cells.

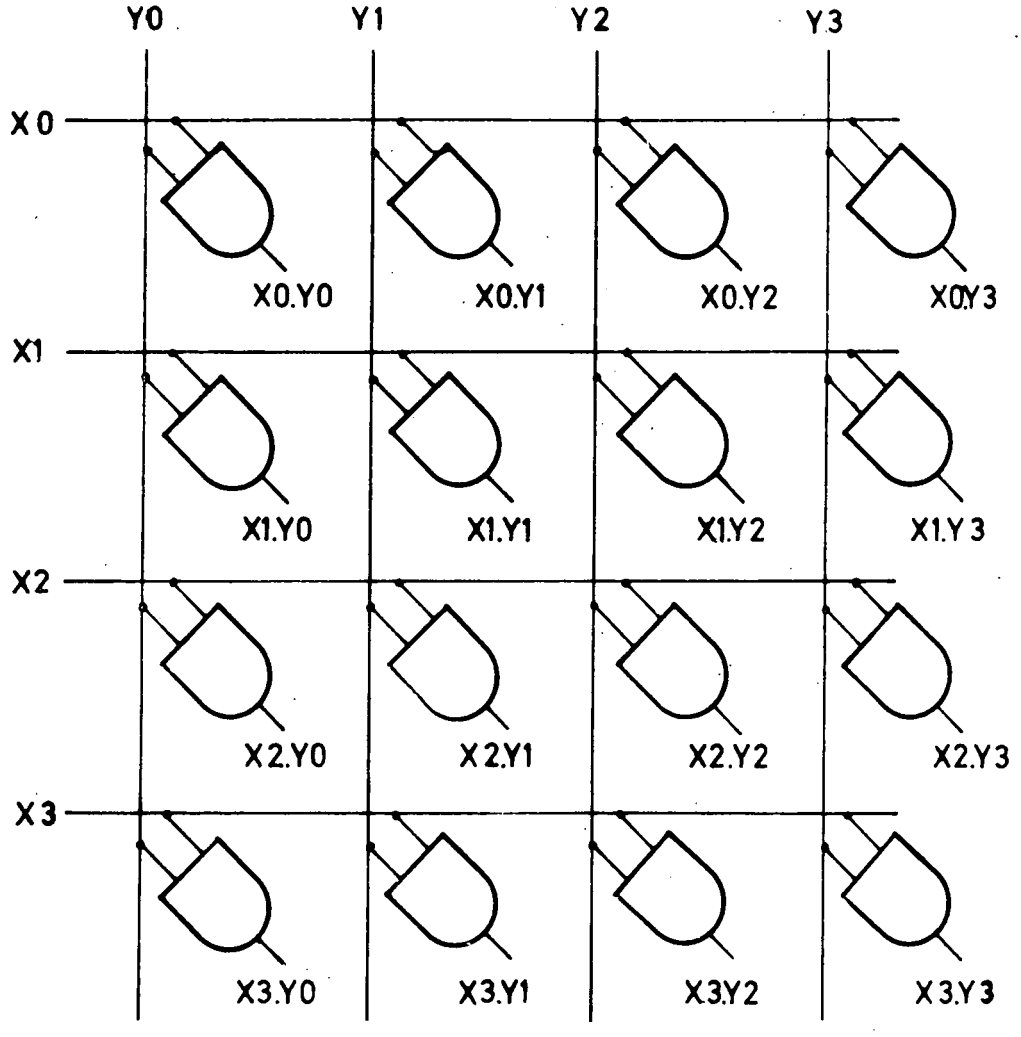



Figure 3.2: Arrangement for generating partial products.

of tens of nanoseconds. The ROM lookup technique for multiplication can be used in variety of ways some of which are described below.

### 3.4.1 Direct ROM Multiplier

The multiplier and multiplicand are appropriately connected to the address bus of the ROM. The product of the two numbers, which is stored at this address is then obtained directly. Figure (3.3) shows an arrangement for a simple ROM multiplier. The disadvantage is that if the numbers are large then this method may become impractical due to complexity, size and cost.

### 3.4.2 Quarter-Squares Lookup Table Multiplication

Let X and Y be the two n-bit numbers to be multiplied. Then the product is obtained in the following manner.

$$XY = \frac{(X + Y)^2 - (X - Y)^2}{4} \qquad (3.1)$$

$$XY = \left[\frac{X + Y}{2}\right]^2 - \left[\frac{X - Y}{2}\right]^2 \qquad (3.2)$$

$$XY = \frac{(X + Y)^2}{4} - \frac{(X - Y)^2}{4} \qquad (3.3)$$

Squares of the sum and difference of the two numbers are stored in separate ROMs. Sum and difference is obtained by conventional method using adder. Figure (3.4) shows an arrangement for such a multiplier.

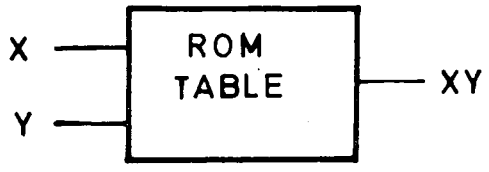



Figure 3.3: Direct ROM multiplier.

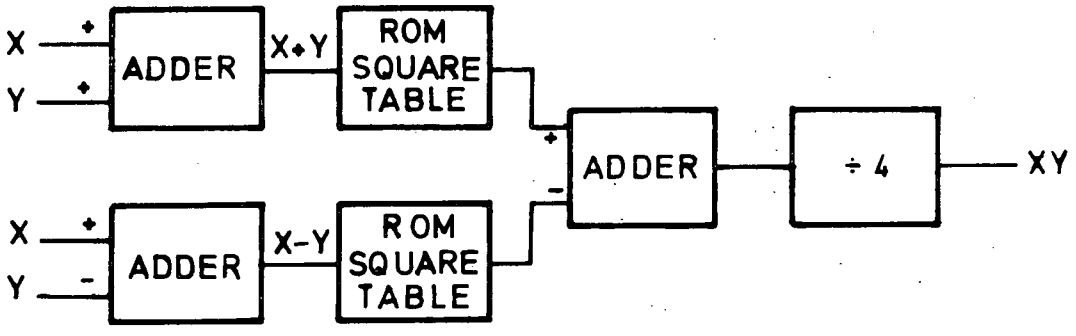



Figure 3.4: Quarter-squares lookup table multiplication.

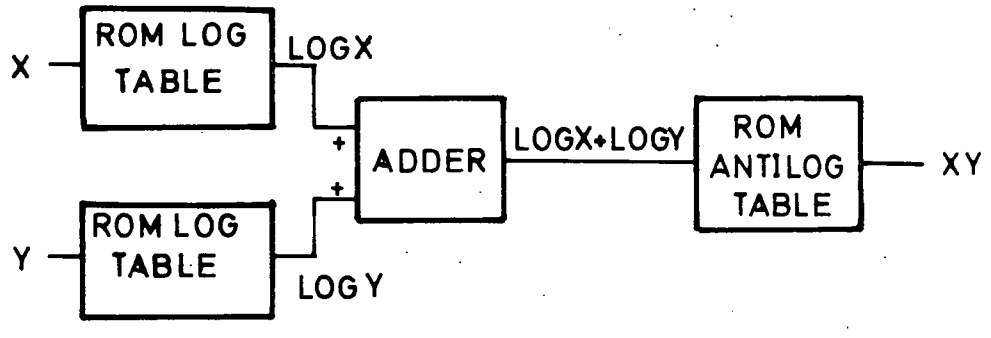



Figure 3.5: Multiplication using logarithms.

In equation (3.1) the product is obtained by dividing the difference of the output of ROM squarer by 4. In equation (3.2) the division by 2 is accomplished before feeding the sum and difference to the ROM square table. This sometimes introduces truncation errors. Equation (3.3) is equivalent to equation (3.1) and gives the same results (26).

For X and Y even or odd we have X = 2m and Y = 2n or X = 2m + 1 and Y = 2n + 1 respectively. If X and Y are even or odd equations (3.1) and (3.3) are equivalent, but equation (3.2) produces truncation errors.

For example, if X is even and Y is odd, then X=2m, Y=2n+1, substituting these in equation (3.3) we get:

$$2m(2n+1) = \frac{(2m + 2n + 1)^2}{4} - \frac{(2m - 2n - 1)^2}{4}$$

$$= (m+n)^2 + (m+n) + \tfrac{1}{4} - (m-n)^2 - (m-n) - \tfrac{1}{4}$$

$$= (m+n)^2 + (m+n) - (m-n)^2 - (m-n)$$

$$= 4mn + 2m \qquad (3.4)$$

Considering the case with equation (3.2), we get:

$$\left[\frac{2m+2n+1}{2}\right]^2 - \left[\frac{2m-2n-1}{2}\right]^2 = (m+n)^2 - (m-n)^2$$

$$= 4mn$$

$$\neq XY \qquad (3.5)$$

Equation (3.5) shows truncation error of 2m. Davies (28), have described implementation of this method directly on the Z80 microprocessor in software.

Johnson (27), have described an improved ROM lookup method. Partial products are stored in separate ROMS and the lookup results are added appropriately. Product time depends upon the access time of the ROMs and the carry propagation delay of the adders. Parasuraman (18), have also described lookup method for multiplication.

### 3.4.3 Multiplication Using Logarithms

Brubaker and Becker (25), have described another approach to binary multiplication. This method employs logarithm and antilogarithm tables stored in ROMs. The product of two numbers are obtained in the following manner.

$$XY = \text{antilog}(\log X + \log Y)$$

This method introduces errors due to truncation and rounding. A disadvantage in this method is that only the product of positive numbers can be directly obtained (since the logarithm of a negative number is undefined). However, the sign of the product can be generated externally if required. Figure (3.5) shows an arrangement for the logarithmic multiplier. The multiplication time is twice the access time of the ROM.

## 3.5 Parallel Multipliers Chips

Parallel multiplication can be achieved using discrete components described. However, VLSI technology now allows the integration of a complete n x n-bit multiplier on a single chip. These chips are easy to interface with a general purpose microprocessor (18), (31), (34), (35), (36), (37), (38), (39),

(41), (42), (44). Usually these multiplier chips can be cascaded so as to allow multiplication of arbitrary length numbers.

The methods discussed previously use twos complement multiplication with discrete components. However, in VLSI chips a facility may be provided to perform signed or unsigned multiplication, rounding etc.

Bywater (16), Lewin (17), Rabiner and Gold (20), Chu (21), Hayes (22), Flores (45), Booth and Booth (46), Abd-alla and Meltzer (47), are also suggested for further reading.

## 3.6 Modular Arithmetic on Microprocessor

Modular arithmetic operations can be implemented on any microprocessor with unsigned compare instructions. Some microprocessors may perform these arithmetic operations more efficiently and faster than the others. This depends upon the clock frequency, number of accesses to the memory to fetch the operands and the number of CPU registers available. If the CPU has enough registers to hold the operands and the intermediate or partial products, then the total number of memory accesses are reduced (during the multiplication), which will produce faster results.

Modular arithmetic routines were written for several microprocessors. Results of the routines are shown in tables (3.1) to (3.3). Appendix-A contains assembler source listings of these modular arithmetic routines. Note that each of the microprocessor has a different clock frequency. Renold (48),

Table 3.1: Results of benchmark programs for modular addition.

| Clock MHz | Microprocessor (No. of bits) | Number of Program Bytes | Number of Instr Executed | Clock Cycles (Time μsec) | Price |
|---|---|---|---|---|---|
| 3 | TMS9900 (16) Texas Instr Ltd | 36 | 8 | 88 (29.3) | 50.0 |
| 2 | M6502 (8) MOS Technology | 42 | 22 | 74 (37.0) | 13.0 |
| 1 | M6809 (8) Motorola | 20 | 8 | 40 (40.0) | 13.0 |
| 8 | 8X300 (8) Signetics | 76 | 26 | 52 (6.50) | 36.0 |
| 4 | Z80 (8) Zilog | 36 | 14 | 75 (18.74) | 11.0 |
| .125 | COP402 (4) National Semiconductors | 205 | 108 | 216 (864.0) | 4.80 |

Table 3.2: Results of benchmark programs for modular subtraction.

| Clock MHz | Microprocessor (No. of bits) | Number of Program Bytes | Number of Instr. Executed | Clock Cycles (Time μsec) | Price |
|---|---|---|---|---|---|
| 3 | TMS9900 (16) Texas Instr. Ltd | 24 | 8 | 88 (29.3) | 50.0 |
| 2 | M6502 (8) MOS Technology | 75 | 16 | 59 (29.5) | 13.0 |
| 1 | M6809 (8) Motorola | 14 | 6 | 32 (32.0) | 13.0 |
| 8 | 8X300 (8) Signetics | 108 | 50 | 100 (12.5) | 36.0 |
| 4 | Z80 (8) Zilog | 49 | 22 | 117 (29.24) | 11.0 |
| .125 | COP402 (4) National Semiconductors | 211 | 134 | 268 (1072.0) | 4.80 |

Table 3.3: Results of benchmark programs for modular multiplication.

| Clock MHz | Microprocessor (No. of bits) | Number of Program Bytes | Number of Instr. Executed | Clock Cycles (Time Usec) | Price |
|---|---|---|---|---|---|
| 3 | TMS9900 (16) Texas Instr. Ltd | 18 | 5 | 242 (80.0) | 50.0 |
| 2 | M6502 (8) MOS Technology | 333 | 1246 | 4866(2433.0) | 13.0 |
| 1 | M6809 (8) Motorola | 128 | 60 | 336 (336.0) | 13.0 |
| 8 | 8X300 (8) Signetics | 160 | 325 | 650 (81.25) | 36.0 |
| 4 | Z80 (8) Zilog | 252 | 108 | 2462 (615.5) | 11.0 |
| .125 | COP402 (4) National Semiconductors | 859 | 2269 | 4553 (18212.0) | 4.80 |

have compared performances of five different microprocessors by means of nine different benchmark programs. He has suggested two methods for comparison.

i) An instruction of medium complexity (load 8-bit register) is chosen as an instruction unit. The number of clock cycles for any instruction is divided by the number of clock cycles of the instruction unit.

ii) Reduce the clock frequency such that the instruction unit takes the same time for all the processors.

Smith (58), have also described comparison of three microprocessors by executing a standard program on each one of them. The performance is compared by looking at the number of program bytes required, execution time etc.

To implement modular arithmetic any value of modulus M may be chosen. The residue is usually computed using division, but division, like multiplication is not an efficient operation when implemented on a microprocessor. Division may also be implemented externally which may require complex hardware. Special techniques may be used to compute the residue.

In a decimal number system, if the modulus is chosen to be 10, then the residue of the number is the least significant digit of the number. For example $103 \equiv 3 \bmod 10$. A similar case is also true in binary number system. If the modulus is chosen to be $2^k$ (k is a positive integer) then the residue is found by masking out the most significant k-bits except the low order k-bits which is the residue. A carry into the kth bit is

congruent to 1 and if added to the least significant k-bits gives the residue. A choice of modulus $2^k$-1 also provides easy calculation of the residue. The residue in this case is computed by adding the k most significant bits to the k least significant bits. But in some cases if the k least significant bits are 1s, and the k most significant bits are zeros, then the result is not correct and may be corrected by adding a one to the k least significant bits.

Let k = 4, $2^4 - 1 = 15$

i) $7 \times 8 = 56 \equiv 11 \bmod 15$

in binary form it is given as

$$0111 \times 1000 = 0011\ 1000$$

```
                1000
              + 0011
                -----
carry = 0       1011        mod F
```

ii) $14 \times 14 = 196 \equiv 1 \bmod 15$

$$1110 \times 1110 = 1100\ 0100$$

```
                0100
              + 1100
                -----
carry = 1       0000
              + 0001
                -----
                0001        mod F
```

If the modulus is chosen as $2^k + 1$ then $2^k = -1$ and $2^{nk} = (-1)^{nk}$. The problem in this case (k-bit arithmetic) is the representation of -1 if it is encountered, it is either rounded to 0 or 2. To implement NTT there are several constraints between the modulus and the wordlength. If the wordlength of the microprocessor does not allow the required dynamic range of numbers, the Chinese Remainder Theorem (CRT) may

be used to perform arithmetic modulo product of several moduli.

A search for a suitable modulus made by Martin (5), showed that a value of M=65521 ($2^{16}$-15) is very convenient for implementation of the NTTs using the WFTA. This is the first prime number below $2^{16}$ and allows a dynamic signal processing range of nearly $2^{16}$. Some examples of arithmetic modulo 65521 ($FFF1) are given below. $ shows a hexadecimal number. All the following examples use hexadecimal numbers, $ is omitted. NTTs deal with unsigned numbers so more emphasis will be given to this type of arithmetic.

### 3.6.1 Addition Modulo 65521

When two 16-bit numbers are fed into a binary adder, a value of $2^{16}$ - 65521 (=15) must be added to the sum,

i) if a carry was generated or

ii) if the sum was greater than 65521.

However, this may generate a further carry, but not more than two carries can ever be generated.

```
i)              0279
              + 041C
                ------
carry = 0       0695        mod FFF1


ii)             FFEF
              + 0014
                ------
carry = 1       0003        mod FFF1
              + 000F
                ------
carry = 0       0012        mod FFF1
```

### 3.6.2 Subtraction Modulo 65521

Subtraction is performed in the usual way by adding the twos complement of the subtrahend to the minuend. A value of 65521 must be added to the result, if the subtrahend was greater than minuend.

i)

```
  0352
- 0140
------
  0212        mod FFF1
```

ii)

```
  0140
- 0352
------
  FDEE
+ FFF1
------
  FDDF        mod FFF1
```

### 3.6.3 Multiplication Modulo 65521

If the product of two 16-bit numbers exceeds 65521 then the product is reduced modulo 65521.

$$0003 * 0003 \equiv 0009 \quad \text{mod FFF1}$$

$$FFF0 * FFF0 \equiv 0001 \quad \text{mod FFF1}$$

$$(FFF0 \equiv -1 \text{ mod FFF1})$$

# CHAPTER 4

## Implementation of the Winograd Fourier Transform Algorithm

### 4.1 Introduction

The Discrete Fourier Transform (DFT) of a sequence x(n) is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W^{nk} \qquad (4.1)$$

and the inverse is given by:

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\, W^{-nk} \qquad (4.2)$$

where $W = e^{-j2\pi/N}$, W is an integer root of unity such that $W^N \equiv 1$, N is the sequence length. Cooley and Tukey (8), showed an efficient way of computing the DFT which reduces the number of operations from $N^2$ to $N\log_2 N$. Attempts have been made to further reduce the number of operations. Winograd (3), proposed a new class of Winograd Fourier Transform Algorithms (WFTA), which requires only 20 percent of multiplications as that of Cooley-Tukey's FFT algorithm and roughly the same number of additions. Winograd proposed short length DFT algorithms of length 2, 3, 4, 5, 7, 8, 9, 16, with minimum number of multiplies. Table (4.1) shows number of additions and number of multiplications for each of these short length DFT algorithms.

| Short-length WFTA | No. of Adds | No. of Multiplies |
|---|---|---|
| 2 | 2 | 2 |
| 3 | 6 | 3 |
| 4 | 8 | 4 |
| 5 | 17 | 6 |
| 7 | 36 | 9 |
| 8 | 26 | 8 |
| 9 | 44 | 13 |
| 16 | 74 | 18 |

Table 4.1: Number of additions and multiplications in the Winograd short length DFT algorithms.

In the FFT the sequence length is $N = 2^m$, where m is a positive integer. However, in the WFTA the transform length is equal to several mutually prime factors. If not more than one factor is chosen from each of the following groups (2, 4, 8, 16), (3, 9), (7) and (5), transform lengths in the range from 2 to 5040 are possible. This is done by nesting the short length algorithms together in the following manner. Each of the short length DFT algorithms consists of input additions followed by multiplications and the output additions. In the nested form all the input additions (for the mutually prime factors) are performed one after the other followed by multiplications (with the coefficients) and the output additions. Instead of performing the multiplications separately for each of the short length factors, the multiplications are also nested (49). This algorithm reduces the total number of multiplications at the cost of increased algorithm complexity. These multiplications are performed with precomputed transform coefficients. There are two sets of transform coefficients, one for the forward transform and the other set for the inverse transform. $N^{-1}$ in equation (4.2) is combined with the inverse transform coefficients so that the forward and the inverse WFTA can be computed with equal computational effort.

For example, for sequence length N = 15 the mutually prime factors are (3,5) = 1. Figures (4.1) and (4.2) show the 3-point and 5-point WFTA respectively. Let x0,x1,... denote the input sequence and X0,X1,... denote the transformed sequence.

3-Point WFTA

$N = 3,\ U = 2\pi/3$

$t1 = x1 + x2$

$m0 = 1 \cdot (x0 + t1)$

$m1 = (COSU - 1).t1$

$m2 = jSINU(x1 - x2)$

$s1 = m0 + m1$

$X0 = m0$

$X1 = s1 + m2$

$X2 = s1 - m2$

5-Point WFTA

$N = 5,\ U = 2\pi/5$

$t1 = x1 + x4,\ t2 = x2 + x3,\ t3 = x1 - x4$

$t4 = x3 - x2,\ t5 = t1 + t2$

$m0 = 1 \cdot (x0 + t5)$

$m1 = (\frac{1}{2}(COSU + COS2U) - 1).t5$

$m2 = \frac{1}{2}(COSU - COS2U).(t1 - t2)$

$m3 = jSINU.(t3 + t4)$

$m4 = j(SINU - SIN2U).t4$

$m5 = j(SIN2U + SINU).t3$

$s1 = m0 + m1,\ s2 = s1 + m2,\ s3 = m5 - m3$

$s4 = s1 - m2,\ s5 = m3 + m4$

$X0 = m0$

$X1 = s2 + s3$

$X2 = s4 + s5$

$X3 = s4 - s5$

$X4 = s2 - s3$

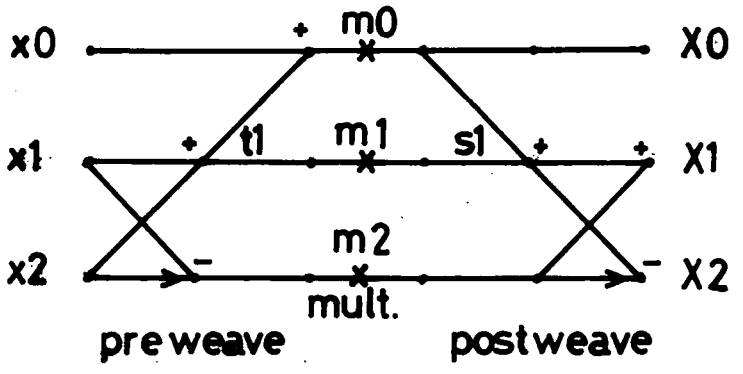



Fig. 4-1: 3-Point WFTA

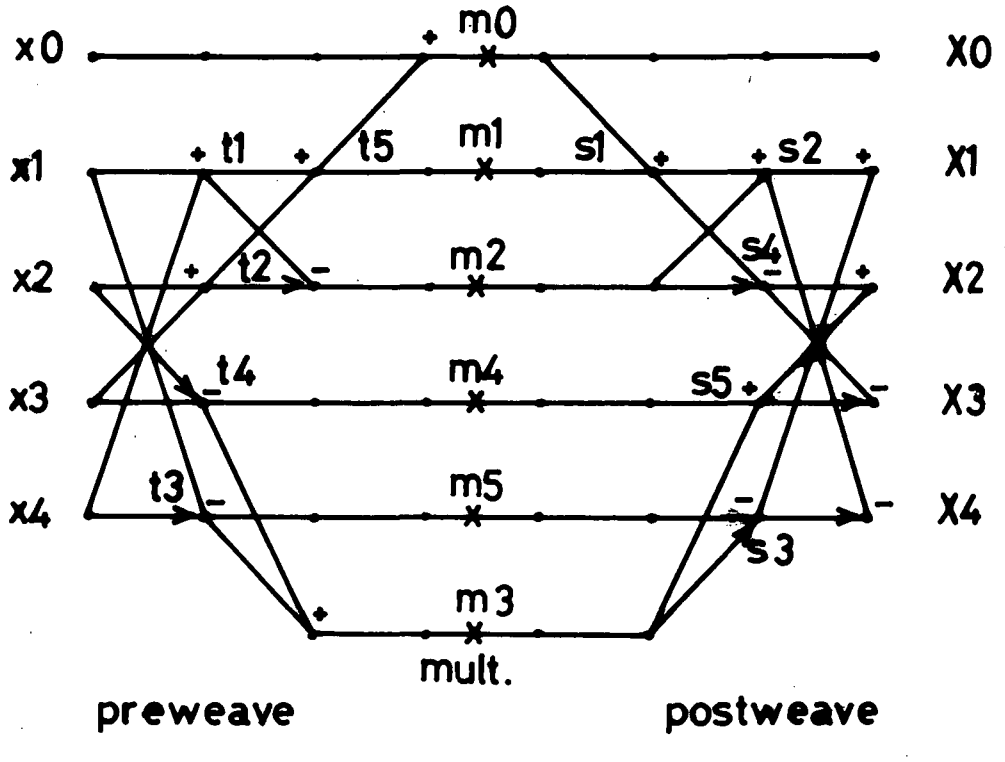



Fig. 4-2: 5-Point WFTA

The nested 15-point WFTA is shown in figure (4.3) which clearly shows five 3-point pre-weaves (premultiply adds), followed by three 5-point pre-weaves. This is followed by the multiplications, the number of multiplications is equal to the product of the multiplications in individual short length DFT algorithms. Finally the three 5-point post-weaves (postmultiply adds) and the five 3-point post-weaves are performed. WEAVE (50) is an acronym for Winograd Elementary Add Vector Elements. Note that there are eighteen multiplications in the 15-point WFTA, since there are three multiplications in the 3-point WFTA and six multiplications in the 5-point WFTA.

Similarly a 60-point WFTA has three mutually prime factors 3, 4, and 5. First of all twenty 3-point, fifteen 4-point, twelve 5-point pre-weaves are performed, followed by 72 modular multiplications with coefficients and the post-weaves for each of the short length WFTA.

The input and the output data must be reordered or shuffled. The input and output shuffle vectors are also precomputed and stored in the memory and the shuffle is then performed using lookup. The disadvantage in the WFTA is that extra memory is required just to store the input/output shuffle vectors and the forward and the inverse WFTA coefficients. However, this algorithm is computationally efficient on machines on which the multiplication time is much longer than the addition time.

Silverman (51), have described memory considerations for the FFT and WFTA, and discussed that the WFTA requires 7N memory

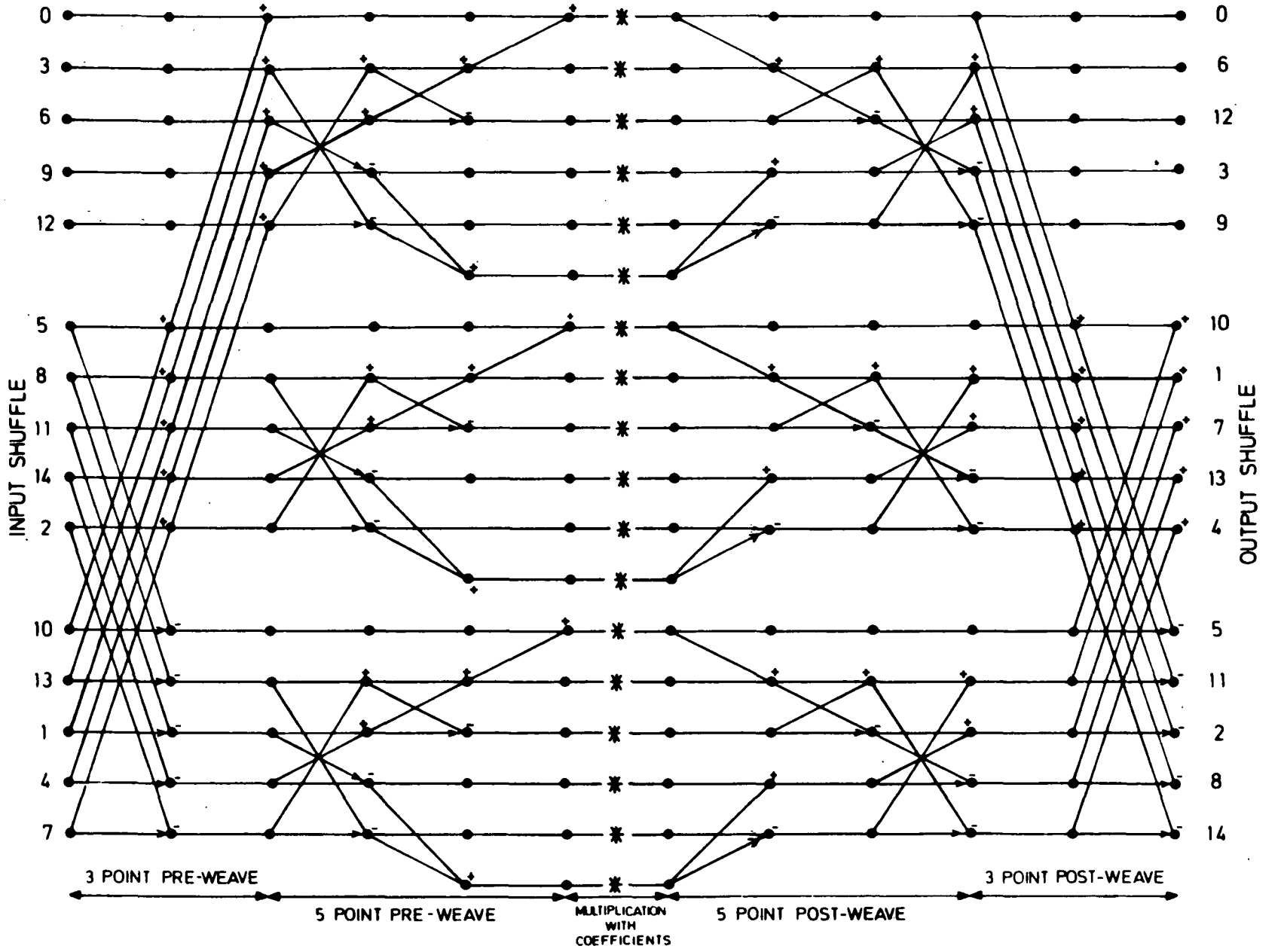



FIG. 4·3: NESTED 15-POINT WINOGRAD FOURIER TRANSFORM ALGORITHM (WFTA)

locations as compared to comparable size FFT algorithm which requires 1.25N memory locations. Unlike the FFT, the WFTA cannot be computed inplace, Silverman called an analogous approach as full overlay. Nawab and McClellan (52), have described that in general the WFTA requires more data transfers than an equivalent length FFT. In addition they have also discussed the minimum number of CPU registers required to perform each short length DFT algorithm efficiently, since register to register instructions are executed much faster.

## 4.2 Computation of NTT using WFTA

The Number Theoretic Transform (NTT) of a sequence x(n) is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)\,\alpha^{nk} \qquad (4.3)$$

and the inverse is given by:

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\,\alpha^{-nk} \qquad (4.4)$$

where $\alpha = e^{-j2\pi/N}$, and is an integer root of unity, such that $\alpha^N \equiv 1 \bmod M$, where M is the modulus, and $\alpha$ is defined in a finite ring of integers $Z_M$. The choice of modulus is made such that $N \mid M$, if M is prime then $N \mid M-1$. The inverse $N^{-1}$ is defined such that $NN^{-1} \equiv 1 \bmod M$. If M is not a prime then $N^{-1}$ may or may not exist. Martin (5), carried out a search for a suitable modulus on the lines described by Bailey

(53), and found that value of M = 65521 is quite adequate for 16-bit modular arithmetic and it is the first prime below $2^{16}$. Since NTTs are similar in structure to the DFT any algorithm which applies to the DFT can also be applied to the NTT.

### 4.2.1 Determination of the Constants for the WFTA

Implementation of the WFTA requires some constants to be precomputed and stored in the memory. These are the input/output shuffle vectors, transform coefficients etc. Consider that we want to implement a 15-point WFTA. The following calculations must be performed before the actual program coding.

1- Choice of modulus M = 65521, since it satisfies the condition $N \mid O(M)$, where O(M) is the g.c.d of $(p_i-1)$. O(65521) = 13 x 5040 and so this modulus will support any Winograd transform algorithm (5), (9).

2- Choice of transform length N = 15.

3- Determination of $N^{-1}$, $15^{-1} \equiv 61153 \bmod 65521$.

4- Determination of element of order N,

$\alpha^{15} \equiv 1 \bmod 65521$, $(7791)^{15} \equiv 1 \bmod 65521$.

5- Determination of mutually prime factors 15 = 3 x 5, such that (3,5) = 1.

6- Determination of j (iota) such that $j.j \equiv -1 \bmod 65521$, $j \equiv 41224 \bmod 65521$, j is an element of order 4, such that $(41224)^4 \equiv 1 \bmod 65521$.

7- Determination of $2^{-1}$, $2^{-1} \equiv 32761 \bmod 65521$

8- Determination of the input and output shuffle or reordering vectors. The input and output shuffle

vectors are obtained using Chinese Remainder Theorem (CRT), in the following manner.

Let $N = r_1 r_2$ such that $(r_1 , r_2 ) = 1$

also let $q_1 = 0,1,\ldots,r_1-1$, and $q_2 = 0,1,\ldots,r_2-1$

The following equation allows mapping from a one dimensional into a two dimensional array.

$$(r_2 q_1 + r_1 q_2) \bmod N$$

Let

$$r_1=3, \quad r_2=5, \quad q_1=0,1,2, \quad q_2=0,1,2,3,4$$

We get

$$(5q_1 + 3q_2) \bmod 15 \qquad (4.5)$$

Using equation (4.5) we obtain the following input shuffle vectors

| | | | | |
|---|---|---|---|---|
| 0 | 3 | 6 | 9 | 12 |
| 5 | 8 | 11 | 14 | 2 |
| 10 | 13 | 1 | 4 | 7 |

Similarly the output reordering vectors are obtained, by using the following relationship and determining the values of x and y, such that: such that:

$$5x \equiv 1 \bmod 3 \Rightarrow x = 2$$
$$3y \equiv 1 \bmod 5 \Rightarrow y = 2 \qquad (4.6)$$

Equation (4.5) is rewritten as

$$(5xq_1 + 3yq_2) \bmod 15$$

substituting values of x and y, we get

$$(10q_1 + 6q_2) \bmod 15 \qquad (4.7)$$

where $q_1 = 0,1,2$ and $q_2 = 0,1,2,3,4$.

This relationship gives us the output reordering vectors as

| | | | | |
|---|---|---|---|---|
| 0 | 6 | 12 | 3 | 9 |
| 10 | 1 | 7 | 13 | 4 |
| 5 | 11 | 2 | 8 | 14 |

9- Determination of the transform coefficients.

By definition

$$COSU = \tfrac{1}{2}(e^{jU} + e^{-jU}) \qquad (4.8)$$

$$SINU = 1/j.\ \tfrac{1}{2}(e^{jU} - e^{-jU}) \qquad (4.9)$$

where $U = 2\pi/N$

Since division has no meaning in an NTT, the trignometric functions must be redefined in the number theoretic sense (53). Rewriting equations (4.8) and (4.9).

$$COSU = 2^{-1}(U + U^{-1})$$

$$SINU = 2^{-1}(-j)(U - U^{-1})$$

where $U = \alpha^5 \mod 65521$, and j is an element of order 4, and (from step 4) $\alpha = 7791$.

The multiplier coefficients for the 3-point WFTA and the 5-point WFTA are calculated separately.

(a) Coefficients for the 3-point WFTA

Let $U = \alpha^5 \mod 65521$

$(7791)^5 \equiv 48847 \mod 65521$

$(48847)^{-1} \equiv 16673 \mod 65521$

$m0 = 1$

$m1 = COSU - 1 = 2^{-1}(U + U^{-1}) - 1$

$= 32761.(48447 + 16673) - 1$

$\equiv$ 32760 mod 65521

m2 = SINU - 1 = 32761.41224.24297(48847 - 16673)

$\equiv$ 16087 mod 65521

Similarly the 5-point transform coefficients are calculated in the following manner.

(b) Coefficients for the 5-point transform

Let $U = \alpha^3$ mod 65521

$(7791)^3 \equiv 30887$ mod 65521

$(30887)^{-1} \equiv 28625$ mod 65521

$COSU = 32761 \cdot (30887+30887^{-1}) \equiv 29756$ mod 65521

$SINU = 32761 \cdot 24297(30887-30887^{-1}) \equiv 13367$ mod 65521

$COS2U = COS^2U - SIN^2U \equiv 3004$ mod 65521

SIN2U = 2. SINU. COSU $\equiv$ 49289 mod 65521

m0 = 1

$m1 = 2^{-1}. (COSU + COS2U) - 1 \equiv 16379$ mod 65521

$m2 = 2^{-1}. (COSU - COS2U) \equiv 13376$ mod 65521

m3 = j(SINU + SIN2U) $\equiv$ 19136 mod 65521

m4 = j(SIN2U) $\equiv$ 18005 mod 65521

m5 = j(SINU - SIN2U) $\equiv$ 48647 mod 65521

The coefficients for the 3-point and 5-point transform are now multiplied (mod 65521) together, such that each of the 3-point coefficients is multiplied by each of the 5-point transform coefficients. This multiplication (mod 65521) is performed using a nested 'DO' loop, such that the 5-point transform coefficients are indexed by the inner loop and the 3-point transform coefficients are indexed by the outer loop.

The values of the inverse transform coefficients are obtained in exactly the same manner (as for the forward transform), except that all the SINU are changed to -SINU and the transform coefficients thus obtained are then multiplied by $15^{-1} \equiv 61153 \bmod 65521$.

### 4.3 Architecture of the TMS9900 Microprocessor

Texas Instruments TMS9900 is a single chip 16-bit CPU capable of addressing 64K byte of memory (54), (55). The instruction set of the microprocessor provides full minicomputer capabilities (including I/O). There are sixteen general purpose 16-bit registers (R0 to R15). These registers can be defined any where in the RAM whose location is determined by contents of the workspace pointer. Register to register instructions are executed faster than memory to register or register to memory instructions. The three on chip registers are accessible to the programmer, these registers are:

a) Workspace Pointer (WP): this register holds the address of the current workspace, which is the same as the address of R0.
b) Program Counter (PC): 16-bit program counter holds the address of the current instruction.
c) Status register (ST): this register represents the current machine state.

The workspace concept increases the programming flexibility and more than one program can reside in the memory and executed without affecting the other programs. The workspace pointer can also be changed during the program execution. This allows the

user to redefine a new set of 16 general purpose registers. The special purpose registers R13, R14 and R15 of the current workspace contains the contents of old WP, old PC and old ST respectively, and a return to old workspace reloads these values in the respective registers. This feature is useful when program environment is changed to a subroutine, since in a conventional CPU the entire machine state is saved on the stack, but in case of the TMS9900 only the workspace needs to be changed. A special purpose register R12 holds the base address of the Communications Register Unit (CRU). All the data read or written to the I/O ports must pass through the CRU.

This microprocessor also contains 16 x 16-bit hardware (unsigned) multiply and 32/16-bit (unsigned) divide, and unsigned compare. These features make it suitable for implementation of the NTT.

### 4.4 Implementation on the Microprocessor

A 15-point and a 60-point WFTA were implemented on the TMS9900 microprocessor in assembler language. As there was no software support available with the TMS9900 microprocessor, a mainframe computer was used for program assembly. A utility routine was written in assembler for the TMS9900 to load the object program directly from the mainframe computer into the memory of the microprocessor. This provided an efficient way of testing and debugging the software.

Appendix-B shows an assembler program source listing of the 15-point WFTA implemented on the TMS9900 microprocessor. A FORTRAN program listing of the 15-point WFTA is also included in the appendix-B.

A 60-point WFTA FORTRAN program is listed in (5). A 60-point WFTA was also implemented in the FORTH language, a source program is listed in appendix-C. FORTH is an interactive high level language for microprocessors (56), (57).

The 60-point WFTA has three factors 3, 4, 5, so this transform has a three dimensional structure. In general a transform length with r factors would have an r dimensional structure. The input and output shuffle vectors, forward and inverse transform coefficients are calculated in a similar manner as for the 15-point WFTA. A 120-point WFTA was also implemented in FORTRAN on a mainframe computer.

An A/D (analogue to digital) converter and a D/A (digital to analog) converter was interfaced with the TMS9900 microprocessor system to perform transforms of real time signals.

# CHAPTER 5

## External Hardware Modular Multiplier

### 5.1 Introduction

Microprocessors have found their way into many digital signal processing applications. Multiplication is one of the basic operation in digital signal processing. Hence the need for performing multiplication on the microprocessor efficiently is of vital importance. In many microprocessors no facility is provided for hardware multiply or divide. However, software routines can be written to perform the required multiplication or division operations.

Some of the later versions of microprocessors are provided with signed or unsigned hardware multiplier. For example Motorola's MC6809 microprocessor and Texas Instrument's TMS9900 microprocessor contains an 8 x 8-bit and 16 x 16-bit unsigned hardware multiplier respectively. A considerable amount of time is needed for multiplication even if the hardware multiplier is available. For example, for the MC6809 microprocessor, 173 clock cycles are required to produce a 32-bit unsigned product (clock speed 1-2 MHz), and for the TMS9900 microprocessor 88 clock cycles (clock speed 3 MHz) are needed. As we are interested in the product reduced modulo M, some more time has to be allowed for modularising the 32-bit result. The most obvious and straightforward way to modularise a 32-bit unsigned number is by division. However, for the MC6809 microprocessor this division

requires 1264 clock cycles. In total 1337 clock cycles are required to produce a 16-bit modular product. Typical program coding for 16 x 16-bit (unsigned) multiply and 32/16-bit divide routine for the MC6809 microprocessor is listed in appendix-A. An alternative approach can be adopted in which the hardware multiplier is used to produce a 16-bit modular product which requires then only 336 clock cycles (see appendix-A). In the case of the TMS9900 microprocessor 132 clock cycles are required to perform a 32/16-bit unsigned hardware divide, so the total number of clock cycles is 220. The number of clock cycles required depends upon the addressing mode of the instruction, since register to register instructions are executed much faster than the register to memory instructions.

The time required for modular multiplication can be reduced further by interfacing a high speed external modular multiplier to the system to increase the throughput of the system, thus increasing the range of digital signal processing applications.

Different algorithms may be adopted to implement external multiplication. Either serial or parallel methods may be employed. For a parallel multiplier the cost increases approximately linearly with the number of bits, whereas for a serial multiplier the execution time increases approximately linearly. Davies (28), have described some aspects of performing multiplication on the Z80 microprocessor, and interfacing an external hardware multiplier to it. Weed (29), have described theoretical clockless multiplication and division circuits using 4 x 4-bit multiplier chips. The product of larger numbers can be

obtained by employing more than one multiplier chip and adding the partial products in an appropriate way. In clockless (combinatorial) circuits the total multiplication time is the sum of the propagation delay on the chip, and the carry propagation delay of the adders. This propagation delay increases approximately linearly with the number of input bits. Parasuraman (18), have described a hardware multiplier interfaced to a microprocessor.

## 5.2 Design and Implementation of an External Hardware Modular Multiplier

Large Scale Integration (LSI) techniques now allow the integration of a complete 8 x 8-bit multiplier on a single chip. For example Advanced Micro Devices (44), and TRW (30), (39), (42), have produced single chip 8 x 8-bit (AM25S558) and 16 x 16-bit (MPY-16AJ) multiplier respectively. These multiplier chips have a typical 8 x 8-bit and 16 x 16-bit multiplication time of approximately 45 and 200 nanoseconds respectively. A single chip multiplier (8 x 12-bits) to produce the 13 most significant bits of the product with an internal propagation delay of about 2 nanoseconds have also been reported, additional delay due to external components adds up to 30 nanoseconds (32).

The interfacing of an external hardware multiplier with a microprocessor have been described by Davies and Fung (31). This interfacing can be achieved in two ways. Either it can behave as an I/O peripheral or it may be mapped into the memory space of the microprocessor.

An external hardware modular multiplier (mod 65521) was designed and constructed using wire wrapping techniques. It was interfaced with the TMS9900 microprocessor.

### 5.2.1 Interfacing Considerations

We shall use the term **modular multiplier** for the **external hardware modular multiplier** interfaced with the TMS9900 microprocessor. The two choices to interface the modular multiplier to the TMS9900 are as follows.

i) connect to the I/O port

ii) connect directly to the address and data bus

In the first choice the main disadvantage is that 262 clock cycles are required to communicate with the external modular multiplier through the I/O port. The strobe signals for the modular multiplier must also be generated at the output port. This process is slow since the TMS9900 communicates with the I/O ports through the Communications Register Unit (CRU) serially. The number of clock cycles thus required are more than when the hardware multiply and divide are used to produce the modular product. In the latter arrangement the modular multiplier behaves like an intelligent memory mapped peripheral, with three unique 16-bit addresses. The data is written to two of the addresses and read from the third.

### 5.2.2 Interfacing the Modular Multiplier with the TMS9900 Microprocessor

Figure (5.1) shows a block diagram of the complete (combinatorial) modular multiplier interfaced with the TMS9900 microprocessor. In figure (5.1) and (5.2) lines with arrowheads represent the data bus.

This modular multiplier combines two of the forementioned techniques, using parallel multiplier chips to produce a 32-bit unsigned product and ROM lookup tables whose outputs are combined by a modular adder. The 32-bit unsigned product is reduced modulo 65521 in the following manner. The high order 16-bits of the 32-bit unsigned product pre-multiplied by a fixed constant $2^{16}$- 65521 (=15) are added to the low order 16-bits of the product using a modular adder. Direct storage of the pre-multiplied data would require a 64K x 16-bit ROM. However, if the output is determined by combining partial products derived from the 8 low order bits and the 8 high order bits of the high order 16-bit input, the storage requirement is reduced to two 256 x 16-bit ROMs.

Figure (5.2) shows the block diagram of the modular adder, which consists of three identical 16-bit binary full adders, with two inputs A1 and A2. The output of FA1 is checked by a carry and overflow detector (CD) circuit (figure 5.3). If a carry or an overflow is detected this circuit activates the gate G1 and a value of $2^{16}$- 65521 (=15) is added to the output of FA1 in FA2. This may generate a carry or overflow activating gate G2 adding a further value of 15 in FA3. The output of FA3 is the final

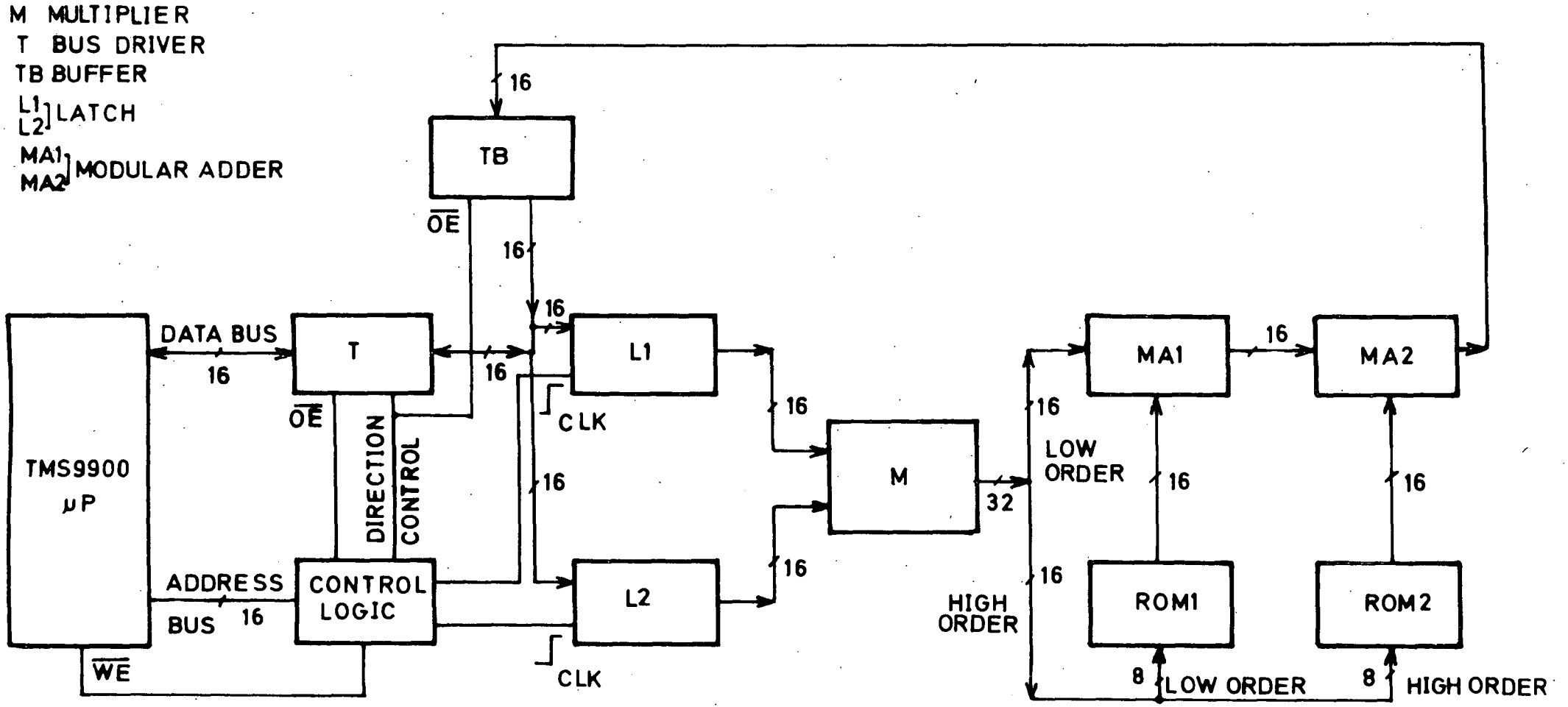



Figure 5.1: Block diagram of the modular multiplier (mod 65521).

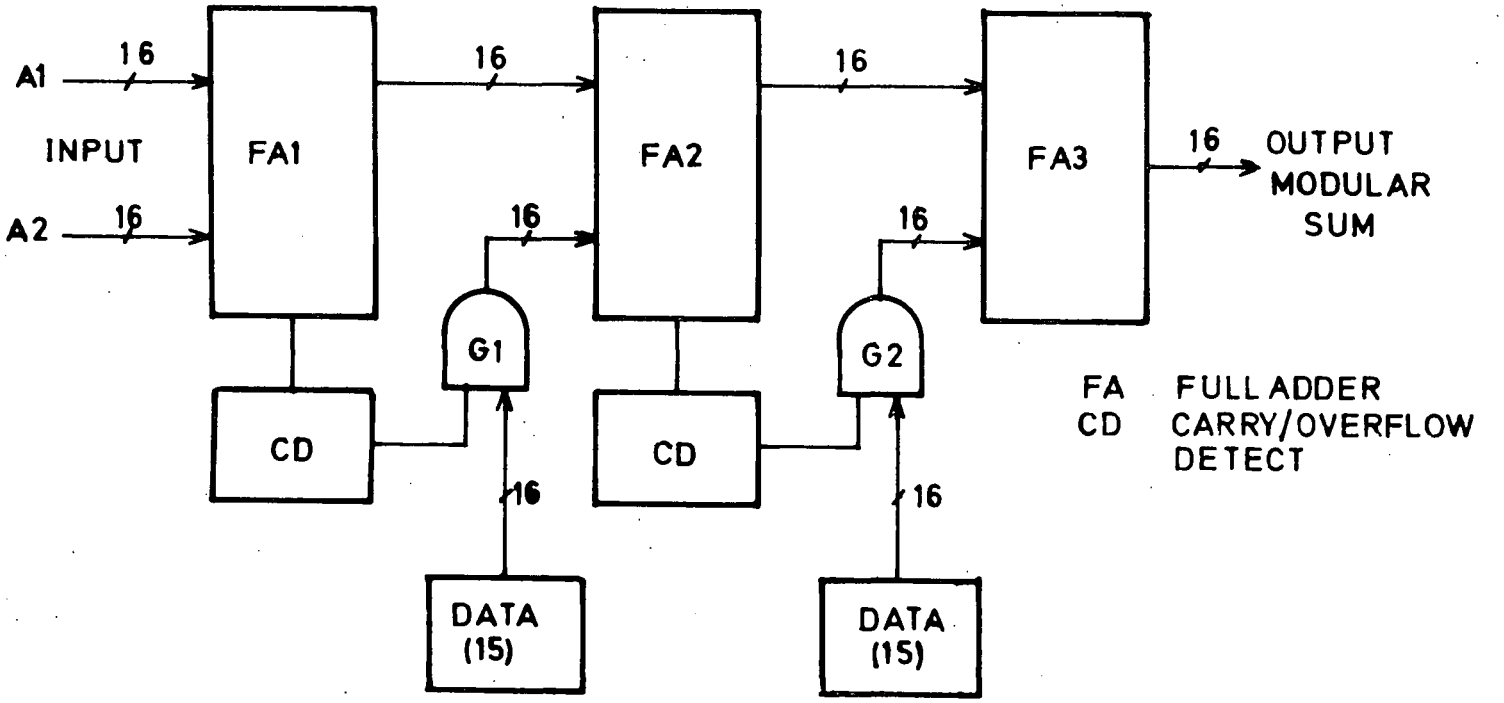



Figure 5.2: Block diagram of the modular adder (mod 65521).

modular sum. A modular adder was designed and constructed for test purpose before implementing it with the modular multiplier.

The basis of this modular multiplier is four (8 x 8-bit) multiplier chips (AM25S558), which achieve a typical 8 x 8-bit multiplication in approximately 45 nanoseconds. These multiplier chips are combined with full adders (SN74LS83) to achieve a 16 x 16-bit to 32-bit multiplication in approximately 110 nanoseconds. Figure (5.4) shows a photograph of the modular multiplier, the four multiplier chips can be seen clearly.

Typical program coding and timings for the hardware multiply and divide operation is shown in figure (5.5), and coding for the use with the external hardware modular multiplier is shown in figure (5.6).

On the first and second move (MOV) instructions the two 16-bit data words are latched in L1 and L2 (SN74LS374) through a bidirectional bus driver T (SN74LS245). Address and control signals for these latches and driver are generated by appropriately decoding the addresses and gating it with the write enable ($\overline{WE}$) line from the TMS9900 microprocessor. The outputs of L1 and L2 are directed to the multiplier M. The 32-bit unsigned product is then split into three parts. The low order 16-bits are connected directly to one of the inputs of the first modular adder MA1. The high order 16-bits are further split into two 8-bit words. The low order half 8-bits are directed to the address bus of ROM1 and the other half 8-bits are directed to the address bus of ROM2. ROM1 and ROM2 are four 256 x 4-bit (AM27S21) PROMs, with a typical access time of 45 nanoseconds.

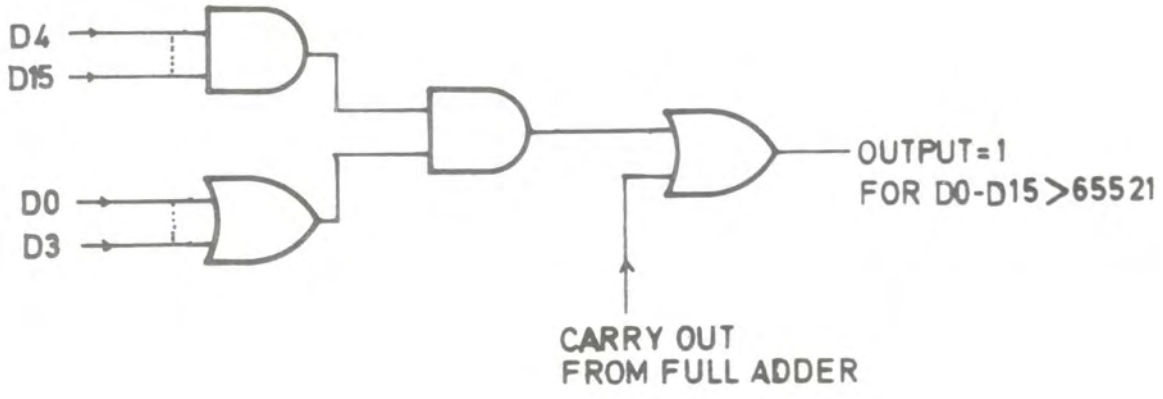



Figure 5.3: Carry and overflow detect circuit.

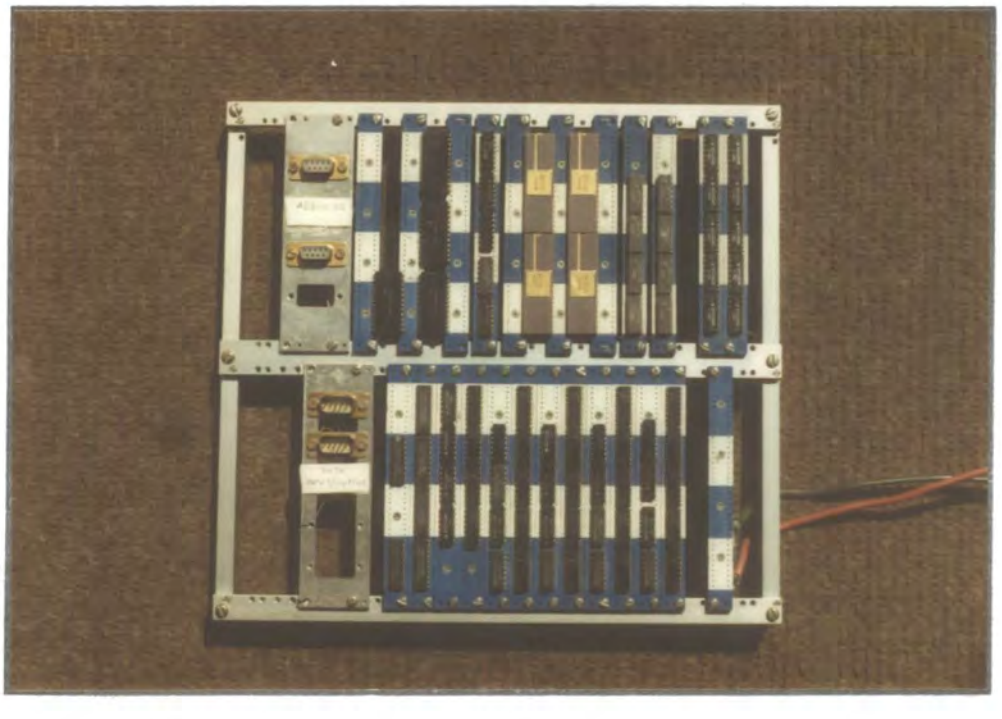


Figure 5.4: Photograph of the external hardware modular multiplier.

```
Clock cycles    Labels    Mnemonics    Operands

14                        MOV          @MPD,R2
88                        MPY          @MPR,R2
132                       DIV          @MOD,R2
----                      RT
234             MOD       DATA         65521
                MPR       DATA         ......
                MPD       DATA         ......

                R3 contains the modular product.
```

Figure 5.5: Program coding for using hardware multiply and divide.

```
Clock cycles    Labels    Mnemonics    Operands

                INPUT1    EQU          >3FF2
                INPUT2    EQU          >3FF4
                OUTPUT    EQU          >3FF6
20                        MOV          @MPR,@INPUT1
20                        MOV          @MPD,@INPUT2
14                        MOV          @OUTPUT,R3
----                      RT
54              MPR       DATA         ......
                MPD       DATA         ......

                R3 contains the modular product.
```

Figure 5.6: Program coding using external modular multiplier.

(> Shows hexadecimal values, and @ shows symbolic names.)

Typical values stored in ROM1 and ROM2 are shown in tables (5.1) and (5.2). The output of ROM1 is connected to an input of the first modular adder MA1. MA1 combines the low order 16-bits of the 32-bit product with the partial product stored in ROM1 from the low order 8-bits of the high order 16-bits. MA2 then adds in the other partial product stored in ROM2. The output of MA2 is finally the 16-bit modular product of the two current 16-bit values in the input latches L1 and L2. The output of these latches, multiplier chips and the PROMs are permanently enabled, so after the second value is latched in L2 the 16-bit modular product is available in less than 500 nanoseconds at the output of MA2. This output can be read back into the microprocessor by activating the tristate buffer TB (SN74LS126) at the output of MA2.

The multiply instruction for the TMS9900 microprocessor works in the following manner. If the multiplicand is in register Rn and the multiplier is in register Rm. Then after the multiply instruction Rn:Rn+1 holds the product and Rm remains unchanged. For example, if register R2 contains $FFFF, and R3 contains $FFFF, then after the multiplication the register pair R3:R4 contains $FFFE0001, where ':' shows concatenation of two registers to form a register pair to hold the 32-bit product.

The division operation also utilises a (consecutive) register pair to hold the quotient and the remainder. Initially the dividend is held in a register pair Rn:Rn+1. After the division the Rn holds the quotient and Rn+1 holds the remainder. For example, if R2 contains the divisor ($0005) and R3:R4

*Table 5.1: Values in ROM 1*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 390 | 780 | 1170 | 1560 | 1950 | 2340 | 2730 | 3120 | 3510 |
| 15 | 405 | 795 | 1185 | 1575 | 1965 | 2355 | 2745 | 3135 | 3525 |
| 30 | 420 | 810 | 1200 | 1590 | 1980 | 2370 | 2760 | 3150 | 3540 |
| 45 | 435 | 825 | 1215 | 1605 | 1995 | 2385 | 2775 | 3165 | 3555 |
| 60 | 450 | 840 | 1230 | 1620 | 2010 | 2400 | 2790 | 3180 | 3570 |
| 75 | 465 | 855 | 1245 | 1635 | 2025 | 2415 | 2805 | 3195 | 3585 |
| 90 | 480 | 870 | 1260 | 1650 | 2040 | 2430 | 2820 | 3210 | 3600 |
| 105 | 495 | 885 | 1275 | 1665 | 2055 | 2445 | 2835 | 3225 | 3615 |
| 120 | 510 | 900 | 1290 | 1680 | 2070 | 2460 | 2850 | 3240 | 3630 |
| 135 | 525 | 915 | 1305 | 1695 | 2085 | 2475 | 2865 | 3255 | 3645 |
| 150 | 540 | 930 | 1320 | 1710 | 2100 | 2490 | 2880 | 3270 | 3660 |
| 165 | 555 | 945 | 1335 | 1725 | 2115 | 2505 | 2895 | 3285 | 3675 |
| 180 | 570 | 960 | 1350 | 1740 | 2130 | 2520 | 2910 | 3300 | 3690 |
| 195 | 585 | 975 | 1365 | 1755 | 2145 | 2535 | 2925 | 3315 | 3705 |
| 210 | 600 | 990 | 1380 | 1770 | 2160 | 2550 | 2940 | 3330 | 3720 |
| 225 | 615 | 1005 | 1395 | 1785 | 2175 | 2565 | 2955 | 3345 | 3735 |
| 240 | 630 | 1020 | 1410 | 1800 | 2190 | 2580 | 2970 | 3360 | 3750 |
| 255 | 645 | 1035 | 1425 | 1815 | 2205 | 2595 | 2985 | 3375 | 3765 |
| 270 | 660 | 1050 | 1440 | 1830 | 2220 | 2610 | 3000 | 3390 | 3780 |
| 285 | 675 | 1065 | 1455 | 1845 | 2235 | 2625 | 3015 | 3405 | 3795 |
| 300 | 690 | 1080 | 1470 | 1860 | 2250 | 2640 | 3030 | 3420 | 3810 |
| 315 | 705 | 1095 | 1485 | 1875 | 2265 | 2655 | 3045 | 3435 | 3825 |
| 330 | 720 | 1110 | 1500 | 1890 | 2280 | 2670 | 3060 | 3450 | |
| 345 | 735 | 1125 | 1515 | 1905 | 2295 | 2685 | 3075 | 3465 | |
| 360 | 750 | 1140 | 1530 | 1920 | 2310 | 2700 | 3090 | 3480 | |
| 375 | 765 | 1155 | 1545 | 1935 | 2325 | 2715 | 3105 | 3495 | |

*Table 5.2: Values in ROM2*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 34319 | 3117 | 37436 | 6234 | 40553 | 9351 | 43670 | 12468 | 46787 |
| 3840 | 38159 | 6957 | 41276 | 10074 | 44393 | 13191 | 47510 | 16308 | 50627 |
| 7680 | 41999 | 10797 | 45116 | 13914 | 48233 | 17031 | 51350 | 20148 | 54467 |
| 11520 | 45839 | 14637 | 48956 | 17754 | 52073 | 20871 | 55190 | 23988 | 58307 |
| 15360 | 49679 | 18477 | 52796 | 21594 | 55913 | 24711 | 59030 | 27828 | 62147 |
| 19200 | 53519 | 22317 | 56636 | 25434 | 59753 | 28551 | 62870 | 31668 | 466 |
| 23040 | 57359 | 26157 | 60476 | 29274 | 63593 | 32391 | 1189 | 35508 | 4306 |
| 26880 | 61199 | 29997 | 64316 | 33114 | 1912 | 36231 | 5029 | 39348 | 8146 |
| 30720 | 65039 | 33837 | 2635 | 36954 | 5752 | 40071 | 8869 | 43188 | 11986 |
| 34560 | 3358 | 37677 | 6475 | 40794 | 9592 | 43911 | 12709 | 47028 | 15826 |
| 38400 | 7198 | 41517 | 10315 | 44634 | 13432 | 47751 | 16549 | 50868 | 19666 |
| 42240 | 11038 | 45357 | 14155 | 48474 | 17272 | 51591 | 20389 | 54708 | 23506 |
| 46080 | 14878 | 49197 | 17995 | 52314 | 21112 | 55431 | 24229 | 58548 | 27346 |
| 49920 | 18718 | 53037 | 21835 | 56154 | 24952 | 59271 | 28069 | 62388 | 31186 |
| 53760 | 22558 | 56877 | 25675 | 59994 | 28792 | 63111 | 31909 | 707 | 35026 |
| 57600 | 26398 | 60717 | 29515 | 63834 | 32632 | 1430 | 35749 | 4547 | 38866 |
| 61440 | 30238 | 64557 | 33355 | 2153 | 36472 | 5270 | 39589 | 8387 | 42706 |
| 65280 | 34078 | 2876 | 37195 | 5993 | 40312 | 9110 | 43429 | 12227 | 46546 |
| 3599 | 37918 | 6716 | 41035 | 9833 | 44152 | 12950 | 47269 | 16067 | 50386 |
| 7439 | 41758 | 10556 | 44875 | 13673 | 47992 | 16790 | 51109 | 19907 | 54226 |
| 11279 | 45598 | 14396 | 48715 | 17513 | 51832 | 20630 | 54949 | 23747 | 58066 |
| 15119 | 49438 | 18236 | 52555 | 21353 | 55672 | 24470 | 58789 | 27587 | 61906 |
| 18959 | 53278 | 22076 | 56395 | 25193 | 59512 | 28310 | 62629 | 31427 | |
| 22799 | 57118 | 25916 | 60235 | 29033 | 63352 | 32150 | 948 | 35267 | |
| 26639 | 60958 | 29756 | 64075 | 32873 | 1671 | 35990 | 4788 | 39107 | |
| 30479 | 64798 | 33596 | 2394 | 36713 | 5511 | 39830 | 8628 | 42947 | |

contains dividend ($0000005B) then after the divide instruction R3 will contain ($0012) and R4 will contain ($0001).

The dividend must be in a register pair (right justified). Before performing the division the microprocessor checks if the divisor is greater than the most significant word of the dividend. If the divisor is greater then normal division takes place. However, if the divisor is smaller than the most significant word of the dividend then overflow bit in the status register is set and the division operation is aborted, and the dividend remains unchanged.

In figure (5.5) register pair (R2:R3) holds the 32-bit unsigned product resulting from a multiply (MPY) instruction. After a divide (DIV) instruction R2 holds the quotient and R3 holds the remainder.

Comparing the two values in figure (5.5) and figure (5.6) shows a saving of 180 clock cycles for a single modular multiplication. For a clock frequency of 3 MHz the total time saved for each modular multiplication is 60 microseconds.

## 5.3 Results

A 15-point and a 60-point WFTA transform were run on a TMS9900 microprocessor, requiring 18 and 72 multiplications respectively. The execution time for a 15-point WFTA is about 4 milliseconds and for a 60-point WFTA is about 32 milliseconds using the hardware multiply and divide instructions. When the external hardware modular multiplier is implemented, execution

time is reduced to about 3 milliseconds for a 15-point transform, and to about 28 milliseconds for a 60-point transform.

A 60-point WFTA implemented in FORTH requires about 739 milliseconds to execute. When the external hardware modular multiplier is used, a saving of 3 milliseconds is achieved.

An interesting point to note is that the modular multiplier generates the 16-bit modular product between the second and third move (MOV) instruction. If the modular multiplier had been slower, then a delay routine would be required between latching the second operand into the modular multiplier and reading the modular product from it.

The modular multiplier was tested extensively. A test routine for the TMS9900 microprocessor was written to check all the possible input combinations of the multiplier and the multiplicand. The modular product obtained from the modular multiplier were compared with modular product of the same two numbers calculated by the microprocessor itself.

Total cost of this external hardware modular multiplier is approximately £ 400 (1980), which is dominated by the cost of the four multiplier chips. Total power consumption is about 16 watts and 81 i.c. packages are used in all.

# CHAPTER 6

## Multi Processor and Parallel Processor Systems

### 6.1 Introduction

A Central Processing Unit (CPU) fetches instructions from its program memory sequentially under the program control (see figure 6.1). These instructions are then decoded and executed. Each instruction may differ in length depending upon the mode of instruction. These instructions are visualised as stream of instructions and operands as stream of data.

The data are manipulated in the CPU registers and the results are stored back in the memory. The arithmetic operations performed in the CPU registers are much quicker than the register to memory or memory to register operations. The onchip registers are also referred to as scratchpad registers. Some of the onchip registers are not accessible to the programmer and are entirely used by the CPU.

### 6.2 System Organisation

Suppose that a processor P is operating at full speed and capacity. Let $M_I$ and $M_D$ be the minimum number of instruction and data stream respectively. The computer systems can then be organised into four different ways according to the instruction and data stream.

### 6.2.1 Single Instruction Single Data (SISD) Machine

In this type of system $M_I = M_D = 1$. This arrangement is typical of a uni processor (with a single Arithmetic-Logic Unit (ALU), and a Control Unit) system. A single instruction I is fetched from the program memory sequentially under the ALU control, and is decoded by the ALU and then executed in m subinstructions $s_1, s_2, \ldots, s_m$ (as shown in figure 6.2). The data are obtained from the data memory, and after the calculations the results are stored back into it. Each instruction represents one arithmetic operation on input data D entering the ALU to generate the output data $D'$.

### 6.2.2 Single Instruction Multiple Data (SIMD) Machine

In this case $M_I = 1$, and $M_D > 1$. Figure (6.3) shows a typical SIMD machine. There are m number of processors P. These processors are arranged in such a manner that the same instruction stream performs operations on m seperate input data streams $D_1, D_2, \ldots, D_m$. To generate the output data streams $D_1', D_2', \ldots, D_m'$. This arrangement is typical of an array processor, with a single control unit with some arrangement to broadcast instructions to the desired processors.

### 6.2.3 Multiple Instruction Multiple Data (MIMD) Machine

In this type of system $M_I > 1$ and $M_D > 1$. Figure (6.4) shows a typical MIMD machine. Processors P are arranged such that each one is distinct and separate, and a separate

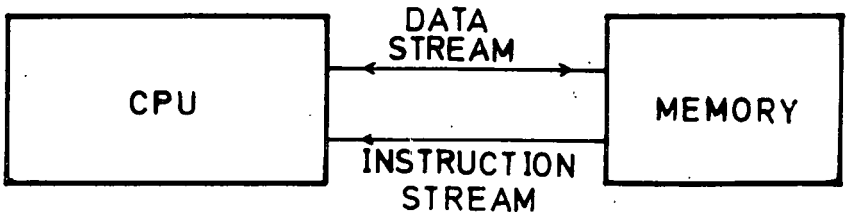



Figure 6.1: Conventional uni processor system.

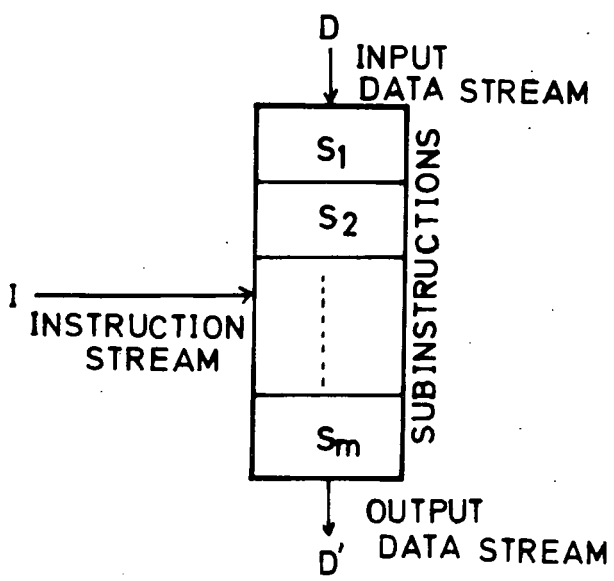



Figure 6.2: A Single Instruction Single Data (SISD) machine.

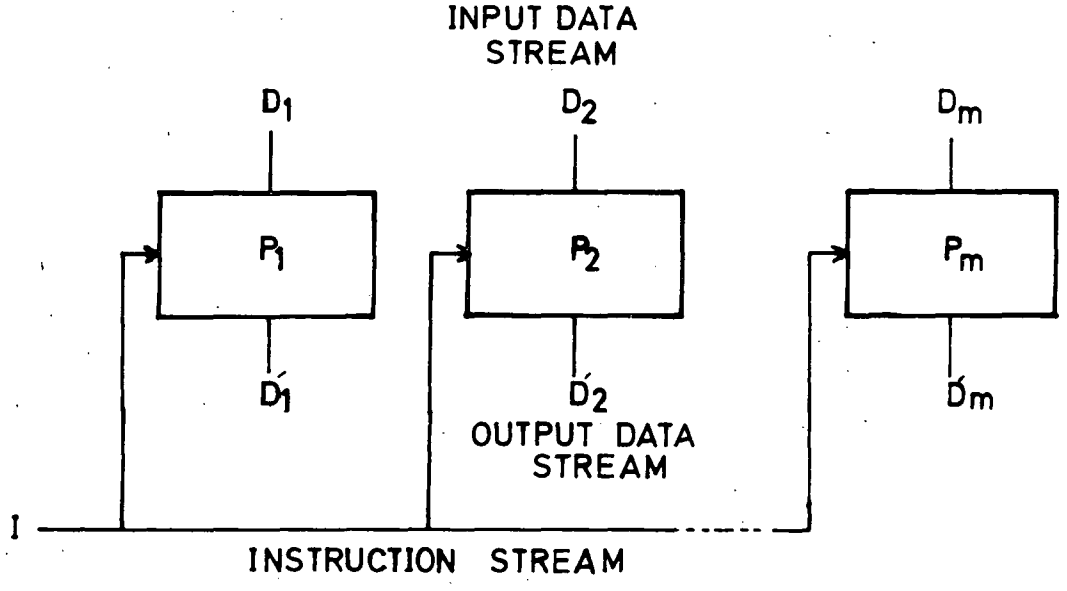



Figure 6.3: A Single Instruction Multiple Data (SIMD) machine.

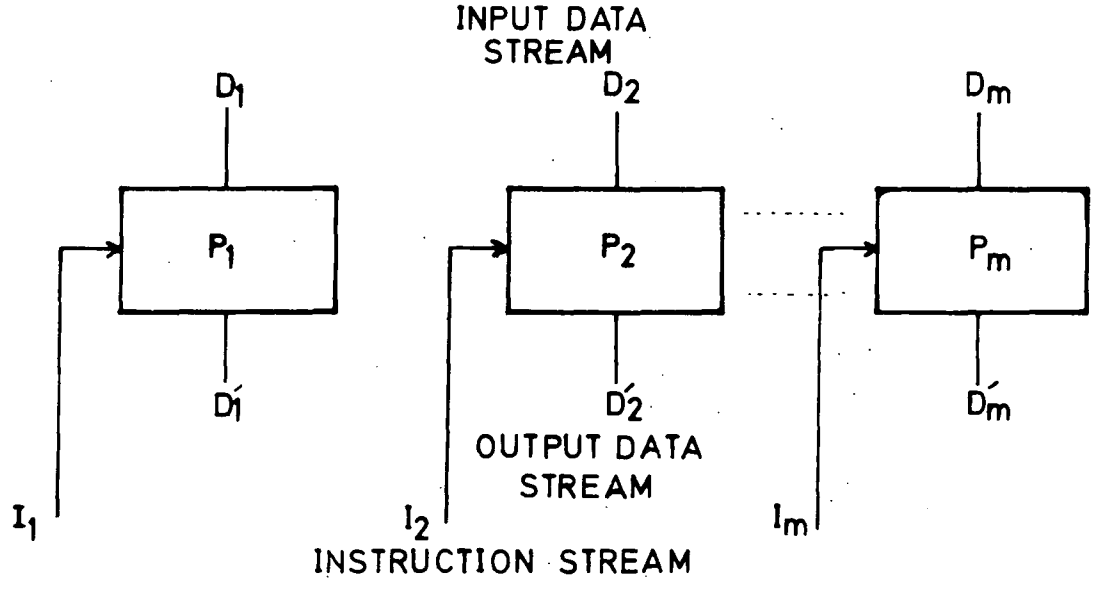



Figure 6.4: A Multiple Instruction Multiple Data (MIMD) machine.

instruction stream is applied to each of the m processing units.

Let each of the processing units have separate input data streams $D_1$, $D_2$,..., $D_m$ to generate the output data streams $D'_1$, $D'_2$,..., $D'_m$. This system executes several independent programs concurrently. It basically forms a multi processor system, such that each processor has a separate program memory.

### 6.2.4 Multiple Instruction Single Data (MISD) Machine

In this case $M_I > 1$ and $M_D = 1$. Figure (6.5) shows a typical MISD machine. The same data passes through different segments. The same set of data D is being operated upon by m instructions to generate the output $D'$. This arrangement can also be called as an m-segment pipeline processor. A pipeline processor requires more hardware and complex circuitry, but has high speed operation. Each of the segments is separated by a buffer register to hold intermediate results.

## 6.3 Multi Processor Systems

Experience reveals that parallelism in hardware circuitry increases the throughput of the system. Increasing the level of parallelism increases the potential operating speed but also the hardware and the cost.

Consider a uni processor system with programmed I/O devices. A CPU performs I/O routines to transfer data to and from the I/O devices using polling. Polling is a scheme in which the CPU

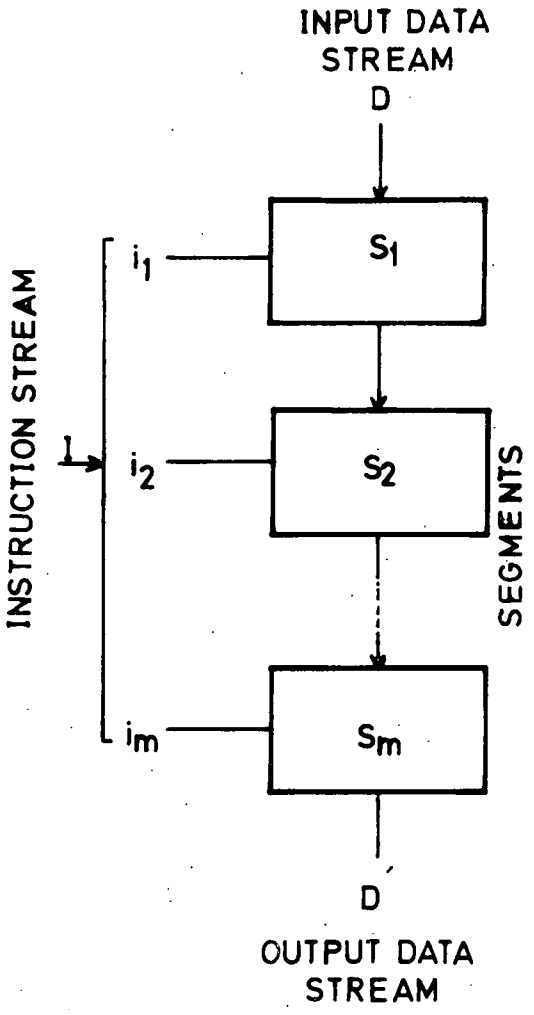



Figure 6.5: A Multiple Instruction Single Data (MISD) machine.

periodically checks the I/O devices to see if any of the devices needs servicing. The system would tend to slow down when the CPU is interfaced to rather slow mechanical devices e.g. a card reader, or a line printer etc. An improvement on programmed I/O method of data transfer is to implement interrupts. In this case the CPU does not poll any of the devices, but when the peripheral or I/O device is ready to receive/transmit data it sends an interrupt signal to the CPU. The CPU branches to the appropriate interrupt service routine, and after performing I/O routines resumes normal operation. A further improvement would be to employ I/O Processors (IOPs) also called Peripheral Processing Units (PPUs). These reduce considerably the load on the main CPU. The IOPs share common memory with the main CPU. But the CPU still initiates and terminates all the data transfer operations. The main CPU behaves as a master and the IOPs as slaves.

The advantage of employing CPU and IOPs side by side is that both can execute their programs concurrently and independently of each other. This basically forms a type of multi processor system. Figure (6.6a) shows a single shared link between memory and I/O devices for local communications. The speed of the system may suffer if the I/O devices are very slow. However, figure (6.6b) shows another arrangement with dual bus, in this case I/O devices are controlled by an IOP (22).

In most practical systems it is required by the processors to communicate with each other. The multi processor systems can be classified as directly or indirectly coupled, which depends upon the method of data exchange.

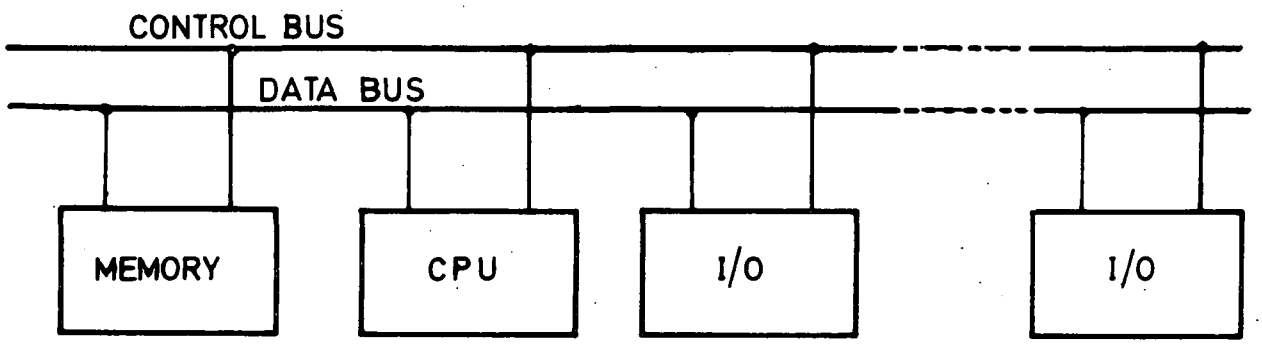



Figure 6.6a: Local communication between CPU and memory and I/O connected through a shared bus.

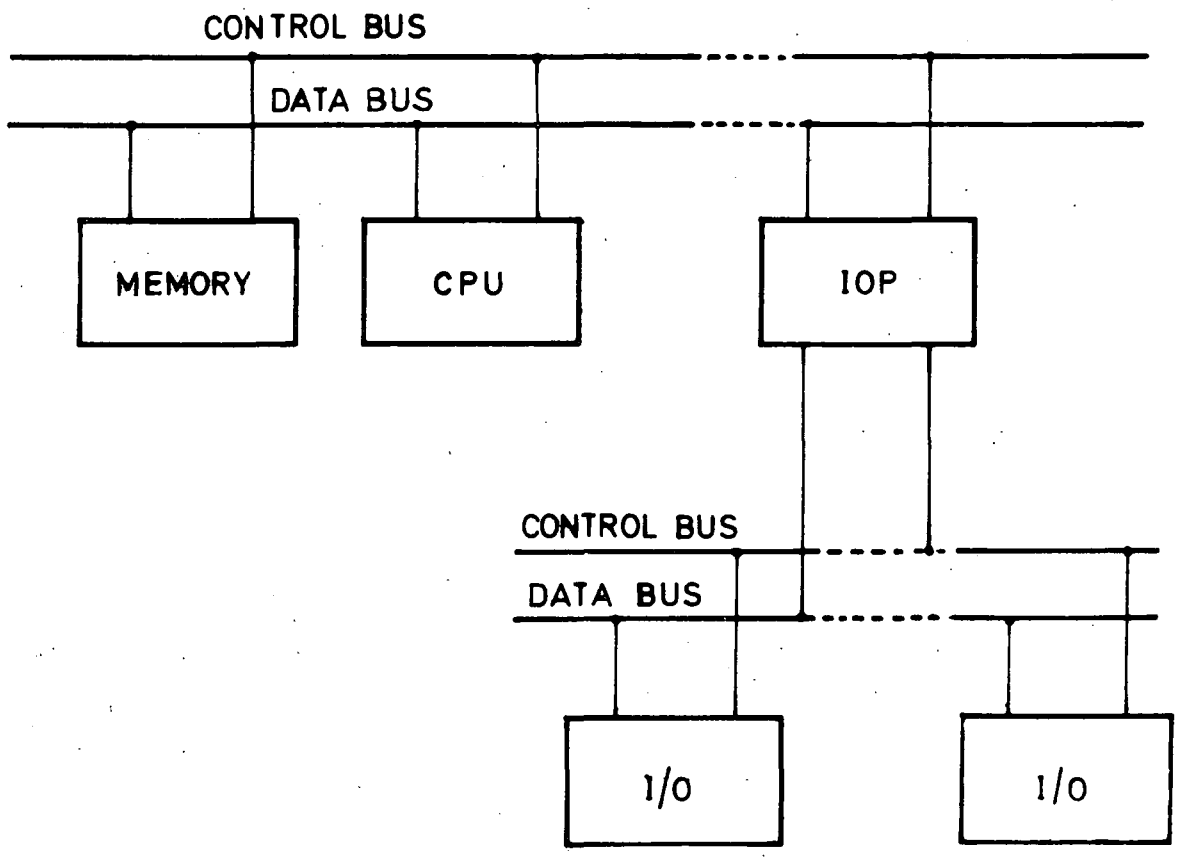



Figure 6.6b: Local communications with memory and several I/O through IOP using dual bus structure.

### 6.3.1 Directly Coupled Multi Processor Systems

A multi processor system is defined as a computer system with more than one CPU, sharing a common memory and I/O devices. The CPUs co-operate with each other at hardware and software level, and exchange data with each other through common memory when required (73). This is known as a directly or tightly coupled multi processor system.

Scales (77), have described two kinds of directly coupled multi microprocessor systems using Motorola's MC6809E microprocessor, namely global-only and local/global type. He has also discussed the basic hardware differences between the MC6809 and the MC6809E version of the microprocessor. The MC6809E version requires an external (TTL) clock, but the MC6809 has an onchip oscillator, which operates by an external crystal. The MC6809E version provides output signals suitable for a multi microprocessor environment.

In the global-only type, the microprocessors continuously use the same global bus, because all the microprocessors share the common (global) memory. The efficiency of the system is low. Each microprocessor is granted the bus by the bus arbiter at the begining of each cycle of the clock E. One of the microprocessors has higher priority than the rest of the microprocessors such that the system behaves as a master and slaves. The master acquires the global bus on powerup reset to initialise the system and peripherals, while the other microprocessors execute the SYNC instruction and wait for the interrupt after the reset has been activated. The priority of

the microprocessors is in round-robin manner. At any instant only one microprocessor uses the global bus and the clocks are stretched for other microprocessors. The maximum time for which the clock can be stretched is 10 microseconds without loss of data.

In the local/global system each of the microprocessor has its own local program and data memory connected to the microprocessor by the local data and address buses. In addition there is a global memory, data bus, address bus and global I/O devices. Each of the microprocessors is allocated a different task, for example one of them performs the I/O operations, the other runs the operating system, and the control microprocessor supervises the entire system.

A bus arbiter controls the flow of the data from the microprocessors to the global memory and global I/O devices. Each of the microprocessors is executing program from its own local program memory using its local bus. If any of the microprocessors wishes to access the global memory, it puts a request to the bus arbiter which makes sure that only one microprocessor is accessing the global bus at a time to prevent bus contention. If two microprocessors simultaneously request the bus arbiter to access the global memory, the bus is granted by the bus arbiter to the microprocessor which has higher priority, and sends the other microprocessors into a wait state with their clocks stretched until the first one has finished the data transfer into the global memory or the global I/O device. As long as the microprocessors are executing programs from their

own local program memories the speed and efficiency of the system is a maximum, but as soon as more than one microprocessor wishes to access the global memory or I/O device, the speed of the system suffers. The number of microprocessors which can be interconnected in this manner is limited (4 in this case).

Hoffner and Smith (68), have described a tightly coupled multi processor system. This system employs two MC6809 microprocessors. These two microprocessors are operated by opposite phases of a common clock. This prevents simultaneous access by the microprocessors to the common memory. The memory in this system should be twice as fast as the processor read cycle, to prevent contention. The processors are connected through a parallel interface buffer to a common memory. The advantage in this system is that in one cycle one of the processor is writing into the memory, while in the next (anti-phase) clock the other processor can read this particular byte. The major drawback in tightly coupled multi processor systems in general is the memory conflict. The method described above circumvents memory conflict problem (limited to 2 microprocessors only).

### 6.3.2 Indirectly Coupled Multi Processor Systems

Indirectly coupled multi processor systems in contrast do not share a common memory (73). The data exchange takes place through an other medium like magnetic tape, magnetic disk or I/O ports etc. Each of the CPUs has its own associated memory. In loosely or indirectly coupled multi processor systems the

processors work more autonomously as compared to tightly coupled systems.

Bellm and Sauer (64), have described three different methods for data exchange between two Intel 8080 microcomputers.

The first method involves parallel data transfer through Programmable Peripheral Interface (PPI) using I/O ports. A further port is used for handshaking. These handshaking signals are also referred to as semaphores. Semaphores are memory locations under the software control which act as flags indicating the presence or absence of data. When one microcomputer transfers the data into its output port, it sets a 1-bit flag in the other output port. This port is being continuously monitored by the other microcomputer, when it is expecting data from the other microcomputer. When the signal on a particular bit changes, the destination microcomputer reads the output port of the source microcomputer. The destination microcomputer then acknowledges this by setting a bit in its own output port. This port is being monitored by the source microcomputer (after it has transferred data to its output port). The source microcomputer after receiving this acknowledgement sends the next data byte. The data transfer can be in either direction, i.e. each of the two microcomputers can at one instant behave as source, and in the next instance as destination. A loop counter determines the number of data bytes to be transmitted and/or received.

The second method uses interrupts. When the data are available at the output port the source sends an interrupt to the destination. After executing the interrupt routine the two microcomputers can resume their normal operation independently. Data exchange still takes place through input and output ports. The destination microcomputer then reads the data, and sends an acknowledge signal back to the source.

The third method employs Direct Memory Access (DMA). The source microcomputer sends a request for DMA to the destination microcomputer. The destination microcomputer forces its address and data buses into high impedance state. The source can then access the address and data buses of the destination microcomputer to access its memory. Then the source microcomputer can write into this remote memory as if it were its own memory. A tristate buffer is required to isolate the common buses of the two microcomputers (67), (77). During the DMA the destination microcomputer is not executing any program. After the DMA is complete a signal transmitted to the destination microcomputer restarts it. This method of data transfer requires complex circuitry. Tanabe and Matsumoto (74) have described a dual bus microprocessor. This microprocessor is capable of behaving as a master or a slave depending upon a control signal. The dual bus architecture allows use of both the buses (local and global) simultaneously, for example on the internal bus the CPU is executing its program, while the external bus is being used for DMA. This prevents the microprocessor idling during DMA, thus increasing the throughput.

### 6.4.1 Time-shared Bus

A time-shared bus is sometimes also referred to as a shared bus (22), (71), (72). This is a single bus which is used by several processors to communicate with each other, or with some other processor or I/O device at different intervals of time. A shared bus has more than one source and destination. The shared bus may be unidirectional or bidirectional. The data transfer rate is low but the cost is also low. The complexity of the hardware and control function increases with the increase in the number of processors on the bus. A major disadvantage is that only one processor can act as a source at a time, and the rest of the processors are effectively cutoff from the bus during this period see figure (6.7). A bus arbiter or a multiplexer controls the dynamic communication path between the two devices. Additional systems can be connected to the bus, without major alterations in the link, provided that the arbiter has the capacity to control all the devices. Such a system is called a modular system.

### 6.4.2 Dedicated Link

A dedicated link is the one in which there is only one source and one destination per link see figure (6.8). A dedicated link provides high speed communications at the expense of increased cost. These dedicated links can either be unidirectional or bidirectional. If an additional device is to be connected to the n-device system then n(n-1)/2 number of links are required. This kind of system is non-modular.

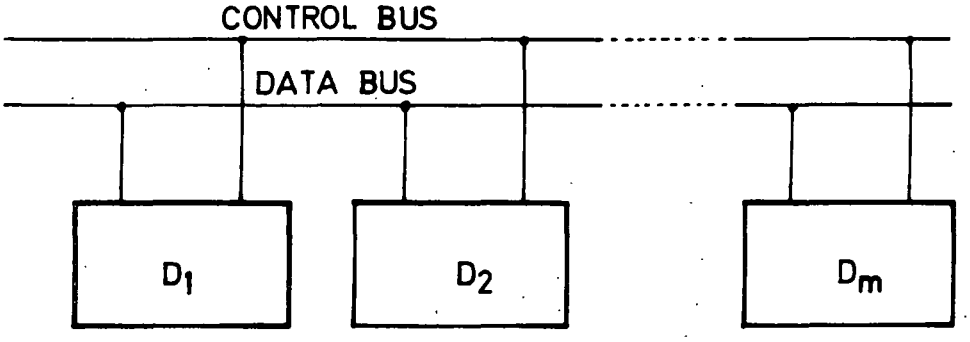



Figure 6.7: A shared bus system.

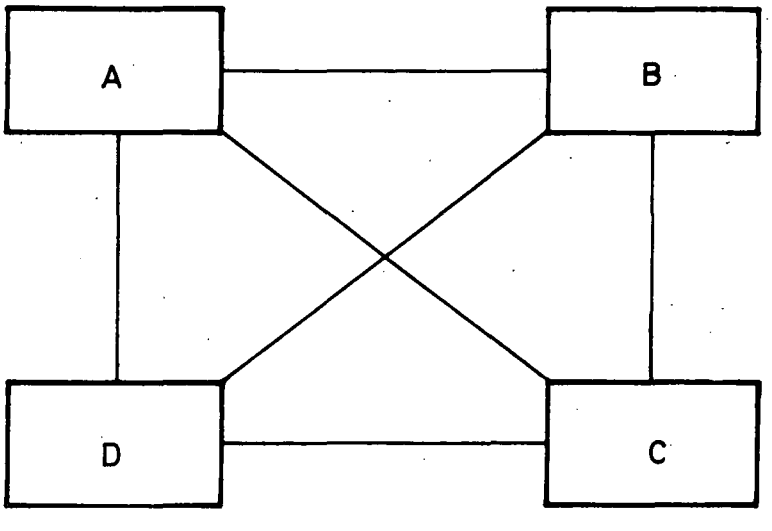



Figure 6.8: Several devices interconnected through dedicated link.

## 6.5 Parallel Processor Systems

The term parallel processing is used in a very general sense, which involves methods to improve computational speed by performing calculations simultaneously or in parallel.

Each of the CPUs has its own local memory (RAM and ROM). These local memories are not accessible to any other processor, not even to the master. The role of the master in this configuration is to control the data flow to and from the slaves. The master can also initiate the task. This type of system is useful in implementing algorithms with inherent parallelism (59), (61). Then a big task is broken down into subtasks and each processor is allocated a subtask (73). The processors communicate with each other through the I/O ports or dedicated buses. A master processor supervises the entire system. The master is capable of communicating with all the slaves. This kind of system is of dedicated type, and it is not very suitable for general applications. Another approach to such a system is that the master is capable of accessing the local memory of the slave(s). This makes the system programmable and more flexible, i.e. the master can transfer program(s) into the local memory of the slave(s) and request them to execute this program on a particular set of data (63). After completing the task the slave(s) informs the master and goes into an idle state and waits for the next task. This method is also useful when the raw data is to be preprocessed to be used at a later stage during the program execution by the master.

A parallel processor system basically forms an MIMD machine. All the processors are under the control of a central control unit. Increased parallelism makes the system special purpose or dedicated, while low order of parallelism makes the system less efficient. Parallelism in a particular problem is obtained by examining the size and type of the problem.

FFT type algorithms can be implemented on a parallel processor system provided that the data exchange among the processors are performed in an efficient manner (1).

Parallelism in an algorithm is defined as number of arithmetic operations that are independent and can therefore be performed in parallel i.e. concurrently. A system which can then utilise this parallelism in full would give a highly efficient system.

## 6.6 Array Processors

A processor is defined as a computer without a control unit (66). These processors can be arranged into arrays with a single control unit. These processors are then much easier to design using integrated circuit technology on a single chip. This would basically form an SIMD machine. The control unit, depending upon the instruction, can disable or enable a particular processor. If a separate control unit is provided for each processor then it would work more autonomously, but still working under the control of a central control unit.

Performance of an array processor is the (data) bandwidth or maximum throughput measured in terms of maximum number of results that can be generated per unit time. One measure often used for high performance machines is the number of floating point operations per second (flops). Sometimes a bigger unit, megaflops (million floating point operations per second), is also used.

Array processors are employed for implementation of algorithms which have inherent parallelism (62), (70). Each processor share the task of processing the data, the load on each processor should be kept at the same level. As the processors are physically located in close proximity to each other, parallel connection exists between them. Each processor can have its own program and data memory. The control unit can appropriately enable or disable the processors as required.

## 6.7 Processor - Memory Interconnection

Processor to memory interconnection is one of the essential factors to be considered while designing a multi processor system. The connection to the main memory with a number of processors can be achieved by multiplexing through a switching network (87).

Figure (6.9) shows a cross-bar switch matrix interconnecting processors P and memory or I/O modules M. The advantage of this arrangement is that the connection between several processors and memory modules can be achieved simultaneously, provided they are accessing different memory modules. In this case the efficiency

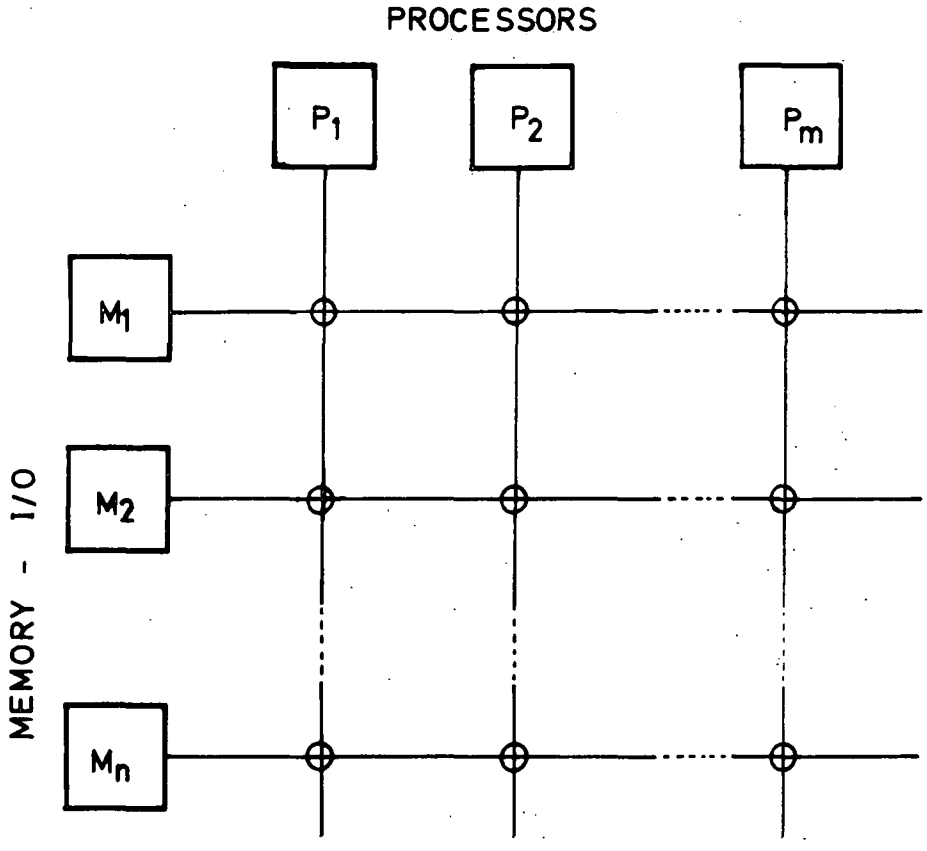



Figure 6.9: Processor-memory interconnection through a cross-bar switch.

S SWITCH
P PROCESSOR

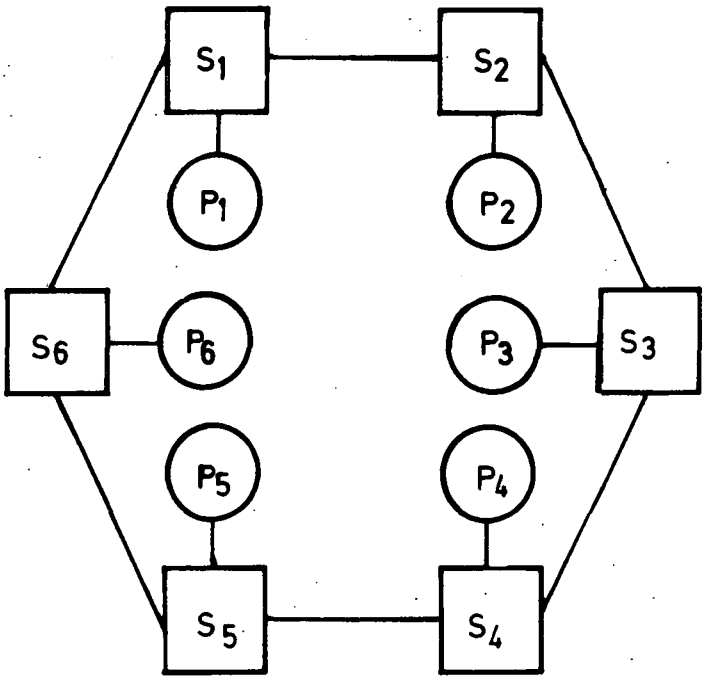



Figure 6.10: Several processors connected to a ring through switches.

would be a maximum. Some arrangement must be provided to prevent simultaneous access by two or more processors to a common memory. The cost of a large cross-bar switch may exceed the total cost of the rest of the system.

Arden and Berenbaum (65), have described a switch with four ports, of which one is the input port and the rest are output ports. The connections of the input port to any of the output port can be achieved by proper addressing. These three output ports can further be connected to similar switches which can extend the capability of the processor to access a bank of memories. But care should be taken not to connect more than one processor to the same memory module accessing a different address. This is referred to as memory interference and it is entirely under software control. Another kind of contention in a multi processor system which can arise is the access of the common system routines or tables. This kind of contention is called system contention. To overcome this problem the routines must be reentrant. A reentrant routine is the one which can be executed by several different processors simultaneously, data should be in different data memory for each processor.

Interleaved memories may also be implemented. In an interleaved memory structure even and odd addresses are located in different memories, such that they can be accessed one after the other in quick succession. This reduces the constraints due to the low access time of the memory. For instance the processor fetches the instruction (op code) from the even address, in the next cycle it will fetch operands from the odd address memory

module.

## 6.8 Computer Systems

The computer systems can be connected in several ways, few of them are described below.

### 6.8.1 Ring Structure

A ring or mesh network is shown in figure (6.10) (22). The ring structure is used for long distance communications or local area networks. The switches S1 to S6 behave as multiplexers, the processors P1 to P6 are interconnected through these switches. Each of the processor before transmitting the data sends the address of the destination processor to the link. Appropriate switch is selected and then the data is transmitted. A particular switch then selects its local processor as the destination and routes the data to its local processor, otherwise forwards it to the next switch in sequence. This form of network is modular. A facility in the system to reconfigure itself in case of a switch failure makes the system more reliable.

### 6.8.2 Star Link

A star link shown in figure (6.11) consists of centralised controller C. Processors talk with each other through this central control switch. Failure of the control switch C would cripple the entire system.

C CENTRAL CONTROLLER

P PROCESSOR

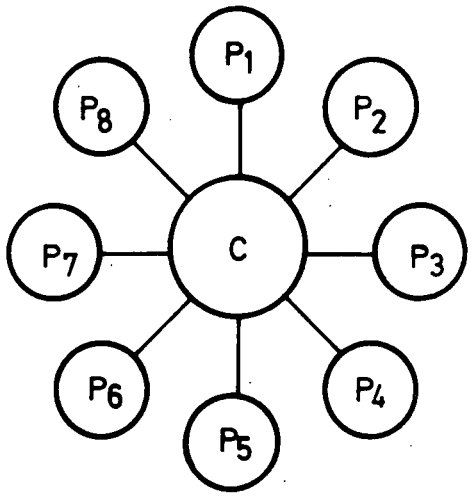



Figure 6.11: Several processors connected to a central control switch to form a star configuration.

P PROCESSOR

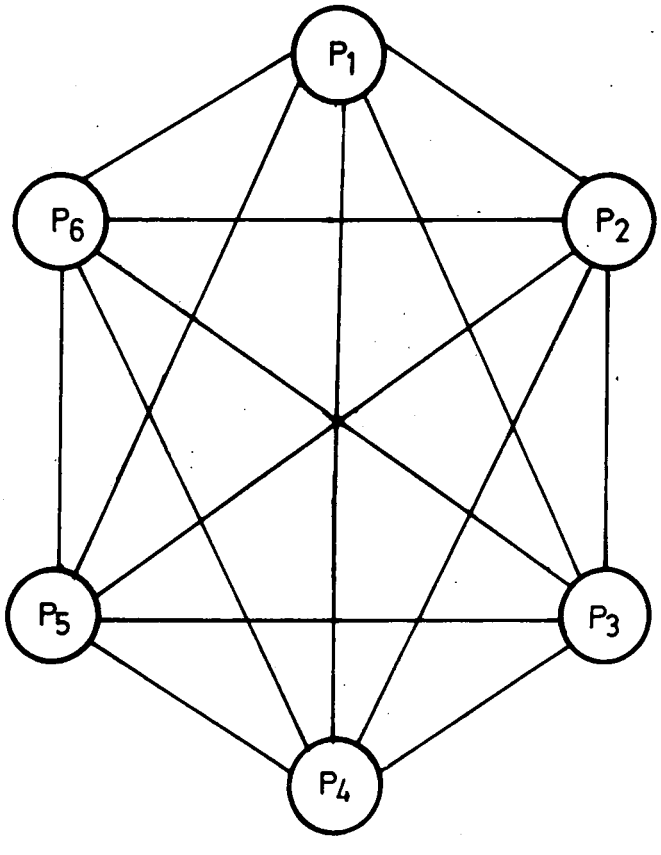



Figure 6.12: Fully connected multi processor network.

### 6.8.3 Fully Connected Link

A fully connected network is shown in figure (6.12). In a large computer network all the computers may be connected to each other through a dedicated or a time-shared bus. This allows the system to bypass a busy or a faulty processor. There is no central control, each processor is allowed to communicate with any other processor independently when required. This network will be costly to implement due to multiple connections. The fully connected network is highly non-modular.

# CHAPTER 7

## A Dedicated Parallel Microprocessor System

### 7.1 Introduction

A number of microprocessors are available now commercially (75), (76). Microprocessors are slow for many applications. However, additional hardware may be employed for better performance e.g. an array processor interfaced with a main frame computer may increase its performance many fold (62), (70). The software on the mainframe computer must be able to detect the degree of parallelism in an algorithm, and generate appropriate code for it.

Arden and Barenbaum (65), and Enslow (66), have suggested that employing several cheap processors in parallel can in certain cases outperform an expensive mainframe computer. With the availability of cheap microprocessors parallel processing technique to implement WFTA was investigated.

Figure (4.3) shows a flow diagram of the 15-point WFTA. Figure (7.1) shows another way of representing it, which illustrates the two dimensional structure in the algorithm. A transform of length N, which can be factorised into n mutually prime factors ($N = r_1 x r_2 x \ldots x r_n$) will have an n dimensional structure. For example in this case N = 15, the two mutually prime factors are 3 and 5. When the 15-point WFTA is implemented on a uni processor system, the 'DO' loop simulates a parallel processor system, calculating the values sequentially rather than

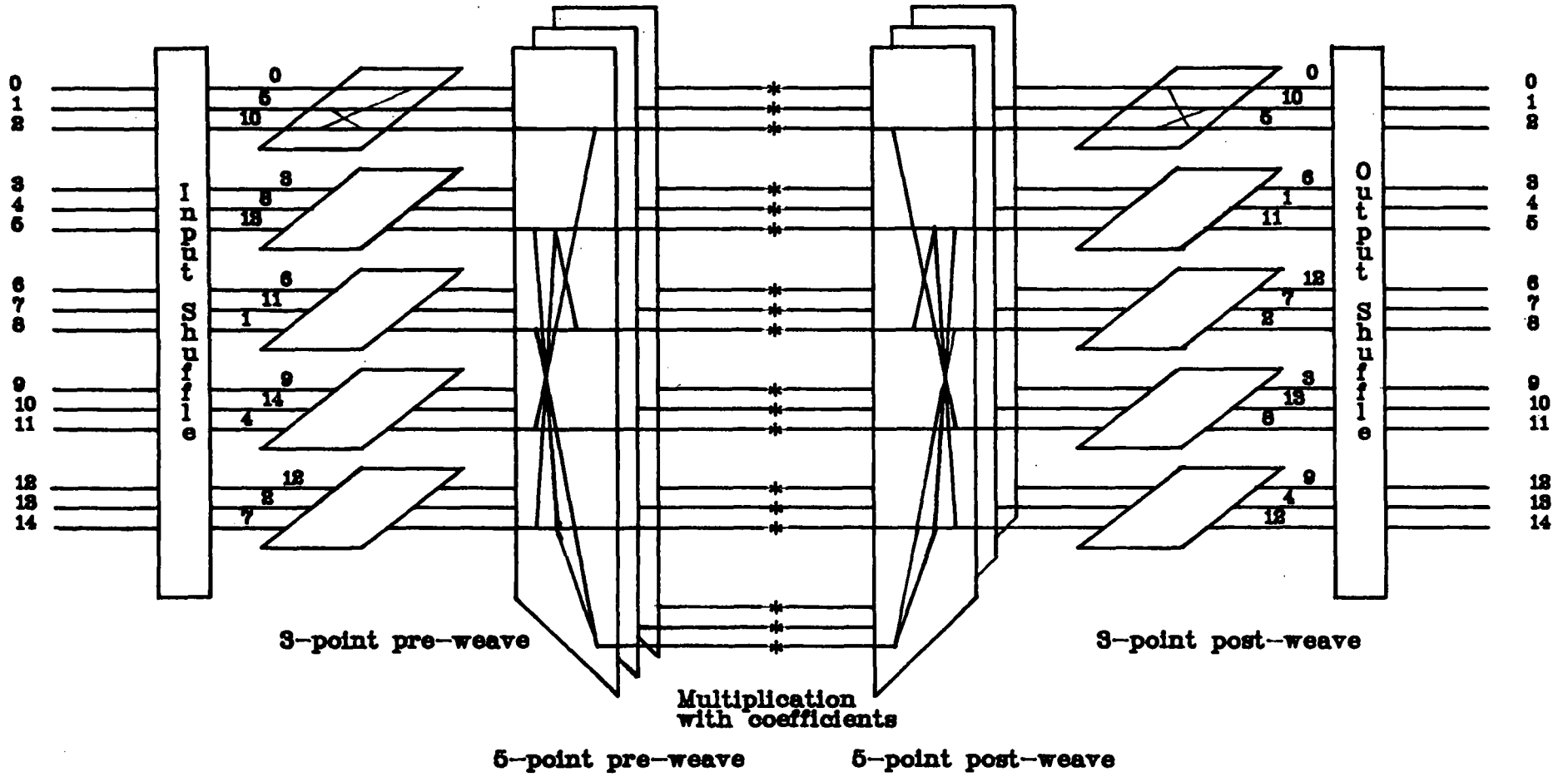



Figure 7.1: 15-Point Winograd Fourier Transform Algorithm (WFTA) showing a two dimensional structure

simultaneously. Coding of a 'DO' loop also hinders efficient program execution. In the case of the WFTA the program coding requires double indexing in the memory to acquire data for arithmetic operations which would load the microprocessor heavily. The consequence is that the microprocessor will spend more time in the indexing and data organisation than actually performing the arithmetic operations.

We are interested in designing a dedicated parallel microprocessor system to implement the 15-point WFTA. Implementation of the 15-point WFTA on a parallel microprocessor system would circumvent some of the problems arising in the uni processor implementation of the algorithm (59), (61). The amount of indexing to be performed by each of the microprocessors is reduced considerably, and fewer data are to be manipulated by individual microprocessors. This frees the microprocessors for more vital tasks. Zohar (60), has suggested the use of address processors to calculate the addresses of the data beforehand, which would effectively increase the systems efficiency.

Attention is now drawn to some essential factors which must be kept in mind for designing a parallel microprocessor system. These factors are, the transform length N, choice of a suitable microprocessor, inter microprocessor communication, systems organisation, cost and power requirements etc.

## 7.2 Choice of a Microprocessor

To investigate the possibility of parallel implementation of the 15-point WFTA requires the selection of a suitable microprocessor. This was done by writing benchmark programs to test the microprocessor's performance in this application. These benchmark programs (for modular arithmetic operations) were written for the following microprocessors, TMS9900, MC6809, Z80 (89), 8X300 (90), COP402 (91) and 6502 (92). Among these the TMS9900 is a 16-bit microprocessor, whereas the MC6809, Z80 and 6502 are 8-bit microprocessors. The 8X300 and COP402 are 8-bit and 4-bit micro-controllers respectively. The MC6800 microprocessor was not included in the above list, because the MC6809 is an enhanced version of the MC6800, and is much faster and more versatile than its predecessor. All these benchmark programs were run on the respective microprocessor systems to test their accuracy, except for the 8X300 and the COP402, which were not available at the time. Appendix-A contains source listings of these benchmark programs, listings for the two micro-controllers are excluded.

Results of these benchmark programs are shown in figures (7.2) to (7.4). Figure (7.5) shows the cost of these microprocessors (1981), which was one of the considerations to obtain a cost effective design (also see tables (3.1) to (3.3)). Comparison of these results show that the MC6809 microprocessor gave an optimum choice. Two of the important features of the MC6809 microprocessor which led to its selection were the availability of an unsigned hardware multiplier and the SYNC

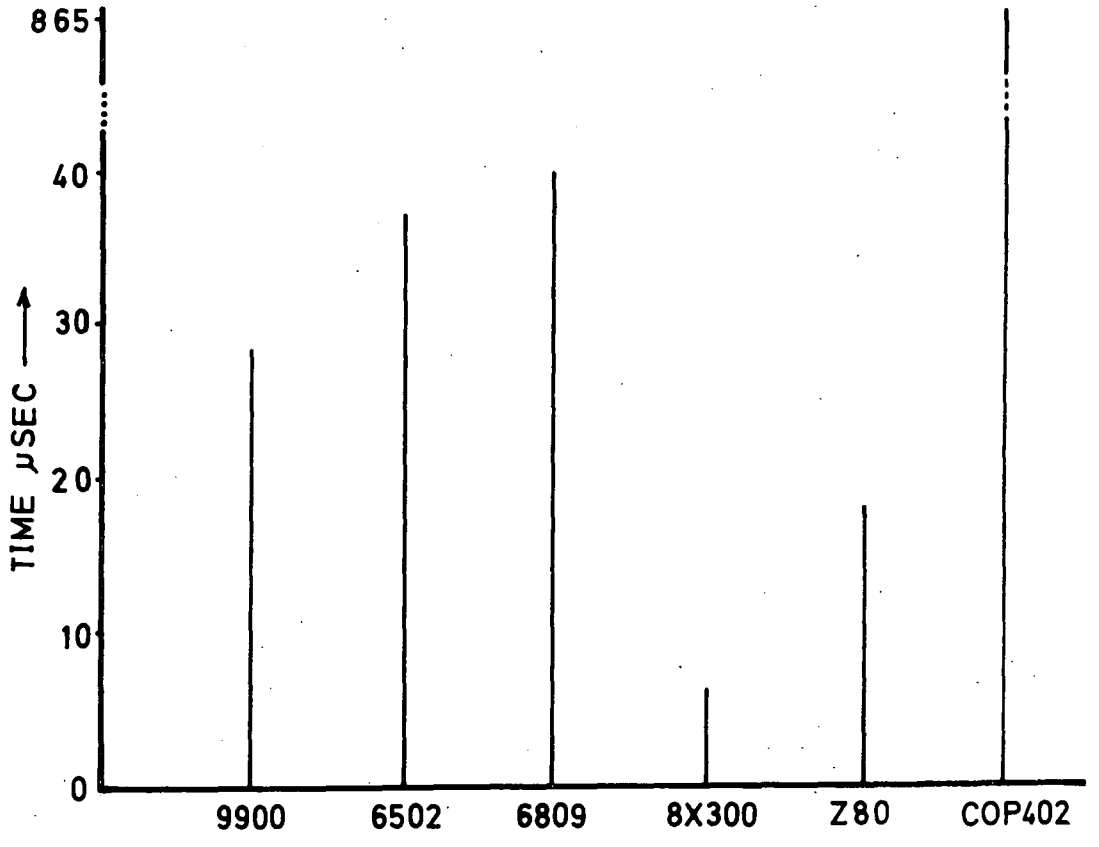



Figure 7.2: Results of the benchmark programs for modular addition.

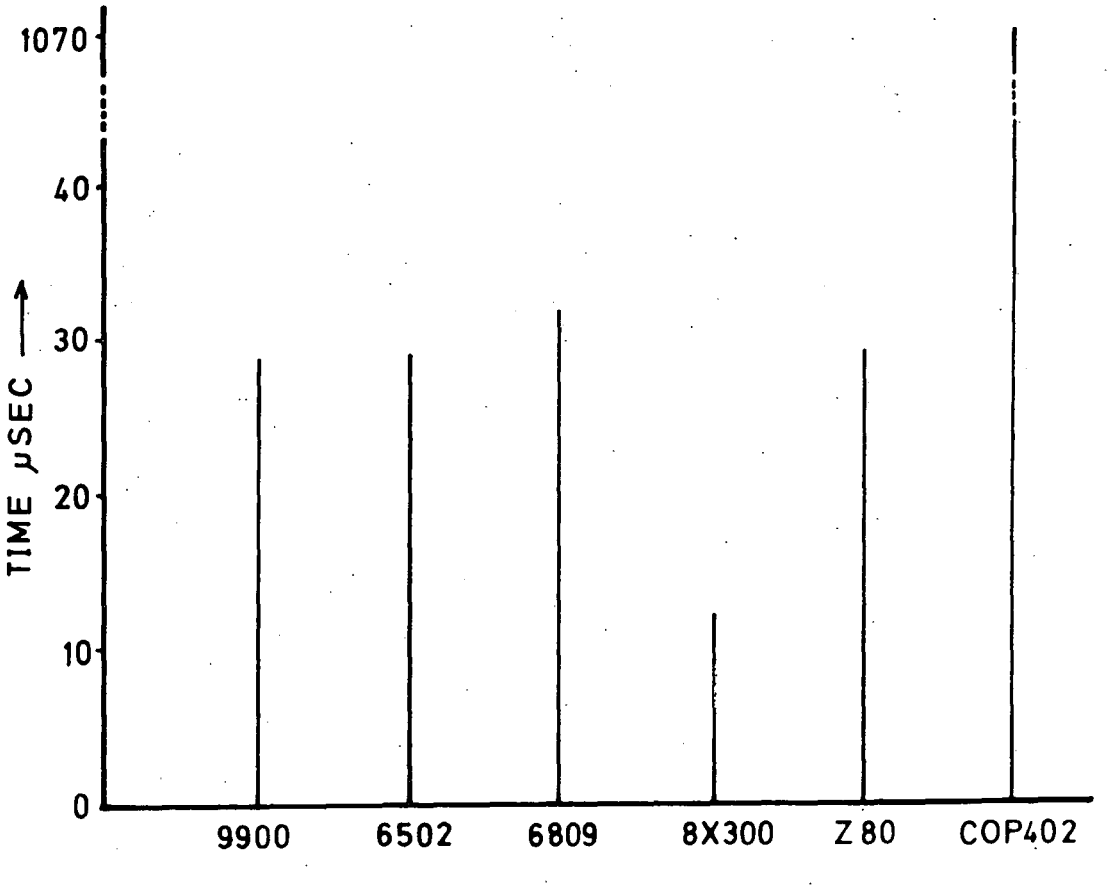



Figure 7.3: Results of the benchmark programs for modular subtraction.

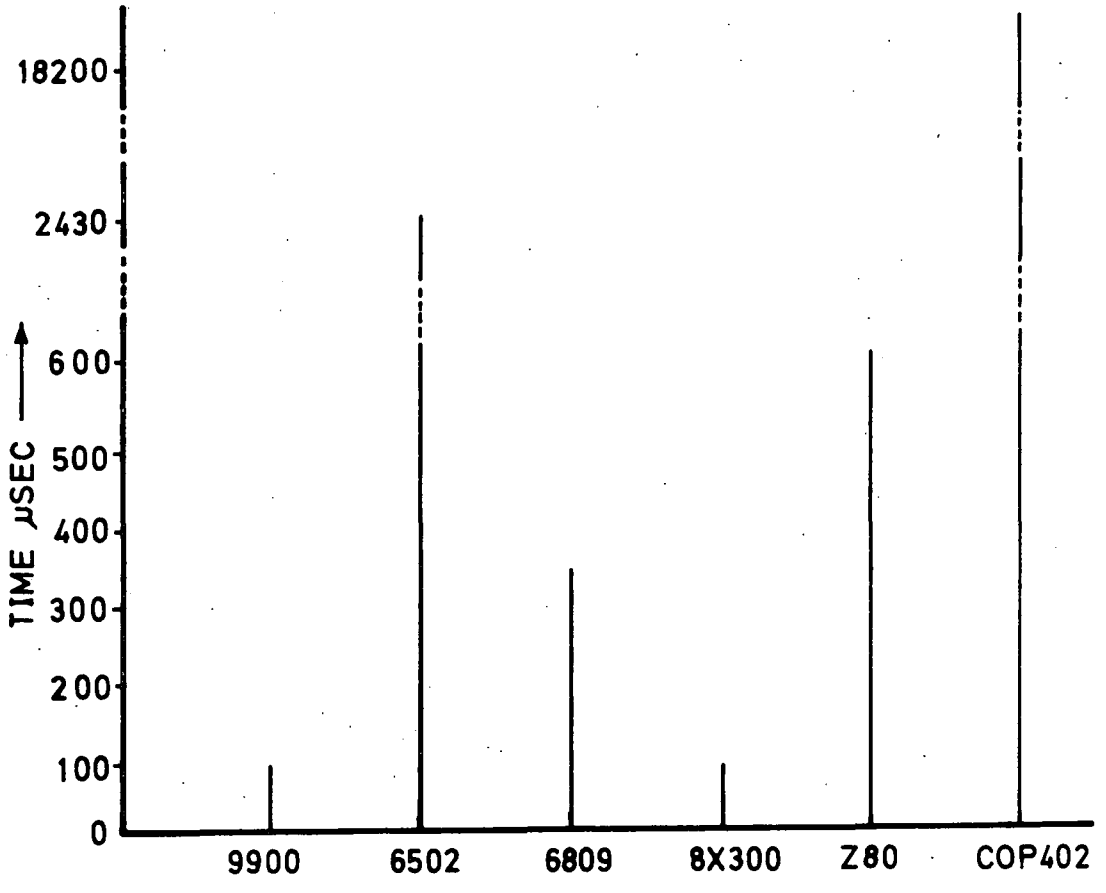



Figure 7.4: Results of the benchmark programs for modular multiplication.

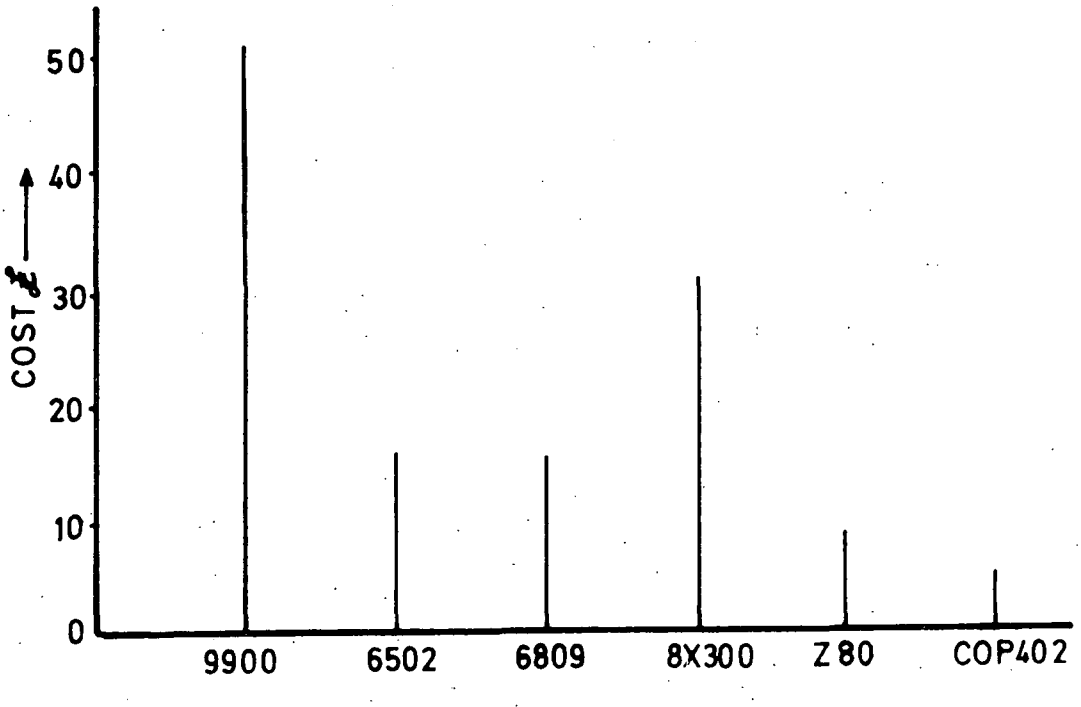



Figure 7.5: Cost of the microprocessors (1981).

instruction. In spite of being an 8-bit microprocessor, its powerful addressing and indexing modes can provide an outstanding performance comparable to the 16-bit microprocessors. Among the rest, only the TMS9900 microprocessor contains an unsigned hardware multiplier.

## 7.3 Architecture of the MC6809 Microprocessor

The Motorola's MC6809 microprocessor is an 8-bit microprocessor, with 16-bit addressing, housed in a 40 pin d.i.l package. Figure (7.6) shows a block diagram of the CPU architecture (78), (79).

It consists of two general purpose 8-bit registers A and B, often called the accumulators. These registers are mostly used for arithmetic purposes. The repertoire of the microprocessor contains signed and unsigned 8-bit and 16-bit arithmetic operations. The accumulators A and B may be concatenated together to form a 16-bit accumulator D, thus allowing 16-bit arithmetic. An 8-bit Condition Code register (CC) provides information about the current machine status.

Two 16-bit index registers X and Y are used in the indexed mode of addressing. These registers are quite useful when sequential data access to and from the memory is required. However, an offset can be specified in the instruction, then the address in an index register behaves as a base address. The accumulators can also be used to hold this offset.

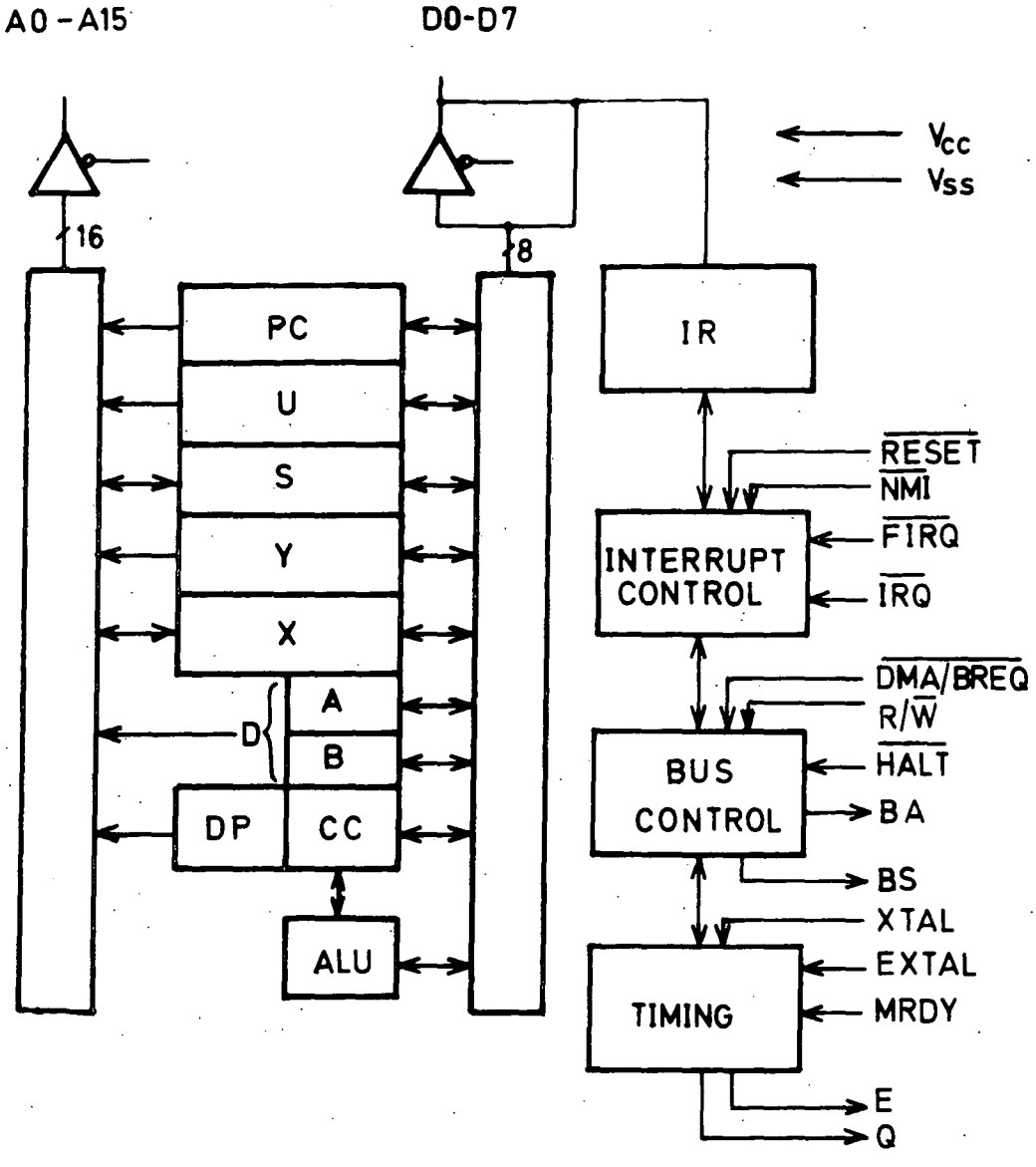



Figure 7.6: MC6809 CPU block diagram.

| MPU STATE | | |
|---|---|---|
| BA | BS | |
| 0 | 0 | NORMAL RUNNING |
| 0 | 1 | INTERRUPT ACK. |
| 1 | 0 | SYNC ACK. |
| 1 | 1 | HALT OR BUS GRANT |

Table 7.1: MC6809 CPU state.

There are two 16-bit stack pointers called the hardware Stack pointer S, and the User stack pointer U. These stack pointers can be used with the same addressing modes as the index registers X and Y. These registers work as pushdown stack pointers, and are accessible to the programmer. When subroutines or interrupt routines are to be executed, the microprocessor automatically utilises the address in the stack pointer S for saving the entire machine state in the memory. The stack pointers U and S may be used as pointers for the pushdown stack thus supporting pull and push instructions. This pushdown stack allows to pass arguments to and from the main program to the subroutines, interrupt routines etc.

A 16-bit Program Counter (PC) allows access to 64K bytes of memory. The program counter contains the address of the next sequential or logical instruction to be executed. An 8-bit Direct Page (DP) register is available to enhance the direct addressing mode. The contents of this register serve as high order 8-bits (A8-A15) during the direct addressing. The DP register is cleared when the microprocessor is reset. This register allows 8-bit relative addressing within the page, whose base address is in the DP register. The direct addressing mode requires fewer program bytes and executes much faster than the absolute addressing mode.

The microprocessor also contains an onchip oscillator, which is accessed through two input pins. This oscillator may be operated by an external crystal of frequency 4f (where f is the bus frequency, typically f = 1 MHz). Alternately an external

(TTL) clock source of 4f may be used to operate the microprocessor. The latter arrangement is useful in systems where synchronous processing is required e.g. in multi processor or parallel processor systems. Two output clock signals E and Q (1 MHz), are used for external timings. Addresses are valid on the leading edge of Q, and data are latched on the falling edge of E.

A low level on the RESET input forces the microprocessor into a known state. A low level on the DMA/BREQ input forces the data and address buses into high impedance state, so as to permit a direct memory access. A low level input on the HALT line halts the microprocessor indefinitely after the end of current instruction without loss of data. A MRDY input allows the microprocessor to access slow memories, by stretching its clock signals. However, the clock signals may not be stretched beyond 10 microseconds without loss of data. A R/W line indicates a Read (high) or a Write (low) cycle. Two output signals BA (Bus Available) and BS (Bus Status) gives information about the current machine status as shown in table (7.1).

### 7.3.1 Hardware and Software Interrupts

Three levels of hardware interrupts are available, and are prioritised in the following order, NMI (Non Maskable Interrupt), FIRQ (Fast Interrupt ReQuest), and IRQ (Interrupt ReQuest).

The NMI is a negative edge triggered interrupt and cannot be disabled through software. When this interrupt occurs, the entire machine state is saved on the hardware stack. This

condition is indicated by setting the E flag in the condition code register. After a reset, the NMI will not be recognised until the first program load of the hardware stack pointer S.

Both the FIRQ and the IRQ are level triggered interrupts and are maskable, i.e. these interrupts can be disabled or enabled through the software. If the F or the I bit in the condition code register is set to logic 1, then the respective interrupt is disabled. Otherwise it is enabled. The FIRQ is the fast interrupt in the sense that, unlike NMI and IRQ it does not save the entire machine state, but saves only the condition code register and the program counter on the hardware stack. The E bit in the condition code register remains cleared. The IRQ interrupt works in a similar fashion as the NMI interrupt, except that it is maskable.

Three levels of software interrupts are also available, and are useful for debugging the system and for software development. Decoding of the low order 4-bits on the address bus determines which level of interrupt had occured.

### 7.3.2 Microprocessor Synchronisation

In a parallel processor system a single out of step processor can produce chaotic results. Synchronisation can be achieved by handshaking at hardware or software level. The handshaking allows data exchange between two or more processors without loss of information.

The MC6809 microprocessor is provided with a SYNC instruction which may be used to synchronise the microprocessor to an external event. When the microprocessor executes the SYNC instruction, it stops processing the instructions and waits for an external interrupt. Two output pins BA . BS = 1 indicate the SYNC acknowledge, where '.' represents a logical AND operation (see table 7.1). If the pending interrupt is a nonmaskable (NMI) or a maskable interrupt (FIRQ or IRQ) with its mask bits (F or I) clear, then after receiving the external interrupt the microprocessor will clear the sync state and will execute the appropriate interrupt routine. However, if the pending interrupt is maskable and it is disabled, then the microprocessor will simply clear the sync state and resume normal operation. This instruction is ideally suited for the situations where the expected input data are occuring randomly, and the microprocessor cannot process further data without it. This data can be from another microprocessor or from some other source.

The use of SYNC instruction is equivalent to a wait loop. An advantage of using the SYNC instruction is that it is faster than the wait loop, since the microprocessor will proceed further as soon as it receives an interrupt. However, in the case of a wait loop a small delay may be introduced before the processor can proceed further.

## 7.4 Inter Microprocessor Communication

In a multi processor or a parallel processor system it may or may not be a requirement for the processors to communicate

with each other at all. However, if a processor requires data from another processor during the task execution, then some form of inter processor communication is required. The method of data exchange will depend upon whether the system is loosely or tightly coupled.

To investigate a principle for inter microprocessor communication a simple example is presented. Consider a system with two general purpose processors P1 and P2 (see figure 7.7). Each of the processor has its own local program memory, and some arrangement for decoding the address and generating the appropriate read/write signals. Consider two latches L1 and L2 with tristate outputs, these latches are connected to the processors such that, P1 can only write into L1 and P2 can only write into L2. Furthermore, P1 can only read the contents of L2 and P2 can only read the contents of L1. In other words L1 is a write only and L2 is a read only latch for P1, and L2 is a write only and L1 is a read only latch for P2. This arrangement forms a loosely coupled multi processor system, and the associated latches may be visualised as I/O ports. These latches are connected through dedicated parallel data buses, with two associated control signals. These two control signals are the output enable (OE) and the clock (CLK) signals.

The two processors exchange data with each other through the communication latches in the following manner. When required, P1 writes data into L1 and P2 writes into L2. The processors are then synchronised with each other at this instant, and then the processors read their respective read only latches (88).

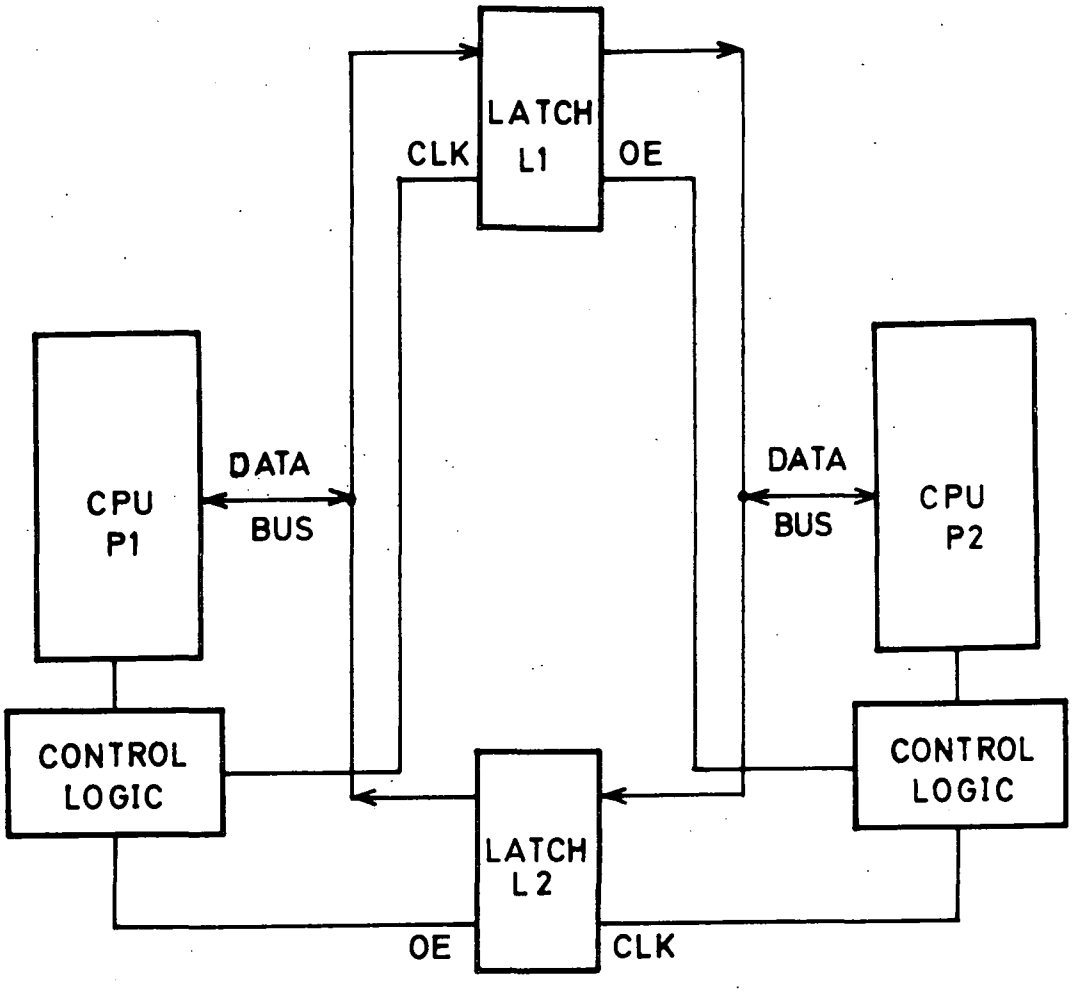



Figure 7.7: A two microprocessor system.

## 7.5 Dual Microprocessor System

Figure (7.8) shows a block diagram of a practical circuit based on the idea discussed in the previous section. This system contains two MC6809 microprocessors P1 and P2. A TMS9900 microprocessor system serves as a host or master to control the two slaves P1 and P2. Each of the microprocessors has its own local program memory and no other microprocessor can access it. A common single phase clock is used to operate the two slaves, which is separate from the master's clock. The microprocessors are located physically very close to each other, and the interface between the master and slaves is through dedicated 16-bit latches with tristate outputs. The master's side consists of a 16-bit data bus, while the slave's side consists of an 8-bit data bus.

In addition to the communication latches L1 and L2, each of the two slave microprocessors have associated with them two additional latches, namely IN1, OUT1 and IN2, OUT2 respectively. IN1 and IN2 serve as the input buffer memory i.e. data to be transferred to the slaves by the master are held in these latches. Results calculated by the slaves are stored in the OUT1 and OUT2 latches, which are to be read by the master. The working of these latches are similar to L1 and L2 as described before, except that these latches are used to exchange data with the master.

The HALT and the RESET inputs of the slave microprocessors are connected to the output port of the master. The logic levels on this port can be changed individually through the software.

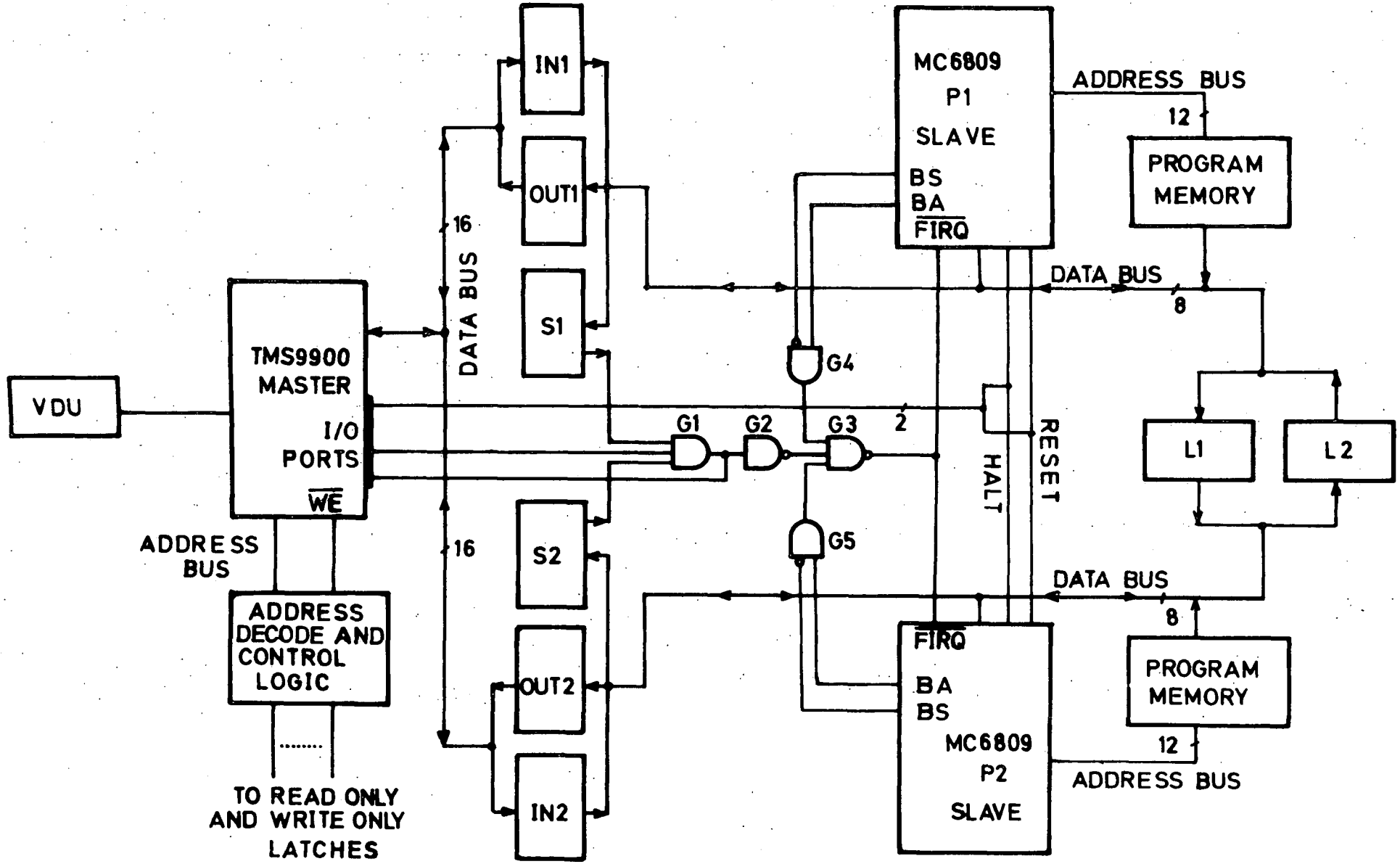



Figure 7.8: TMS9900 microprocessor controlling the two slaves (MC6809).

Initially the master resets and then halts the slaves, until it has transferred data into the input latches of the slaves.

Another important feature in this system is the synchronisation between the two slaves. This is achieved by using the SYNC instruction and the FIRQ interrupt input, with the F bit in the CC register set to logic 1. When the HALT input goes high the slaves read the input latches and transfer these data values into their appropriate communication latches, and then execute the SYNC instruction. The sync acknowledge signal from the two processors are ANDed (G3) together and inverted to generate interrupt to themselves. This condition indicates that valid data are available in the latches L1 and L2. After receiving the interrupt the slaves read their appropriate read only latches, and perform the desired operation. One of the slaves was chosen to perform modular addition and the other modular subtraction.

Some form of protocol is also necessary between the master and the slave microprocessors to facilitate synchronisation and communication. For this purpose an 8-bit status (STATUS) latch is also associated with each of the slaves S1 and S2, only the least significant bit is used. The output of the status latch determines the system status. For example a logic 0 at the output of the status latch indicates that the slave is busy executing its program. While a logic 1 indicates termination of the task (see figure 7.8), the slaves execute the SYNC instruction after setting status to logic 1. The output of the two status latches are permanently enabled and are ANDed (G1)

together to generate the system status signal. Another 1-bit signal which is being ANDed in G1 is obtained from the output port of the TMS9900 microprocessor. This bit is called the status control bit (SCB). When this bit is low the status latch output has no effect on G3, as G1 is disabled. When the master desires to read the output latches, it sets the status control bit to logic 1, and continuously monitors for the output of G1 to go high. When the system status signal goes high, the master reads the output latches. The slaves execute the SYNC instruction after outputting the data, hence the slaves will remain in that state until the status control bit goes low again. This is done by the master after transferring new values into the input latches, which forces the output of G3 low, thus generating an interrupt to the slaves, the slaves repeat the same cycle again, by first clearing the status latch.

This loosely coupled multi processor system was designed just to test its performance and the principle of slave-slave and master-slave communication. Additional software on the master checks that the results calculated by the slaves are correct.

### 7.5.1 Merits and Demerits

In general two microprocessors cannot communicate with each other in real time, without one of them waiting for the other to send data. But if some intermediate buffer memory is used, then the source microprocessor can transfer the data into this buffer memory, and the destination microprocessor can read this data at leisure. If we are dealing with a single or a double byte

buffer, then care must be taken that the source does not overwrite this data before the destination microprocessor had a chance to read it (64), (68). Another situation might also arise, in which the destination microprocessor keeps reading the same data without realising that the data have not been updated since it was last read. These conditions can be circumvented by using a single bit flag which indicates whether the data had been read or updated in the buffer or not.

Previously we have seen that the latch was used as a communicating medium between the two microprocessors. The input of the latch is connected directly to the data bus of the source microprocessor. The output of these (tristate) latches can be connected directly to the data bus of the destination microprocessor. The control signal i.e. the clock (CLK) and the output enable (OE) may be appropriately generated. This means that each side of the latch consists of ten lines in all, i.e. an 8-bit data bus and two control signals for either the output enable or the clock signal (since 16-bit data is being transmitted through a unidirectional dedicated data bus). We are investigating a method for inter microprocessor communication to be used for the implementation of the 15-point WFTA. We will see later that in the parallel microprocessor system (for the parallel implementation of the WFTA) only one 16-bit value is exchanged between two microprocessors at any instant on a particular bus. The use of latches thus reduce the circuit complexity considerably, but at the expense of increased chip count, cost and power consumption.

Alternately a common memory (RAM) can be employed for inter microprocessor communication. Although it provides more storage and may be cheaper, it also increases the circuit complexity considerably. The major problem in a shared memory system is to prevent memory conflict or memory contention. An attempt by two or more microprocessors to access common memory is called memory contention. The shared and the local address and data buses have to be multiplexed (67). The throughput is reduced considerably when all the processors wish to access the common memory simultaneously. Hoffner and Smith (68), have suggested a method of preventing memory contention in a system with two MC6809 microprocessors by operating them opposite phases of a common clock. The number of microprocessors connected in this manner is limited to two.

## 7.6 Design and Implementation of the Dedicated Parallel Microprocessor System

The dual microprocessor system worked quite satisfactorily. The method adopted for inter microprocessor communication through latches seemed quite suitable for the parallel microprocessor system to implement a 15-point WFTA. Each of these latches would be connected through dedicated unidirectional 8-bit data buses. All the data exchange among the microprocessors can then take place simultaneously, hence the system should provide a very high efficiency and throughput.

Close examination of figure (4.3) reveals that the implementation of the 15-point WFTA algorithm consists of following steps.

1. Input shuffle or reordering

2. Five 3-point preweave or premultiply adds

3. Three 5-point preweave or premultiply adds

4. Eighteen modular multiplications with precalculated coefficients

5. Three 5-point postweave or postmultiply adds

6. Five 3-point postweave or postmultiply adds

7. Output shuffle or reordering.

It may be noted here that the 5-point WFTA requires six modular multiplications which requires extra storage. Hence the total number of modular multiplications in the 15-point WFTA is eighteen. Since modular multiplication is the most time consuming operation, the parallel microprocessor system was designed such that all the microprocessors share the load equally during the modular multiplication. This requires 18 microprocessors in the complete system.

### 7.6.1 System Architecture

Figure (7.9) shows a block diagram of the dedicated parallel microprocessor system. The microprocessors are interconnected to form a two dimensional array with three rows and six columns. The five 3-point transforms are performed along the columns. The microprocessors numbered 16, 17, and 18 do not take an active

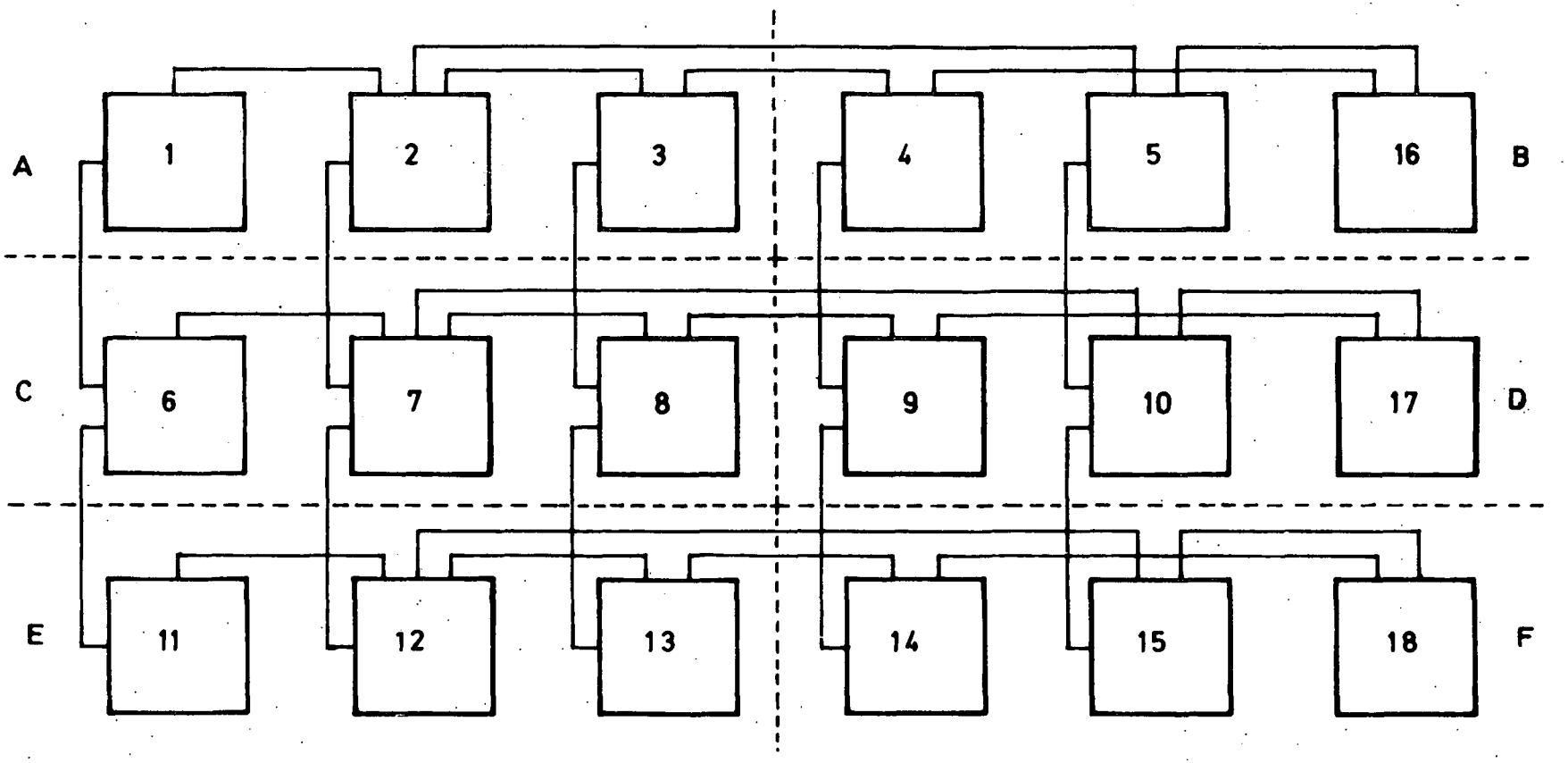



Figure 7.9: Block diagram of the parallel microprocessor system (the control microprocessor is not shown).

part at this stage hence no connection exists between them along the column. For the three 5-point transforms the microprocessors are active along the rows. Comparison shows that each '.' (column wise) in figure (4.3) corresponds to a box which is a microprocessor with associated hardware in figure (7.9). Each of the connecting lines along the rows and columns consists of two 8-bit dedicated data buses with two associated control signals, to facilitate bidirectional communication between the two microprocessors. All the microprocessors in the system are driven from a common single phase clock source of 4 MHz. Each of the slave microprocessors generate their own local timing signals.

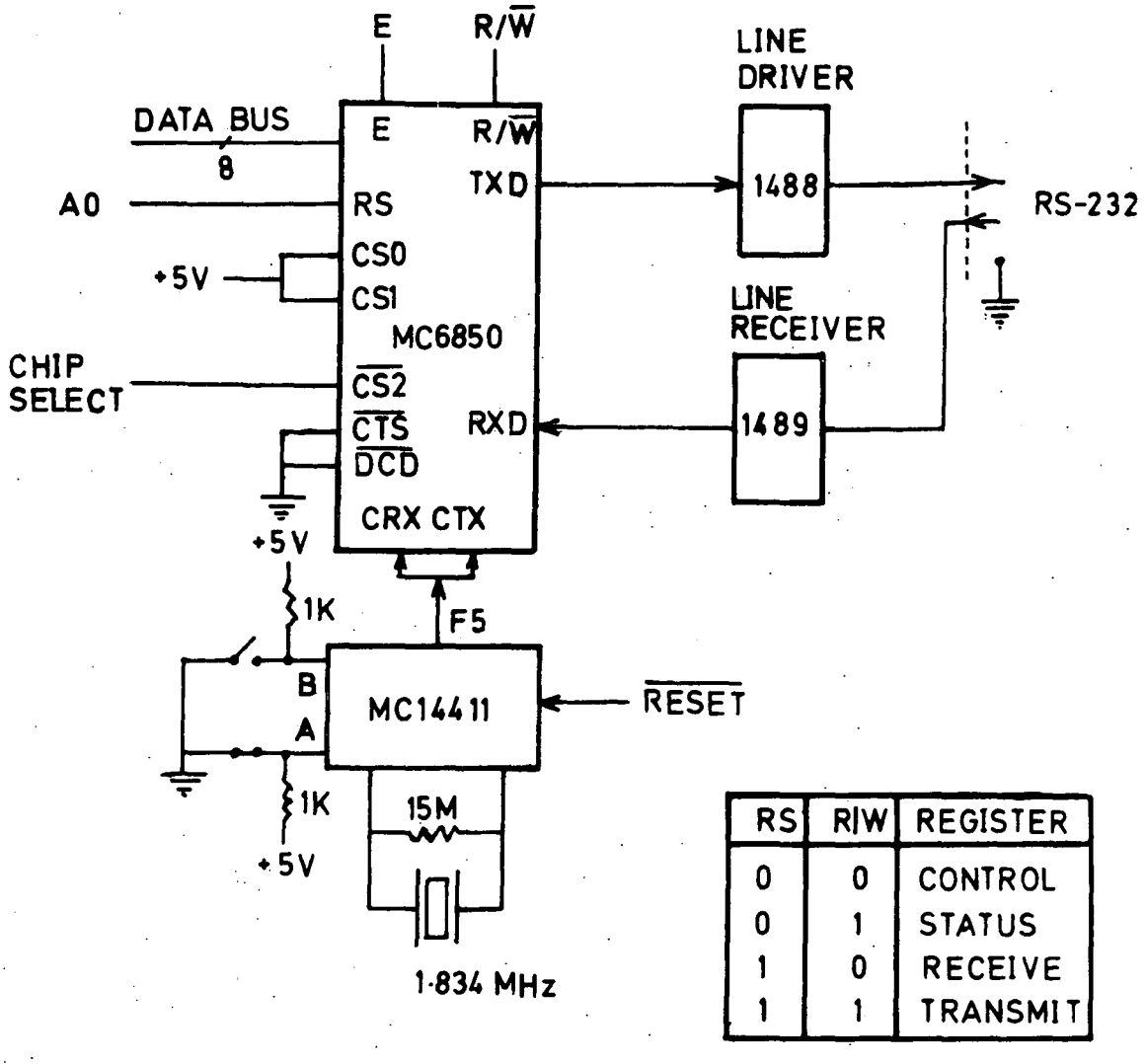
The microprocessors in the system are partially connected, i.e. there are no redundant connections. This system basically forms a loosely coupled dedicated MIMD machine. The prototype system was assembled on seven standard plugin 6U eurocards, using wire wrapping techniques. The dotted line in the figure (7.9) shows how these microprocessors are distributed among the six boards labelled A to F. The seventh board in the system consists a control or a master microprocessor, with associated circuitry.

### 7.6.2 Design of the Control Microprocessor

The slave microprocessors are not capable of communicating directly with the outside world i.e. with a VDU or any other real time device. Hence an extra dedicated microprocessor is employed to serve as a host or a master microprocessor (not shown in figure 7.9). This brings the total number of microprocessors in the system to nineteen. The control microprocessor not only

serves as a controller for the slaves, but also provides an interface to the outside world. The control microprocessor has an RS-232 serial interface with the VDU to provide access to the system. Figure (7.10) shows a circuit arrangement for the serial interface using Motorola's MC6850 ACIA (Asynchronous Communications Interface Adapter). A baud rate generator Motorola MC14411 is used to generate the receive/transmit rate clock for the ACIA (82), (83), (84), (85).

The parallel microprocessor system appears to the master as a black box, the only part accessible to the master are the input and the output latches associated with the slaves. This black box appears as an intelligent peripheral to the control microprocessor. The master microprocessor transfers data to the input latches and reads the transformed values from the output latches. For demonstration purposes the master then reads the output latches and stores these values into its memory and displays on the VDU, or oscilloscope via a D/A converter. The master microprocessor does not interfere in the data exchange among the slaves, and in fact it is unaware of that. All the I/O data has to pass through the master. For large N, this may become a limiting factor, and may degrade the system's performance. For example 178 microseconds are required to transfer fifteen 16-bit data to or from the slave microprocessors. Alternative arrangement can be made to transfer the data directly into/from the input and output latches, which would increase the throughput.




| RS | R/W | REGISTER |
|---|---|---|
| 0 | 0 | CONTROL |
| 0 | 1 | STATUS |
| 1 | 0 | RECEIVE |
| 1 | 1 | TRANSMIT |

Figure 7.10: ACIA interface.

Figure (7.11) shows circuitry associated with the control microprocessor. The control microprocessor has it own program memory of 2K x 8-bits (2716), and 1K x 8-bit (2 x 2114) of local RAM. A number of address decoders (SN74LS154) are required to access all the input and output latches. A bidirectional bus transceiver (SN74LS245) is used to drive all these latches which reduces loading on the data bus of the microprocessor. However, the local RAM and ROM are connected directly to the data bus of the microprocessor.

An 8-bit write only control latch (CONTRL) is associated with the master (see figure 7.12). The output of the control latch is permanently enabled and the low order 5-bits are used for control purposes. A location STATUS in the RAM keeps a record of the contents of the control latch. These control signals are as allocated as follows.

Bit 0 : master RESET for the slaves

Bit 1 : HALT for the slaves

Bit 2 : RESET for the baud rate generator

Bit 3 : status control bit (SCB)

Bit 4 : chip enable for the A/D converter

Bit 5 : signal to slaves to perform forward or inverse transform

These bits can be individually set to a logic 1 or reset to logic 0 through software using logical bit instructions. The status control bit (SCB) is used to detect the condition of the complete cycle of the transform (see figures 7.12, 7.13). When the master desires to read the output latches, it sets the status

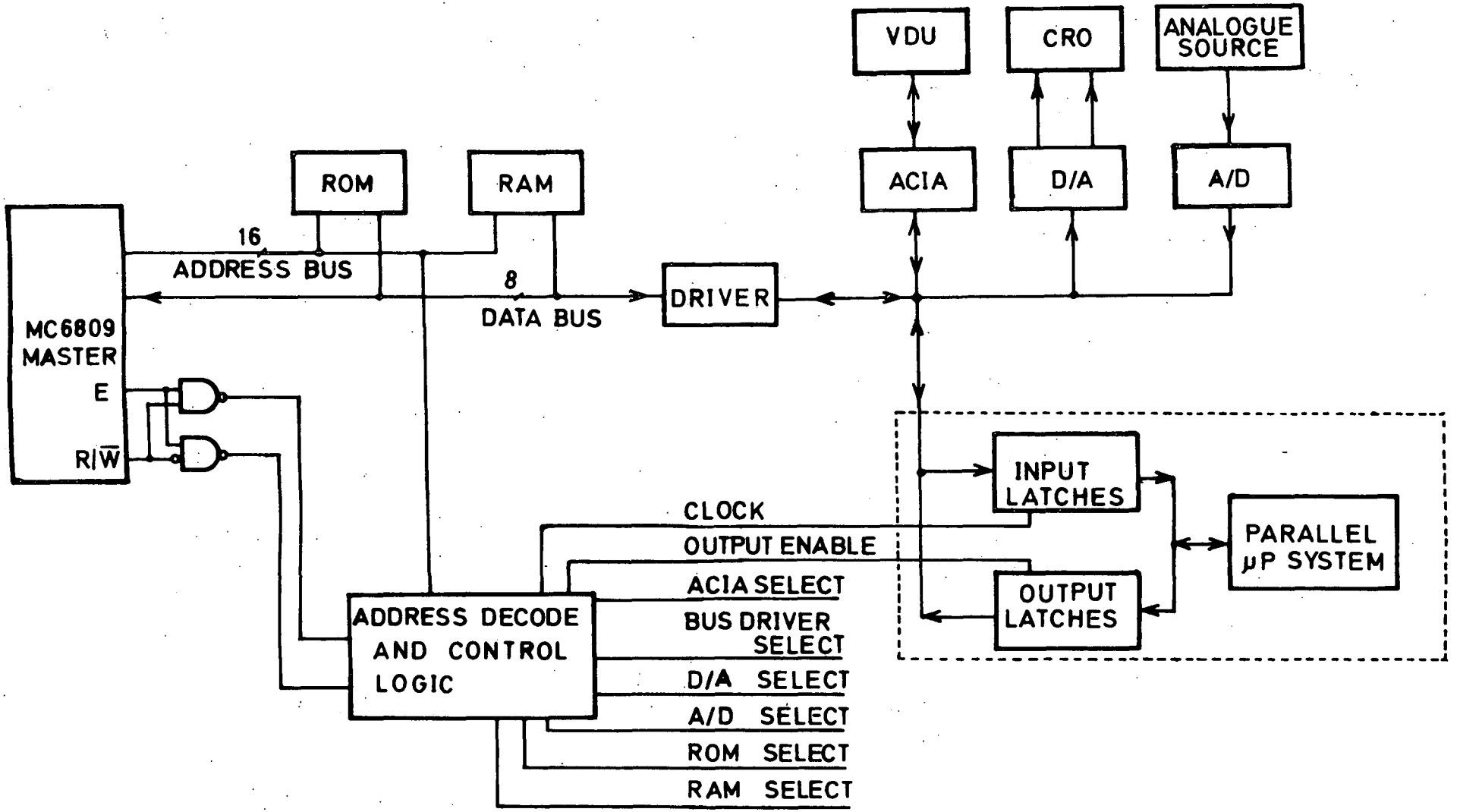



Figure 7.11: Complete parallel microprocessor system showing the master and the slaves.

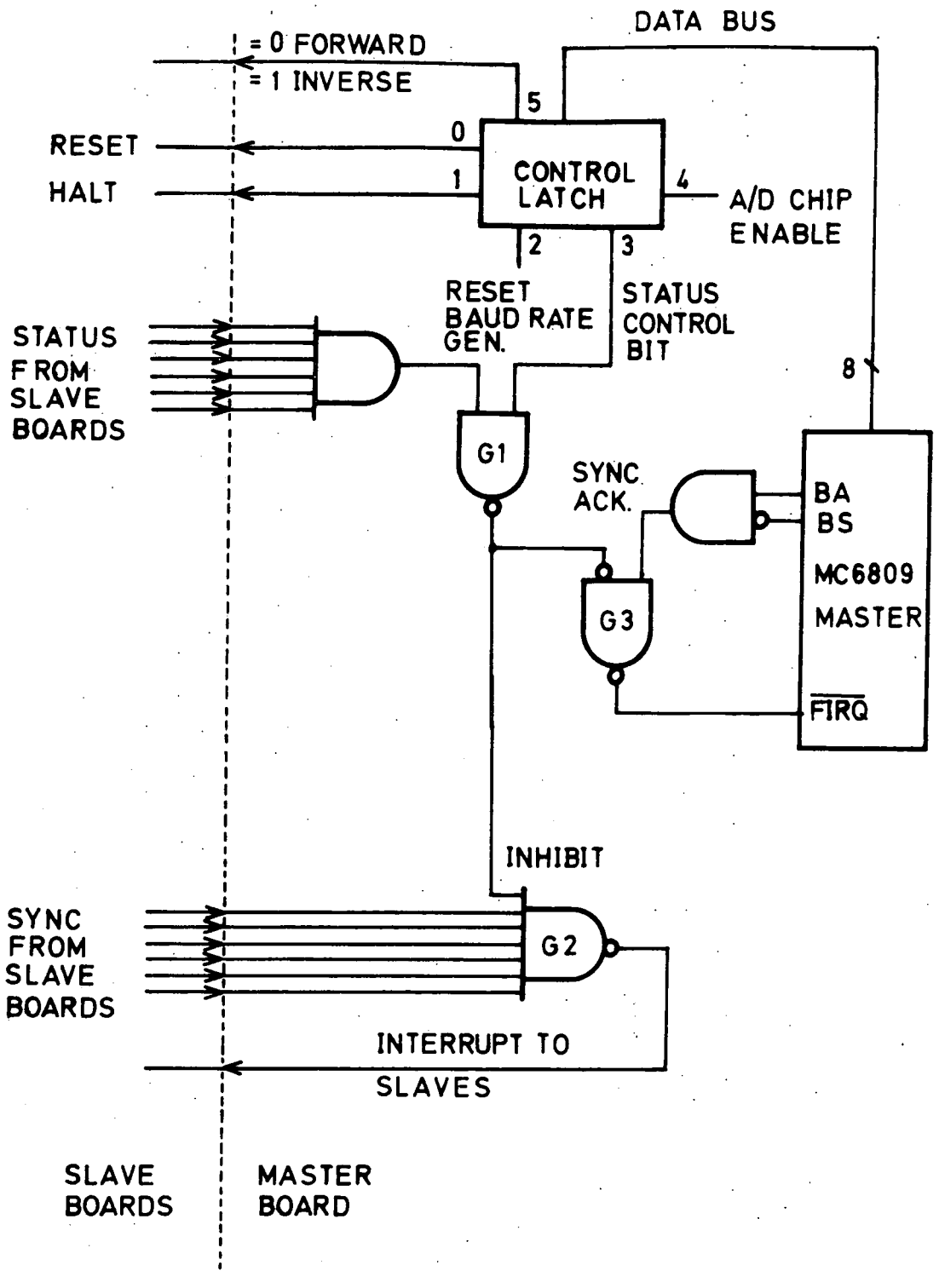



Figure 7.12: Master microprocessor with associated circuitry.

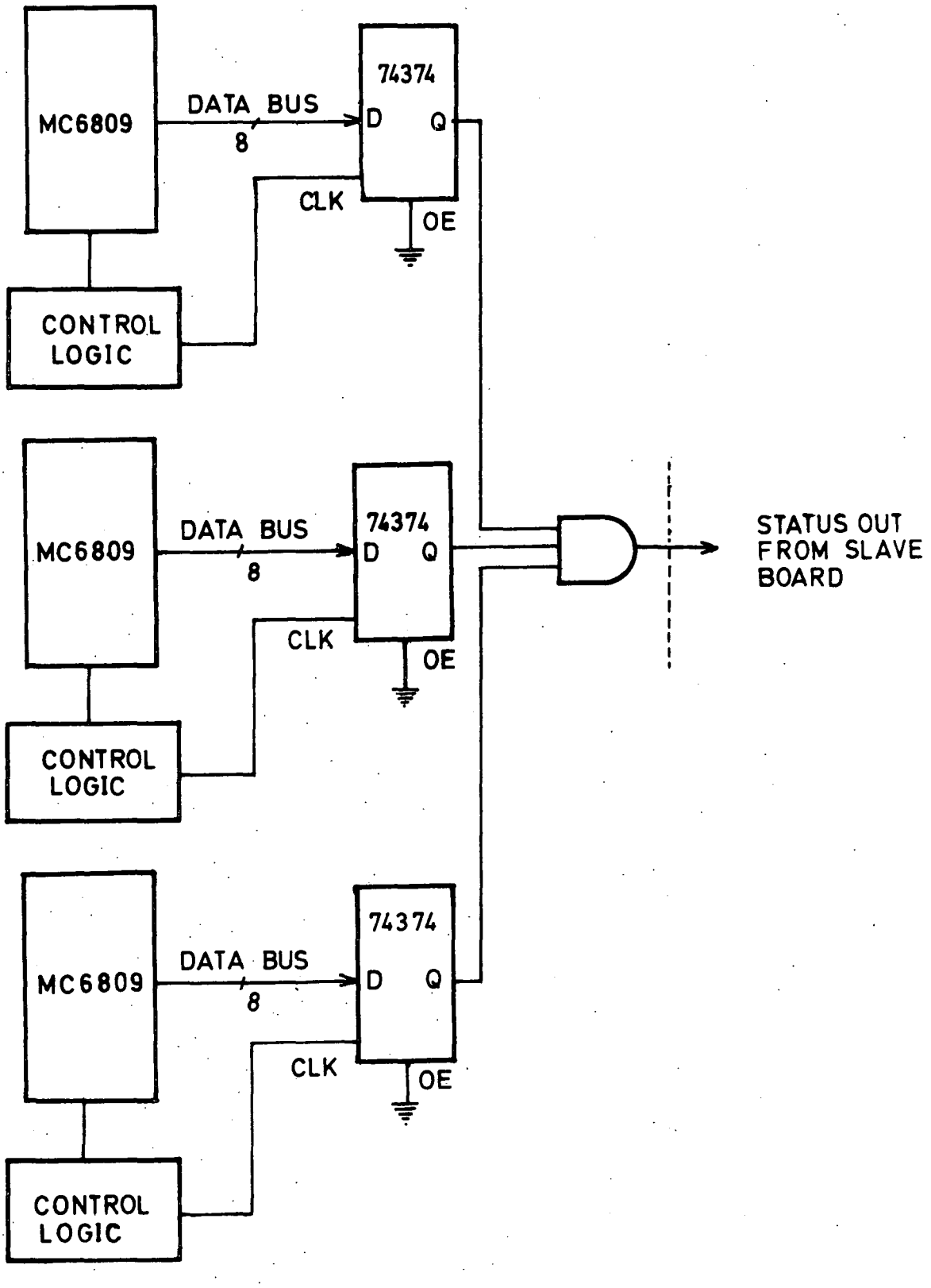



Figure 7.13: Arrangement for generating STATUS signal from each slave board.

control bit (SCB) to a logic 1 and executes the SYNC instruction and waits for the slaves to complete the transform. When the slave microprocessors finish the transform cycle, they set their respective status latches to a logic 1, and execute the SYNC instruction. At this time the output of the gate G1 goes low disabling G2, simultaneously generating an interrupt signal to the master through G3. The master then resumes normal operation and reads the output latches. However, as long as the status control bit remains high, it prevents the interrupt signal from reaching the slaves. After reading the output latches the master clears the status control bit. This forces the output of G1 high, enabling G2 and consequently generating an interrupt to the slave microprocessors. The slaves then start the next cycle of the transform.

### 7.6.3 Software of the Control Microprocessor

To facilitate the development of the software, the control microprocessor provides an interactive interface with the parallel microprocessor system (see figure 7.11). This allows manual insertion of data into the parallel microprocessor system.

When the power is switched on, the powerup circuitry resets the master microprocessor. The master then resets the baud rate generator and the slaves, and halts the slaves. It then resets and initialises the ACIA for the data receive/transmit data format and the baud rate. The halt state of the slaves is then cleared. Source listing of the monitor program is included in appendix-D.

For test purposes a 15-point WFTA verify routine is stored in a separate ROM (see appendix-D). The control microprocessor executes the 15-point WFTA on the same input data as the slaves, and verifies the transformed values obtained from the slave microprocessors. The control microprocessor displays an error message on the VDU, if the two results do not tally, and prints these values. A 15-point WFTA was also implemented in FORTRAN on a main frame computer to verify these results.

### 7.6.4 Design of a Typical Slave Microprocessor

A typical circuit arrangement for the slave microprocessor interfaced with local program memory 2K x 8-bits (2716), local RAM 1K x 8-bits (2 x 2114) is shown in figure (7.14). However, microprocessors numbered 16, 17 and 18 have a slight variation in their circuit arrangement which is shown in figure (7.15). Each of the six eurocards contains three such circuits. Each of the slave microprocessors has associated with it input (INPUT), output (OUTPUT), and status (STATUS) latches, except for the slaves numbered 16, 17, and 18. In addition a number of communication latches are also associated with each of them. The number of latches for a particular microprocessor depends upon how many microprocessors it is communicating with. All these latches are 16-bit (2 x SN74LS374) latches, with tristate outputs, except the status latch which is an 8-bit latch. The clock and the output enable signals are generated using a 4-line to 16-line decoder (SN74LS154), and gating it appropriately with E and R/W. All the latches are driven by the bidirectional bus

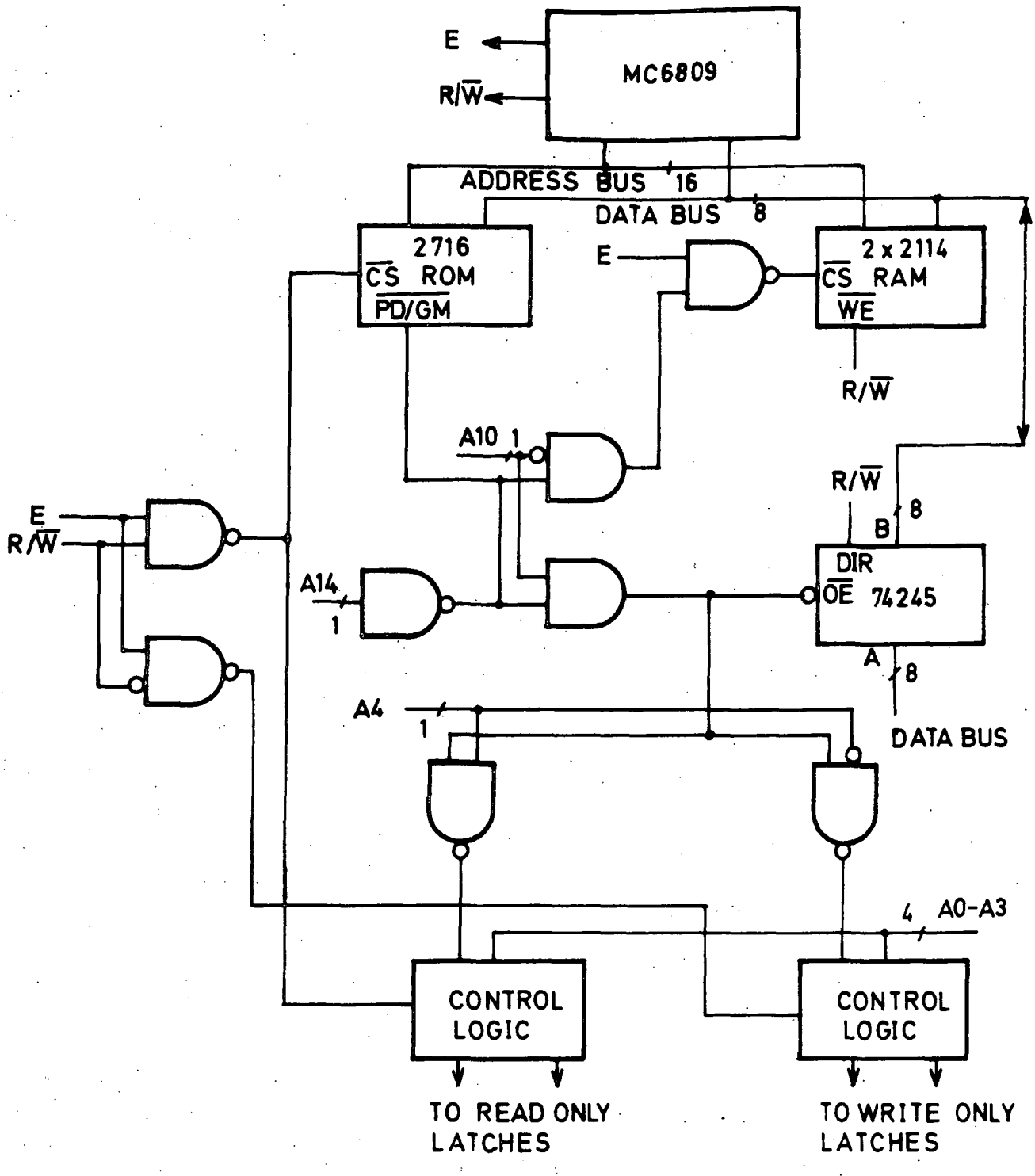



Figure 7.14: Typical slave microprocessor (1 to 15) with associated hardware.

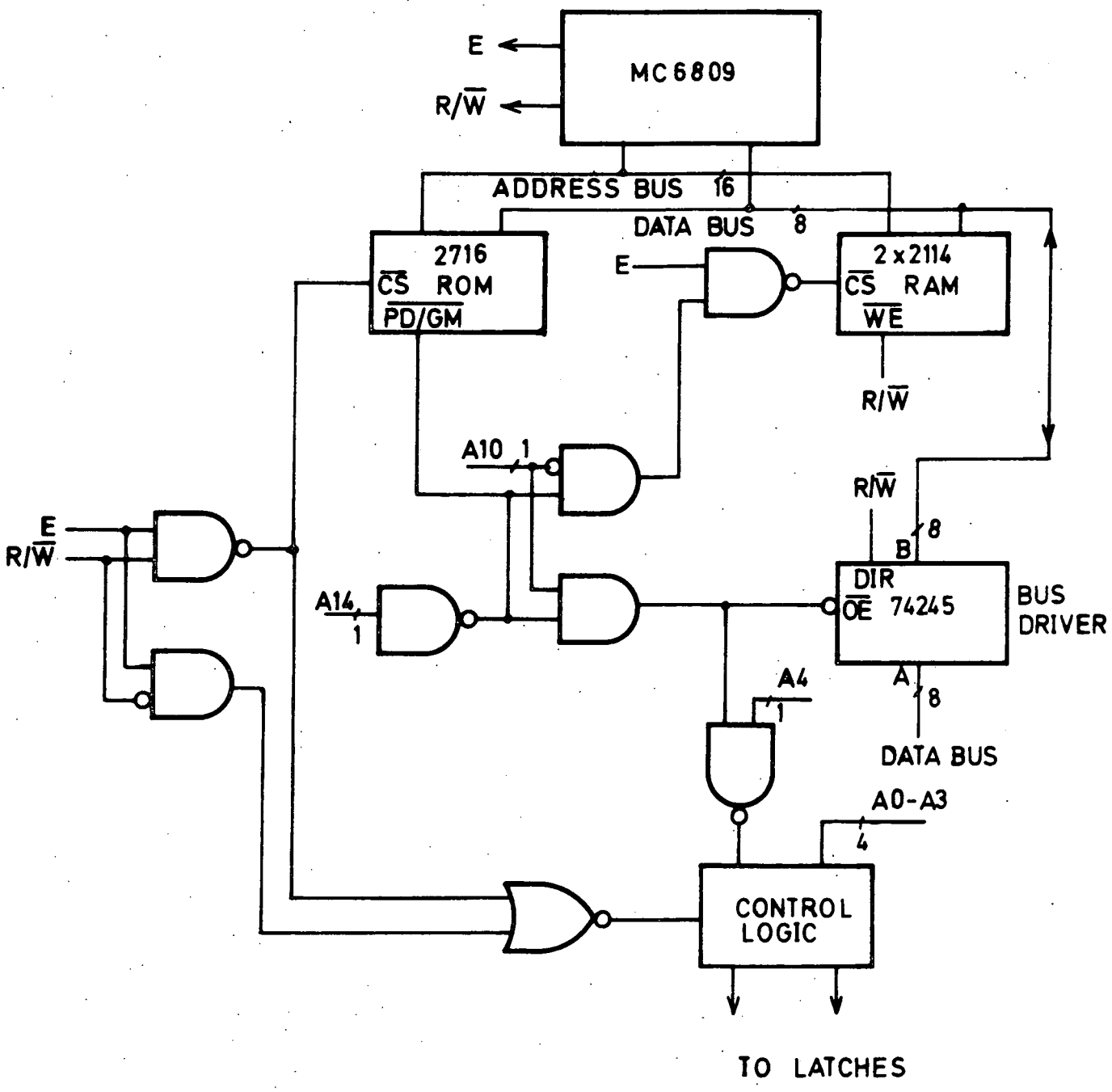



Figure 7.15: Typical slave microprocessor (16 to 18) with associated hardware.

transceiver (SN74LS245), and the direction of data flow is controlled by the R/W line.

The operation of slaves numbered 1 to 15 is as follows. After receiving the reset signal from the master, the slaves set their respective status latches to 1, and execute the SYNC instruction. If the status control bit is high, the slaves then wait until it goes low. After transferring the results to their respective output latches the slaves set the status latch to a logic 1 again. Thus allowing the master to read the output latches. If at this instant the status control bit remains low, the slaves start the next transform cycle assuming that the data have been updated in the input latches. The microprocessors numbered 16, 17 and 18 receive data from other microprocessors just before the multiplication cycle. They behave as external modular multipliers, who for the most of the time are idling (executing a series of SYNC instructions). After performing the modular multiplications, these microprocessors return the results to the appropriate microprocessors through communication latches. These microprocessors then continue to execute another series of SYNC instructions until the next multiplication cycle. Figure (7.16) shows a flowchart for the master and slave microprocessors, which also shows how the software of the master and the slaves interact. Figure (7.17) shows a timing diagram.

### 7.6.5 Software of the Slave Microprocessors

All the slave microprocessors are executing programs concurrently although the software of each of the slaves is

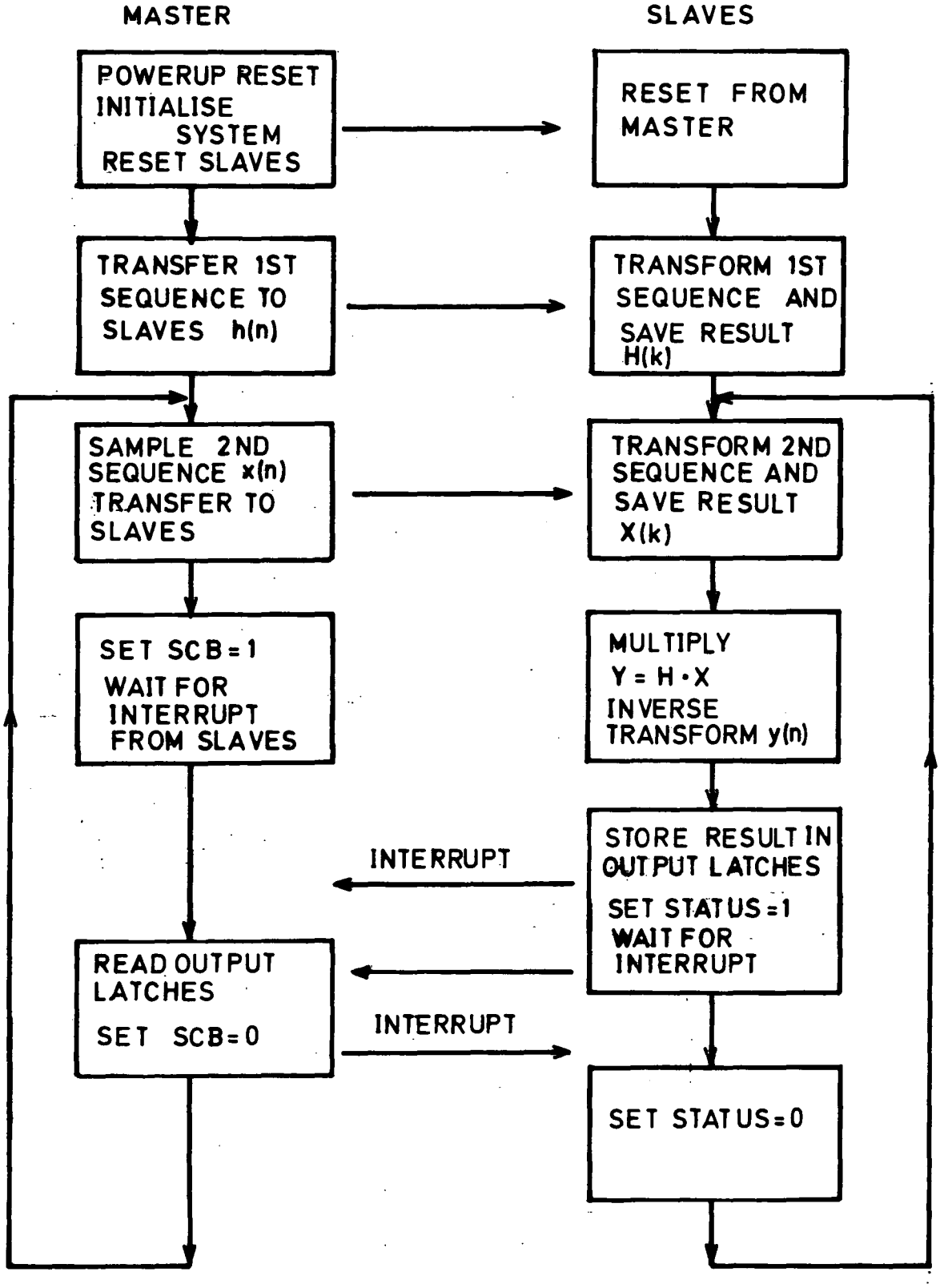



Figure 7.16 : Flow diagram for the master-slave interaction.

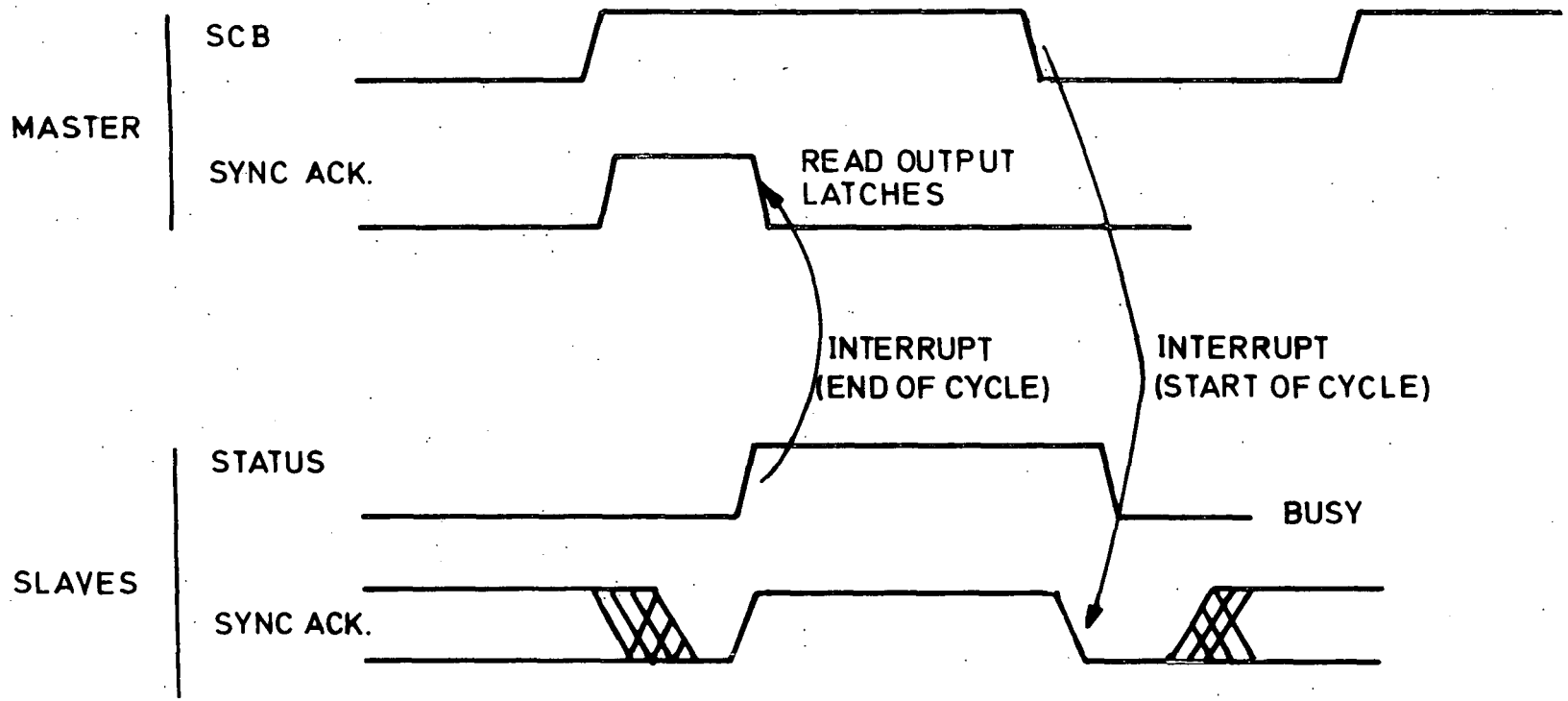



Figure 7.17: Timing diagram for the master-slave interaction.

different from any other. The source listings are given in appendix-D. The symbol Rn means that this particular address is of a read only latch and it is receiving data from the microprocessor numbered n, where n can have any value between 1 to 18. For example, in the listing for microprocessor number 1, R6 means that the microprocessor numbered 1 is receiving data from microprocessor numbered 6 whose address is $0412. Similarly, Tn indicates an address of a write only latch, where n can have any value between 1 to 18. For example, in the source listing of microprocessor number 1, T6 means that the microprocessor numbered 1 is transmitting data to microprocessor numbered 6 whose address is $0403.

All the modular arithmetic operations are coded directly in the main program. No subroutines are being used, as this would slow down the microprocessor considerably. For example for the MC6809 microprocessor a JSR (jump to subroutine) instruction requires 7 to 8 clock cycles, and an RTS (return from subroutine) requires 5 clock cycles. This means that 12 to 13 clock cycles are required for each subroutine call. Results in table (7.3) show that the time for a single subroutine call is considerable as compared to the total transform time. Table (7.2) shows number of modular arithmetic operations for the 15-point WFTA on a single and the parallel microprocessor system.

The slaves are executing their programs in an endless loop. The master must ensure that the output latches are read before they are over written by the slaves.

| | |
|---|---|
| No. of pre-weave modular additions | 39 |
| No. of modular multiplications | 18 |
| No. of post-weave modular additions | 39 |

Table 7.2a: Shows number of operations for the 15-point implementation on a uni processor.

| Proc. No. | No. of data exchange | | No. of additions |
|---|---|---|---|
| | Receive | Transmit | |
| P1 | 2 | 2 | 2 |
| P2 | 6 | 6 | 6 |
| P3 | 5 | 5 | 5 |
| P3 | 5 | 5 | 5 |
| P4 | 4 | 4 | 4 |
| P5 | 4 | 4 | 4 |
| P6 | 4 | 4 | 4 |
| P7 | 8 | 8 | 8 |
| P8 | 7 | 7 | 7 |
| P9 | 6 | 6 | 6 |
| P10 | 6 | 6 | 6 |
| P11 | 3 | 3 | 3 |
| P12 | 7 | 7 | 7 |
| P13 | 6 | 6 | 6 |
| P14 | 6 | 5 | 5 |
| P15 | 5 | 5 | 5 |
| P16 | 2 | 2 | 1 |
| P17 | 2 | 2 | 1 |
| P18 | 2 | 2 | 1 |

Table 7.2b: Shows number of operations per microprocessor for 15-point WFTA on the parallel microprocessor system. (Each microprocessor is performing one modular multiplication.)

### 7.6.6 Synchronisation of the hardware and the Software

Synchronisation among the slave microprocessors is one of the most crucial factors in this system. Recall that the slaves are executing programs from their own local program memories. The essential requirement is that they should do so in a predetermined and in a synchronised manner. Each of the slave microprocessors after performing a modular arithmetic operation, stores the result in an appropriate communication latch and executes the SYNC instruction. The sync acknowledge from all the slaves are ANDed (G2) together as shown in figures (7.12) and (7.18). This signal is inverted and fed into the FIRQ interrupt input of all the slave microprocessors. The result is that the slaves cannot proceed further until they have all executed the SYNC instruction. After receiving the interrupt the slaves read their appropriate read only latches and start processing the data further (see figure (7.17)). The advantage in this arrangement is that all the microprocessors always find valid data in the communication latches.

This synchronisation could also be achieved by coding dummy instructions such as a NOP (no operation) in the main program. The purpose of these dummy instructions would be to waste microprocessor time so that each of the modular arithmetic operation is executed in equal number of clock cycles. For example, 14, 18 or 22 clock cycles are required for a modular add if the sum > 65535, 65521 > sum > 65535, or sum < 65521 respectively.

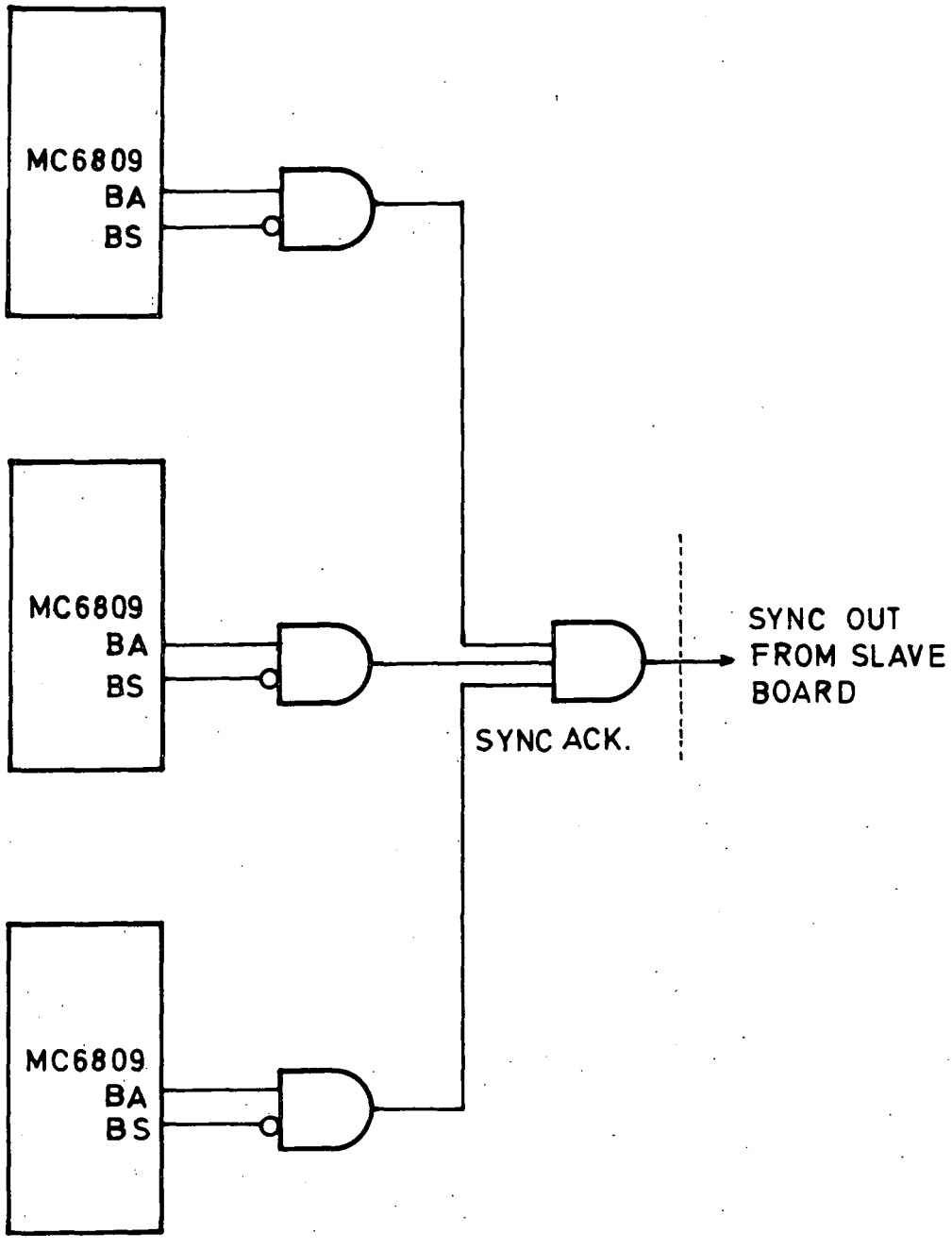



Figure 7.18: Arrangement for generating the SYNC signal from each slave board.

The former method for synchronisation was chosen for the system, because the use of the SYNC instruction optimises the program execution time for each transform cycle. However, in the latter case the dummy instructions are executed when carries are generated, so the time for the transform execution time is fixed (equivalent to worst case).

## 7.7 Transforms of Real Time Signals

A 12-bit successive approximation analogue to digital (A/D) converter (RS754) interfaced with the control microprocessor allows transforms of real time signals (see figure (7.19)). The conversion time is between 15 to 35 microseconds depending upon whether the 8-bit or 12-bit mode is being used. A sample and hold (S/H) circuit (LF398) is used to hold the input to the A/D converter steady while the conversion is being carried out.

A latch is connected to the output of the converter, such that when the conversion is complete the data are automatically latched into it. A read on this latch by the microprocessor, also sends a start convert signal to the A/D converter, and to the S/H circuit to hold the sample. The control microprocessor then executes the SYNC instruction. When the conversion is complete, the status bit from the A/D is used (as clock signal for the latch) to latch the data and simultaneously send an interrupt signal to the control microprocessor. The advantage is that the status bit (of the A/D converter) need not be monitored. The control microprocessor reads this latch, this again sends the start convert signal to the A/D converter, which then starts

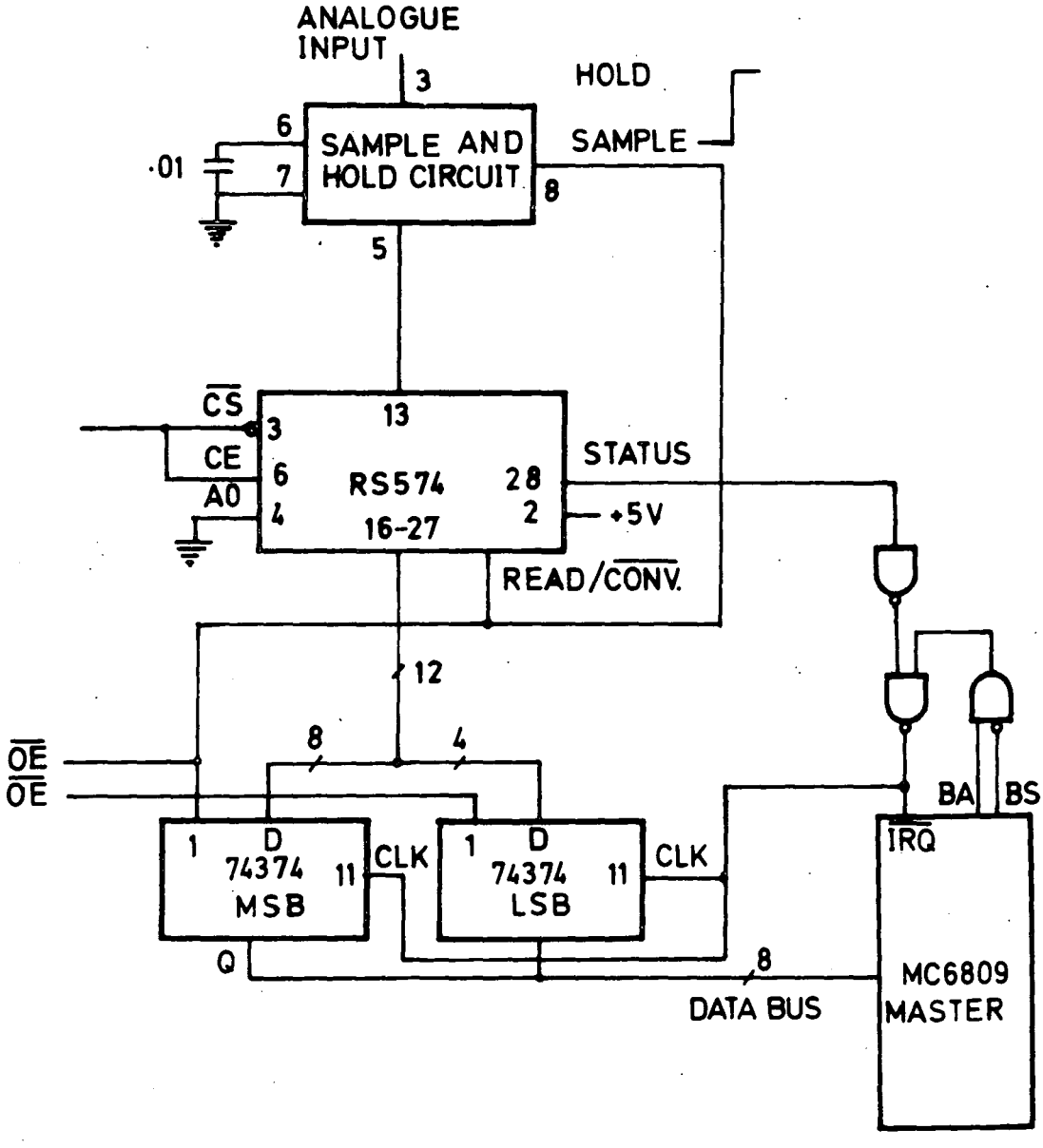



Figure 7.19: Analogue-to-Digital (A/D) interface with the master microprocessor.

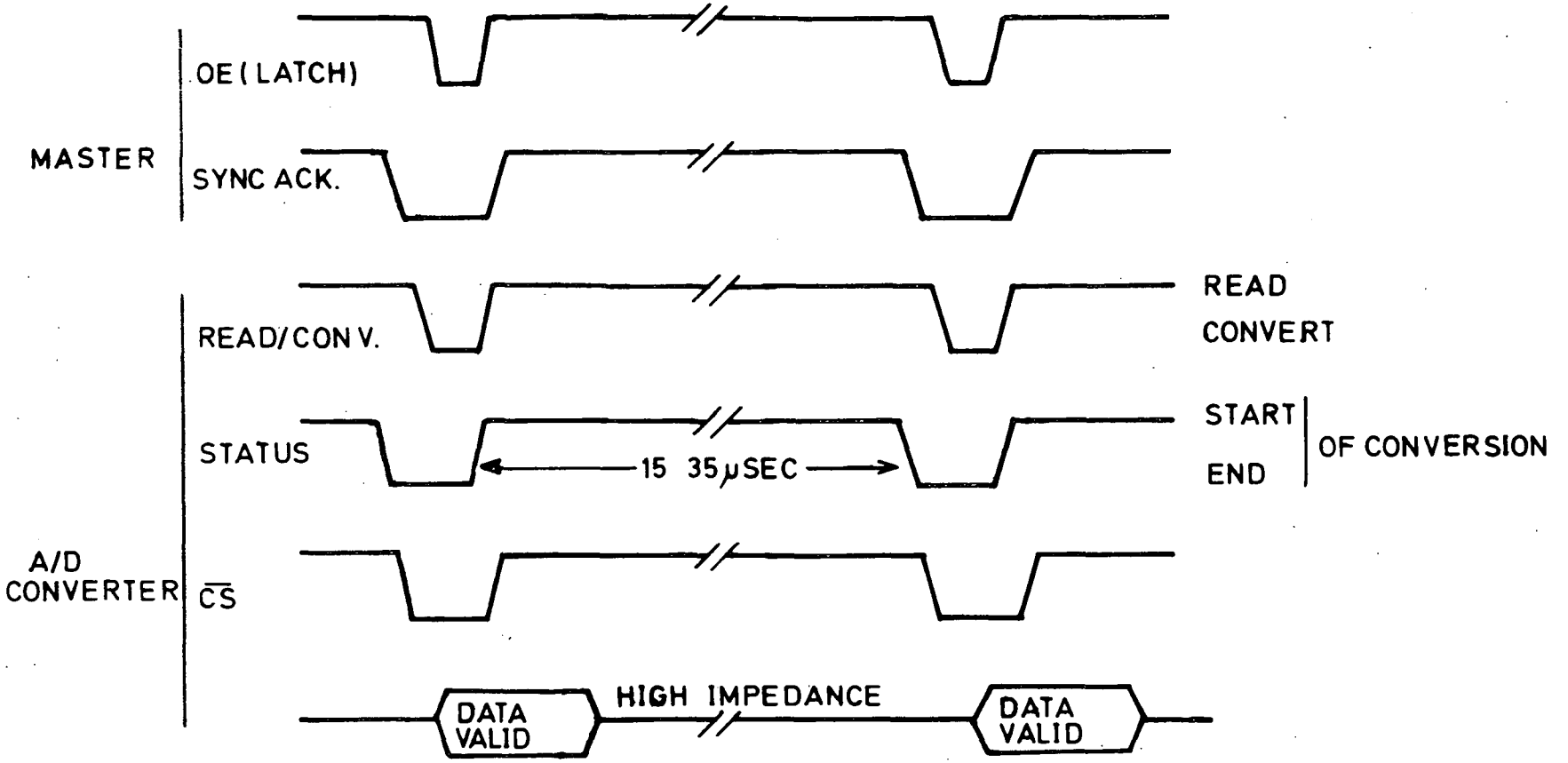



Figure 7.20: Timing diagram for the A/D converter.

converting the next sample. The use of the latch simplifies the circuitry and also increases the throughput. While the A/D is converting the next sample, the microprocessor is busy storing the previous data into the memory. In this manner full advantage of the conversion time is being utilised. A sampling rate of 28KHz is obtained, figure (7.20) shows timing diagram for the A/D conversion.

Figure (7.21) shows an arrangement for a digital to analogue (D/A) converter (DAC1220) interface. Actually there are two D/A converters interfaced with the control microprocessor. One for displaying the input and the other for displaying the transformed values on the oscilloscope. These are 12-bit multiplying D/A converters, with a typical conversion time of 1.5 microseconds.

Figures (7.22) and (7.23) show photograph of the master board and the slave board (with three microprocessors) respectively. Figure (7.24) shows a photograph of the parallel microprocessor system.

A 15-point convolution was also implemented on the parallel microprocessor system. Figure (7.25a) shows a pulse to be convolved with itself. Figure (7.25b) shows the NTT of the pulse. Figures (7.25c) and (7.25d) show the product of the two NTTs and the convolution respectively. However, if the amplitude is large then the effect of modular arithmetic can be seen in figures (7.26a-7.26d), which shows the folding of amplitude.

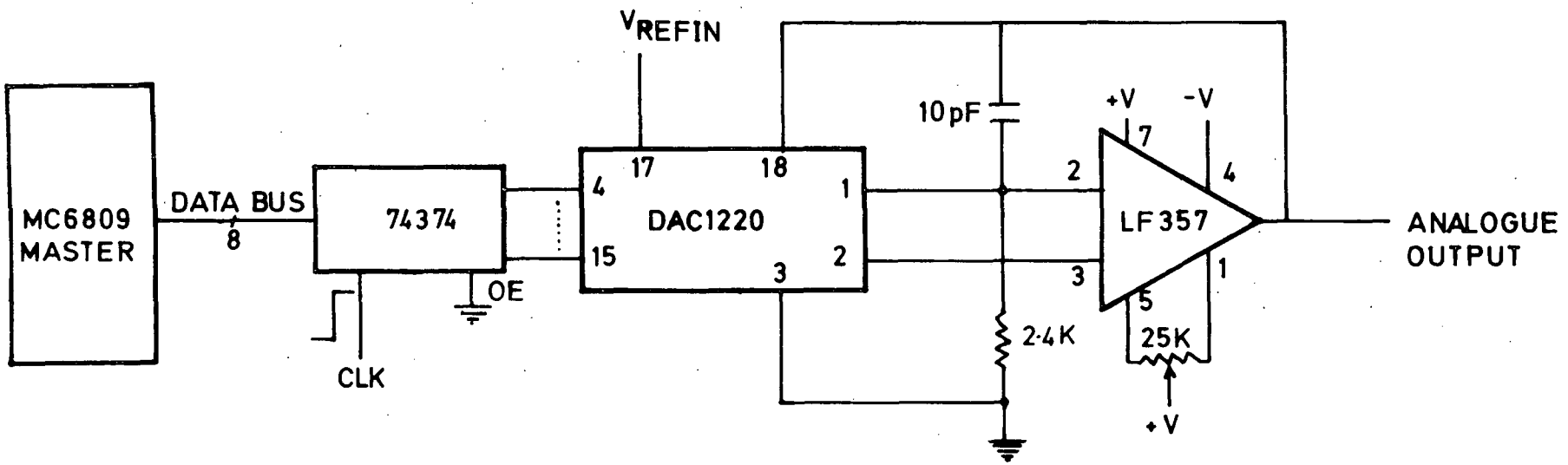



Figure 7.21: Digital-to-Analogue (D/A) interface with the master microprocessor.

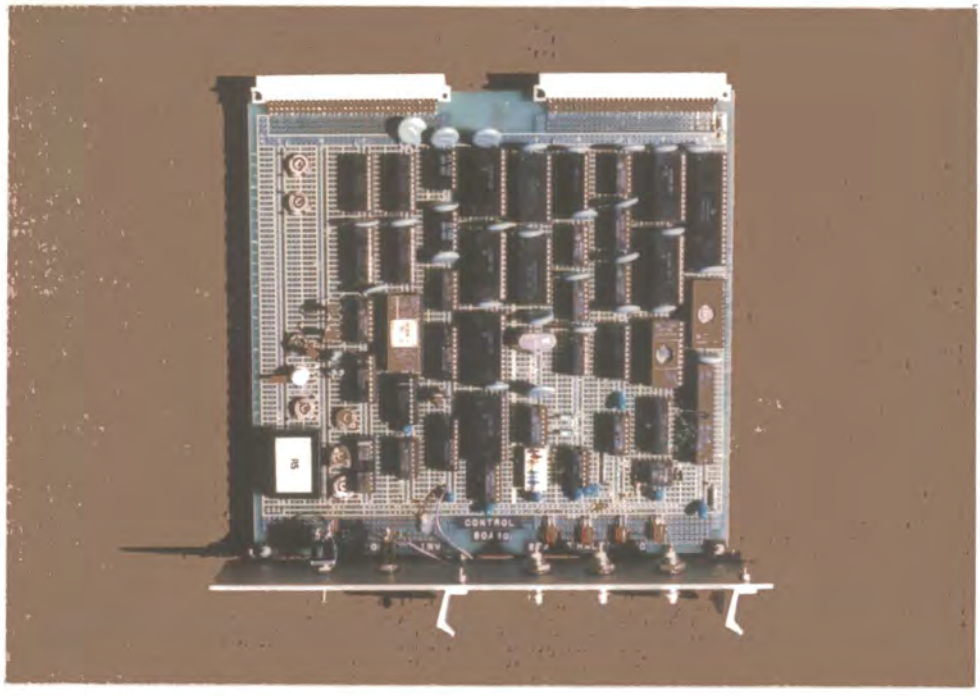


Figure 7.22: Photograph of the master microprocessor with associated hardware.

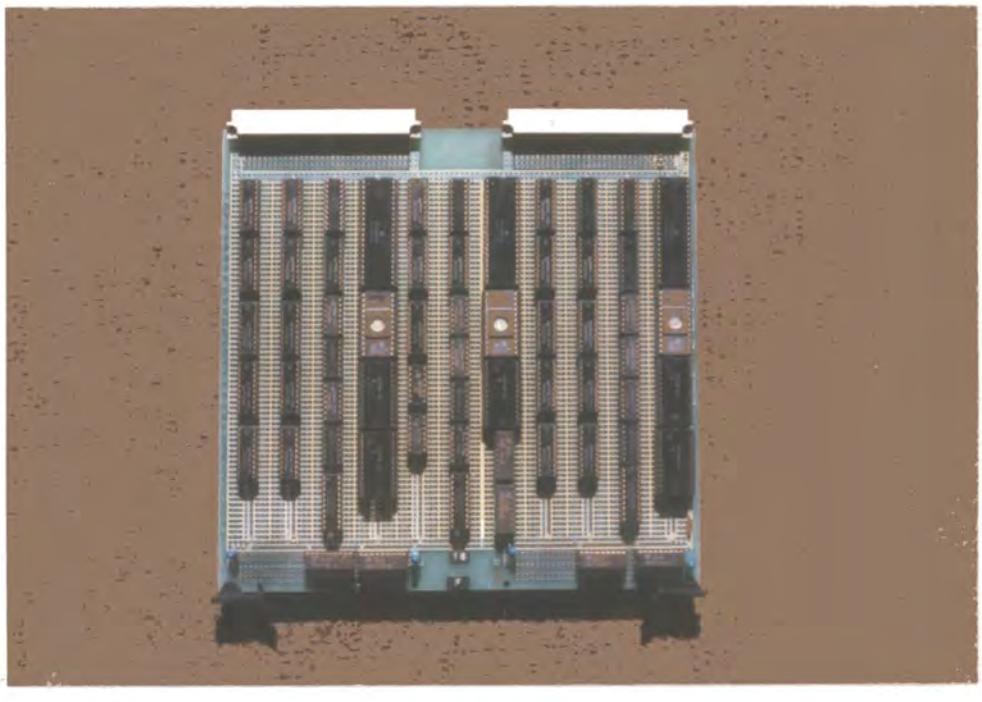


Figure 7.23: Photograph of the slave microprocessor showing three slaves on the board with associated hardware.

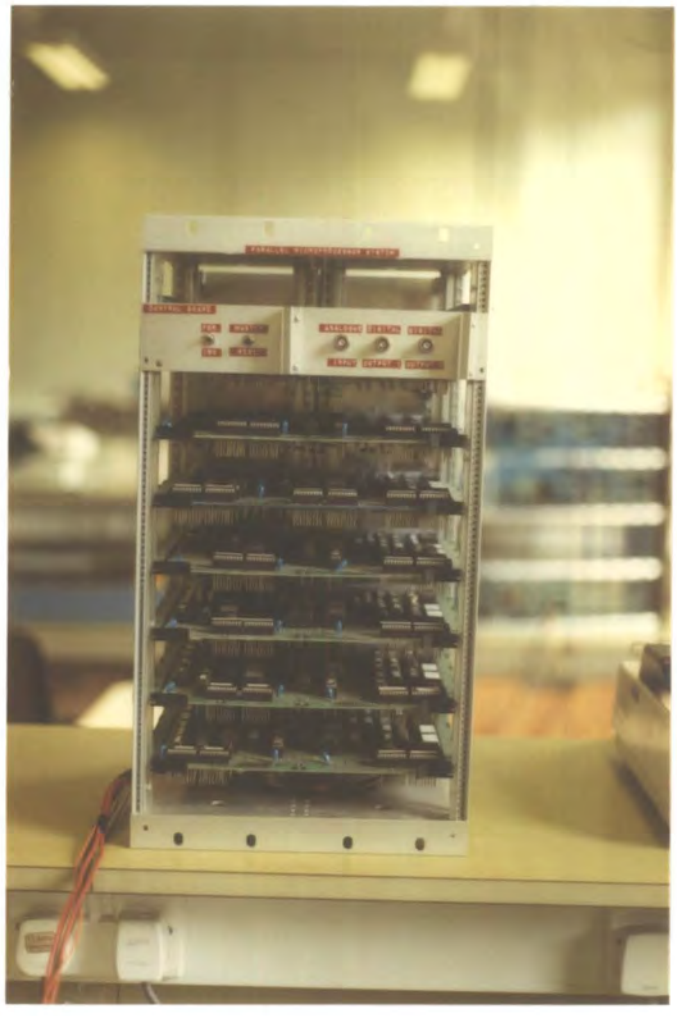


Figure 7.24: Photograph of the complete parallel microprocessor system.

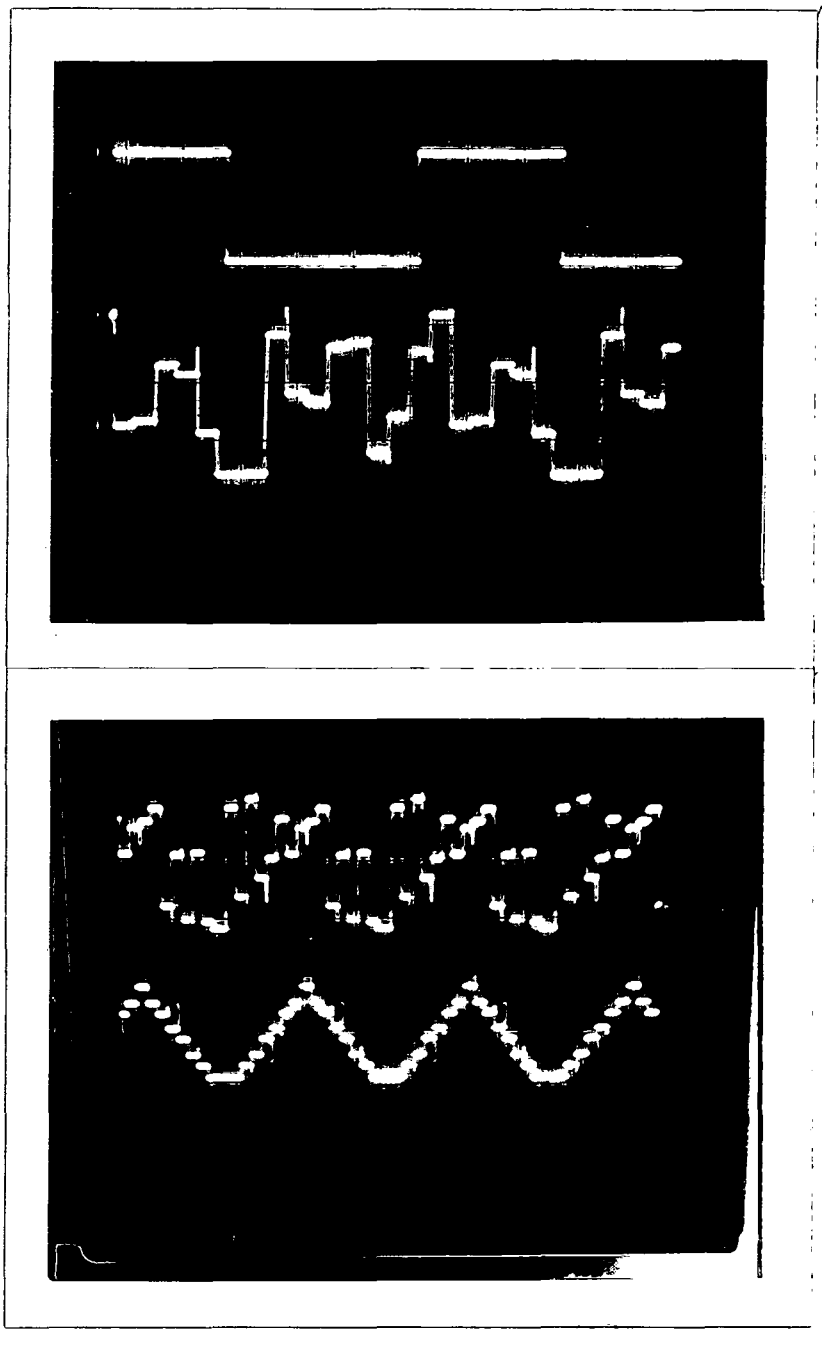



Figure 7.25
(a) Shows a pulse to be convolved with itself.
(b) Shows the NTT of the pulse.
(c) Shows product of the two NTTs.
(d) Shows convolution of the two pulses.

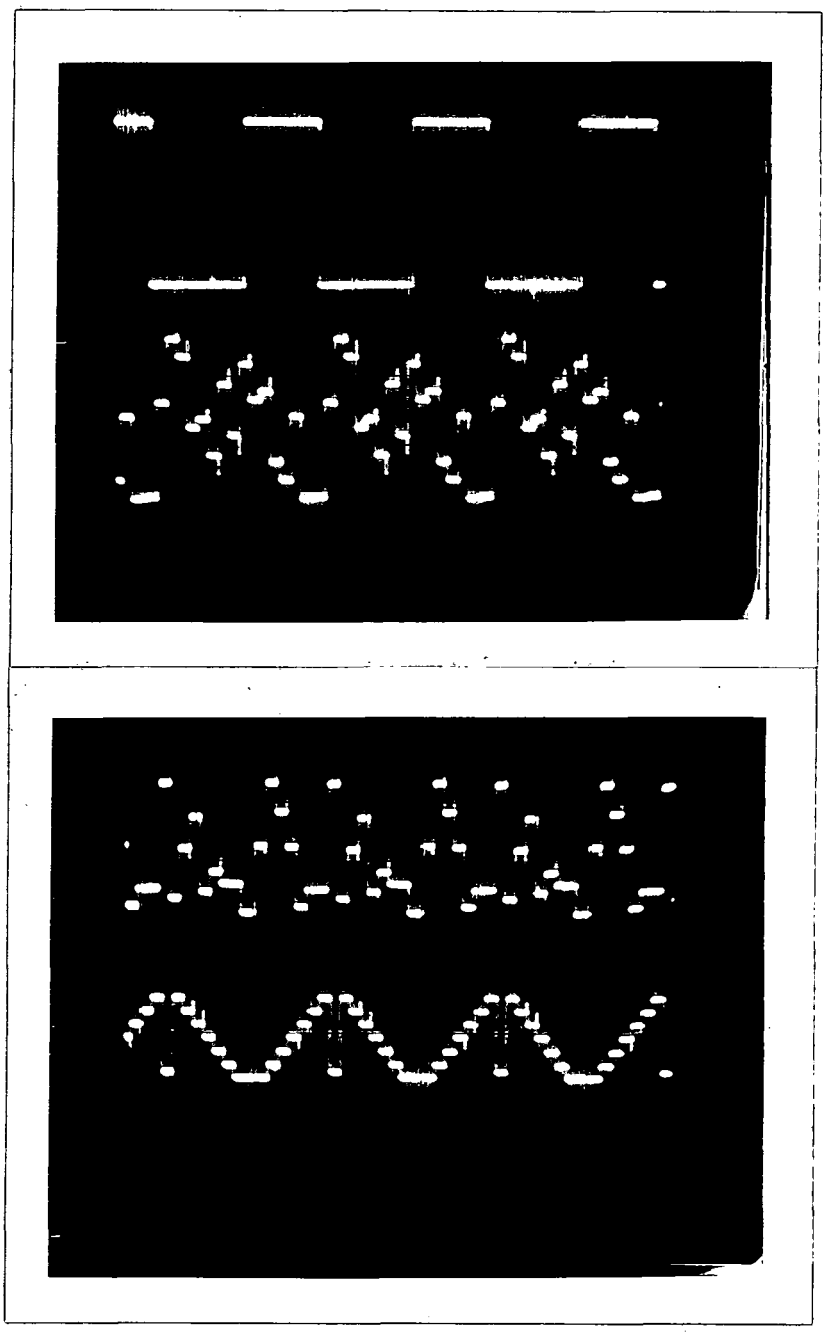



Figure 7.26

(a) Shows a pulse of a larger amplitude to be convolved with itself.
(b) Shows NTT of the pulse.
(c) Shows product of the two NTTs.
(d) Shows convolution of the two pulses, folding of the amplitude occurs due to the use of modular arithmetic.

## 7.8 Results

The program timings show that a 15-point WFTA run on a single MC6809 microprocessor requires approximately 10 milliseconds to execute. However, when the parallel dedicated microprocessor system is employed, the transform execution time is reduced to 675 microseconds.

Table (7.3) shows comparison of the 15-point WFTA execution times. The program written in FORTRAN was not optimised for time, but it gives a rough estimate for comparison.

| System | Assembler | FORTRAN |
|---|---|---|
| MC6809 | 10 msec | -- |
| Parallel Structure | 675 usec | -- |
| TMS9900 | 4 msec | -- |
| IBM 370/168 | 365 usec | 2 msec |
| IBM 370/4341 | 1 msec | 5 msec |

Table 7.3: Comparison of timings for the 15-point WFTA

The total power consumption of the system is about 65 watts, and the total cost of the system is in the range of £ 1500 (1981).

# CHAPTER 8

## Conclusion

The object of this work was to investigate and implement WFTA on microprocessors and to design hardware to improve the execution time. Special purpose hardware was also designed and constructed to exploit parallelism in the WFTA.

An external hardware modular multiplier (mod 65521) was designed, constructed and interfaced with the TMS9900 microprocessor. Since a number of modular additions and subtractions are also performed it may be beneficial to employ an external hardware modular adder (mod 65521). If an external hardware modular adder is used then only three move instructions are required for external modular add. This will save a compare, an add, and two branch instructions. There is no benefit in designing hardware for modular subtraction.

A parallel microprocessor system was designed and constructed for the implementation of the 15-point WFTA. Benchmark programs were written for several microprocessors to select a suitable microprocessor for the parallel structure. Motorola's MC6809 gave an optimum choice, since it contains an (8 x 8-bit) unsigned hardware multiplier and a SYNC instruction (the SYNC instruction is used to synchronise the microprocessor to an external event). This parallel microprocessor is a very highly dedicated MIMD machine. A host processor is used to control the

parallel structure. The use of the host processor was necessary in the development stages since it provides an interface with the parallel microprocessor system. A serious difficulty is the development of the software for the parallel microprocessor system which requires large amount of effort, since proper synchronisation between all the microprocessors must be maintained at all times.

The parallel microprocessor system being very dedicated executes the 15-point WFTA in times comparable with the IBM mainframe computers. Table (7.3) shows the program execution times on the parallel microprocessor system, MC6809 and two IBM mainframes (model 370/168 and 370/4341). All these programs were written in assembler language. This agrees with the argument given by Arden and Berenbaum (65), and Enslow (66), about achieving higher performance from several cheap processors rather than an expensive one.

This pragmatic approach to parallel processing, i.e. to implement one microprocessor per point may not seem to be a cost effective design approach for a bigger size transform. However, bigger size transforms can be implemented on the parallel microprocessor system by combining the power of each of the slave microprocessors with the power of the parallel structure. The length of this transform should be an integer multiple N of L, where N is one of the short length WFTAs, and L is the transform length implemented on the parallel structure. This may be done by allowing each of the slave microprocessors to accept N values from the master, and perform an N point preweave. Then the

parallel microprocessor system is used to perform N (L length) transforms. Finally each of the microprocessor performs the N point postweave.

The parallel structure employs microprocessors with 1 MHz clock, a 2 MHz version of the MC6809 is also available but at much higher price. If the 2 MHz version is used then faster memories have to be employed which means further increase in the total cost of the system. However, this would double the program execution speed.

Alternately, if an external modular multiplier is interfaced to each of the slave microprocessors (as described in chapter 5), this would also almost double the program execution speed. However, the cost of a modular multiplier is considerable, and this may not be practical due to cost.

The parallel microprocessor system is not 15 times faster than a single microprocessor, this is due to the over heads involved. Estimated time for 60-point WFTA on MC6809 microprocessor is about 50 milliseconds, of which 712 microseconds are required for input/output shuffle. On the parallel microprocessor system the execution time is about 3.5 milliseconds.

## Appendix-A

Modular arithmetic routines for the following microprocessors

i) TMS9900

ii) MC6809

iii) Z80

iv) 6502

32/16-bit division routine for the MC6809 microprocessor

```
*
*   ***************************************************************
*   * MODULAR ARITHMETIC PROGRAMS FOR TMS9900                     *
*   ***************************************************************
          OPTION XREF,SYMT
          AORG  >4000
*
* ********************************
* *   MODULAR ADDITION         *
* ********************************
*
START     LWPI  WKS
          MOV   @AD1,R1
          MOV   @AD2,R2
          A     R1,R2
          JOC   OVER
          CI    R2,65521
          JL    OVR
OVER      AI    R2,15
OVR       MOV   R2,@SUM
*
* ********************************
* * MODULAR SUBTRACTION         *
* ********************************
          MOV   @SUBT1,R1
          MOV   @SUBT2,R2
          MOV   R1,R3
          S     R2,R1
          C     R3,R2
          JHE   OVER1
          AI    R1,65521
OVER1     MOV   R1,@RES
*
*   ***************************************************************
*   * MODULAR ARITHMETIC PROGRAMS FOR MC6809                       *
*   ***************************************************************
*
          NAM   M6809
          OPT   CRE,L,S,W,P
          ORG   $30
*
* ********************************
* * MODULAR  ADDITION           *
* ********************************
*
START     LDX   #ADS
          LDD   ,X++
          ADDD  ,X++
          BCS   SKIP
          CMPD  #65521
          BLO   SKIP1
SKIP      ADDD  #15
SKIP1     STD   ,X
* ********************************
* * MODULAR MULTIPLICATION      *
* ********************************
          MOV   @MPR,R1
          MOV   @MPR,R1
          MOV   @MPD,R2
          MPY   R1,R2
          DIV   @MOD,R2
          MOV   R3,@PROD
          B     @>0080
*
WKS       BSS   32
AD1       BSS   2
AD2       BSS   2
SUM       BSS   2
SUBT1     BSS   2
SUBT2     BSS   2
RES       BSS   2
MPR       BSS   2
MPD       BSS   2
PROD      BSS   2
MOD       DATA  65521
LAST      END   START
*
*
*
*
          JMP   OVER
ADS       FDB   0
          FDB   0
          FDB   0
*
* ********************************
* * MODULAR SUBTRACTION         *
* ********************************
*
OVER      LDX   #SBTN
          LDD   ,X++
          SUBD  ,X++
          BCC   SKIP2
          ADDD  #65521
SKIP2     STD   ,X
          JMP   OVER1
```

```
SBTN      FDB   0
          FDB   0
          FDB   0
*
* ********************************
* * 16*16 BIT MULTIPICATION   *
* ********************************
*
OVER1     LDX   #MLTR
          LDY   #MLTN
          LDU   #PROD1
          CLR   0,U
          CLR   1,U
          LDA   1,X
          LDB   1,Y
          MUL
          STD   2,U
          LDA   0,X
          LDB   1,Y
          MUL
          ADDD  1,U
          STD   1,U
          BCC   SKIP3
          INC   0,U
SKIP3     LDA   1,X
          LDB   0,Y
          MUL
          ADDD  1,U
          STD   1,U
          BCC   SKIP4
          INC   0,U
SKIP4     LDA   0,X
          LDB   0,Y
          MUL
          ADDD  0,U
          STD   0,U
*
* ********************************
* *      MODULARISING          *
* ********************************
*
;
;  ***************************************************************
;  * MODULAR ARITHMETIC PROGRAMS FOR Z80                         *
;  ***************************************************************
;
;
; ********************************
; *      MODULAR ADDITION        *
; ********************************
          ORG   100H
START:    LD    HL,(ADD1)
          LD    BC,(ADD2)
          ADD   HL,BC
          LDA   1,U
          BEQ   OMIT
          LDB   #15
          MUL
          ADDD  2,U
          BCS   SKIP6
          CMPD  #65521
          BLO   SKIP7
SKIP6     ADDD  #15
SKIP7     STD   2,U
OMIT      LDA   0,U
          BEQ   OMIT1
SKIP8     LDY   #TEMP
          CLR   0,Y
          CLR   1,Y
          CLR   2,Y
          LDB   #15
          MUL
SKIPA     STD   0,Y
          LDA   0,Y
          BEQ   SKIPE
          LDB   #15
          MUL
          ADDD  1,Y
          BRA   SKIPD
SKIPE     LDD   1,Y
SKIPD     ADDD  2,U
          BCS   SKIP8
          CMPD  #65521
          BLO   SKIPC
SKIP8     ADDD  #15
SKIPC     STD   2,U
OMIT1     JMP   $F564
MLTR      FDB   0
MLTN      FDB   0
PROD1     FCB   0
PROD2     FCB   0
PROD3     FCB   0
PROD4     FCB   0
TEMP      FDB   0
          END

          JP    C,OVER1
          LD    A,255
          CP    H
          JP    NZ,OVER
          LD    A,241
          CP    L
          JP    Z,OVER1
          JP    NC,OVER
```

```
OVER1:    LD     BC,15
          ADD    HL,BC
          LD     (SUM),HL
OVER:     JP     SKIP
;
ADD1:     DEFW   0
ADD2:     DEFW   0
SUM:      DEFW   0
; ********************************
; *     MODULAR SUBTRACTION      *
; ********************************
SKIP:     LD     HL,(SUBT1)
          LD     DE,(SUBT3)
          AND    A
          SBC    HL,DE
          LD     A,(SUBT3)
          LD     D,A
          LD     A,(SUBT1)
          CP     D
          JP     NC,OVR
          JP     Z,ZERO
BACK:     LD     BC,65521
          ADD    HL,BC
          JP     OVR
ZERO:     LD     A,(SUBT4)
          LD     D,A
          LD     A,(SUBT2)
          CP     D
          JP     NC,OVR
          JP     Z,OVR
          JP     BACK
OVR:      LD     (RES),HL
          JP     SKIP2
;
SUBT1:    DEFB   0
SUBT2:    DEFB   0
SUBT3:    DEFB   0
SUBT4:    DEFB   0
RES:      DEFW   0
; ********************************
; *   MODULAR MULTIPLICATION     *
; ********************************
SKIP2:    LD     A,(MPR1)
          LD     H,A
          LD     A,(MPD1)
          LD     E,A
          CALL   MULT
          LD     (PROD3),HL
          LD     A,(MPR2)
          LD     H,A
          LD     A,(MPD2)
          LD     E,A
          CALL   MULT
          LD     (PROD1),HL
          LD     A,(MPR2)
          LD     H,A
          LD     A,(MPD1)
          LD     E,A
          CALL   MULT
          LD     (PROD5),HL
          LD     A,(MPR1)
          LD     H,A
          LD     A,(MPD2)
          LD     E,A
          CALL   MULT
          LD     DE,(PROD5)
          ADD    HL,DE
          JP     NC,BAK
          LD     B,1
          LD     A,(PROD2)
          ADD    A,B
          LD     (PROD2),A
BAK:      LD     (PROD5),HL
          LD     A,(PROD4)
          LD     E,A
          LD     A,(PROD1)
          LD     D,A
          ADD    HL,DE
          JP     NC,BAK1
          LD     B,1
          LD     A,(PROD2)
          ADD    A,B
          LD     (PROD2),A
BAK1:     LD     (PROD5),HL
          LD     A,H
          LD     (PROD1),A
          LD     A,L
          LD     (PROD4),A
;
; ********************************
; *  PROD1:PROD2:PROD3:PROD4     *
; ********************************
;
          LD     A,(PROD1)
          LD     H,A
          LD     E,15
          CALL   MULT
          LD     DE,(PROD3)
          ADD    HL,DE
          JP     NC,BAK2
          LD     BC,15
          ADD    HL,BC
          LD     (PROD3),HL
          JP     BAK3
BAK2:     LD     (PROD3),HL
          LD     A,255
          CP     H
          JP     NZ,BAK3
```

```
         LD     A,241
         CP     L
         JP     Z,BAK6
         JP     NC,BAK3
BAK6:    LD     BC,15
         ADD    HL,BC
         LD     (PROD3),HL
BAK3:    LD     A,(PROD2)
         LD     H,A
         LD     E,15
         CALL   MULT
         LD     A,L
         LD     (TMP2),A
;
         LD     A,0
         LD     (TMP1),A
         LD     E,15
         CALL   MULT
         LD     DE,(TMP1)
         ADD    HL,DE
         LD     DE,(PROD3)
         ADD    HL,DE
         JP     NC,BAK4
         LD     BC,15
         ADD    HL,BC
         JP     BAK5
BAK4:    LD     (PROD3),HL
         LD     A,255
         CP     H
         JP     NZ,BAK5
         LD     A,241
         CP     L
         JP     Z,BAK7
         JP     NC,BAK5
BAK7:    LD     BC,15
         ADD    HL,BC
BAK5:    LD     (PROD3),HL
         JP     0000H
;
; ******************************
; * MULTIPLICATION SUBROUTINE*
; ******************************
;
MULT:    LD     L,0
         LD     D,0
         LD     B,8
JUMP:    ADD    HL,HL
         JR     NC,NOADD
         ADD    HL,DE
NOADD:   DJNZ   JUMP
         RET
MPD1:    DEFB   0
MPD2:    DEFB   0
MPR1:    DEFB   0
MPR2:    DEFB   0
PROD1:   DEFB   0
PROD2:   DEFB   0
PROD3:   DEFB   0
PROD4:   DEFB   0
PROD5:   DEFB   0
PROD6:   DEFB   0
TMP1:    DEFB   0
TMP2:    DEFB   0
         END
*
*  ************************************************************
*  * MODULAR ARITHMETIC PROGRAMS FOR 6502                     *
*  ************************************************************
         NAM    M6502
         ORG    $1024
*
* ******************************
* * MODULAR ADDITION          *
* ******************************
*
START    LDX    #AD1
         CLC
         LDA    1,X
         ADC    3,X
         STA    5,X
         LDA    0,X
         ADC    2,X
         STA    4,X
         BCS    OVR
         CMP    #$FF
         BNE    SUBT1
         LDA    5,X
         CMP    #$F1
         BEQ    SKIP1
         BMI    SUBT1
OVR      LDA    5,X
SKIP1    CLC
         ADC    #15
         STA    5,X
         LDA    #0
         ADC    4,X
         STA    4,X
SUBT1    JMP    SUBT
*
         ORG    $0023
AD1      FDB    0
AD2      FDB    0
SUM      FCB    0
SUM1     FCB    0
```

```
*
* ********************************
* * MODULAR SUBTRACTION        *
* ********************************
*
          ORG    $1024
SUBT      LDX    #SUB
          LDA    #0
          STA    CHECK
          LDA    0,X
          CMP    2,X
          BEQ    OMIT
          BCS    JMP
          INC    CHECK
JMP       LDA    1,X
JMP1      SEC
          SBC    3,X
          STA    5,X
          LDA    0,X
          SBC    2,X
          STA    4,X
          LDA    CHECK
          BEQ    MULT1
          CLC
          LDA    5,X
          ADC    #$F1
          STA    5,X
          LDA    4,X
          ADC    #$FF
          STA    4,X
MULT1     JMP    MULT
OMIT      LDA    1,X
          CMP    3,X
          BEQ    OMIT1
          BCS    JMP1
          INC    CHECK
          JMP    JMP1
OMIT1     LDA    #0
          STA    4,X
          STA    5,X
          JMP    MULT
          ORG    $0023
SUB       FDB    0
SUB1      FDB    0
SUB2      FCB    0
SUB3      FDB    0
CHECK     FCB    0
*
* ********************************
* * MULTIPLICATION ROUTINE    *
* ********************************
*
          ORG    $1024
MULT      LDX    #MPLR
          LDA    8,X
          STA    2,X
          LDA    6,X
          STA    0,X
          JSR    SUBRT
          LDA    4,X
*
          STA    16,X
          LDA    3,X
          STA    15,X
          LDA    7,X
          STA    2,X
          LDA    6,X
          STA    0,X
          JSR    SUBRT
          LDA    4,X
          STA    14,X
          LDA    3,X
          STA    13,X
          LDA    8,X
          STA    2,X
          LDA    5,X
          STA    0,X
          JSR    SUBRT
          LDA    4,X
          STA    12,X
          LDA    3,X
          STA    11,X
          LDA    7,X
          STA    2,X
*
          LDA    5,X
          STA    0,X
          JSR    SUBRT
          LDA    4,X
          STA    10,X
          LDA    3,X
          STA    9,X
*
          CLC
          LDA    14,X
          ADC    12,X
          STA    14,X
          LDA    13,X
          ADC    11,X
          STA    13,X
          LDA    #0
          ADC    9,X
          STA    9,X
          CLC
          LDA    15,X
          ADC    14,X
          STA    15,X
          LDA    13,X
```

```
          ADC   10,X
          STA   14.X
          LDA   #0
          ADC   9,X
          STA   13,X
*
* ********************************
* * ROUTINE FOR MODULARISING *
* ********************************
*
          LDA   15,X
          CMP   #$FF
          BNE   JMPA
          LDA   16,X
          CMP   #$F1
          BEQ   JMPB
          BCC   JMPA
JMPB      CLC
          ADC   #15
          STA   16,X
          LDA   #$0
          ADC   15,X
          STA   15,X
JMPA      LDA   14,X
          STA   2,X
          LDA   #15
          STA   0,X
          JSR   SUBRT
          CLC
          LDA   16,X
          ADC   4,X
          STA   16,X
          LDA   15,X
          ADC   3,X
          STA   15,X
          BCC   OVRA
          LDA   15,X
          CMP   #$FF
          BNE   OVRA
          LDA   16,X
          CMP   #$F1
          BEQ   JMPC
          BCC   OVRA
JMPC      CLC
          ADC   #15
          STA   16,X
          LDA   #$0
          ADC   15,X
          STA   15,X
OVRA      LDA   #0
          STA   17,X
          STA   18,X
          STA   19,X
          LDA   13,X
          STA   2,X
          LDA   #15
          STA   0,Y
*
          JSR   SUBRT
          LDA   4,X
          STA   18,X
          LDA   3,X
          STA   2,X
          LDA   #15
          STA   0,X
          JSR   SUBRT
          CLC
          LDA   4,X
          ADC   19,X
          STA   19,X
          LDA   3,X
          ADC   18,X
          STA   18,X
          CLC
          LDA   16,X
          ADC   19,X
          STA   16,X
          LDA   15,X
          ADC   18,X
          STA   15,X
          BCS   JUMPC
          CMP   #$FF
          BNE   OVER1
          LDA   16,X
          CMP   #$F1
          BEQ   JUMPC
          BCC   OVER1
JUMPC     CLC
          LDA   16,X
          ADC   #15
          STA   16,X
          LDA   #0
          ADC   15,X
          STA   15,X
OVER1     BRK
*
* ********************************
* *  MULTIPLICATION ROUTINE  *
* ********************************
*
SUBRT     LDA   #0
          STA   1,X
          STA   3,X
          STA   4,X
          LDY   #8
          JMP   BAK
OVER      ASL   1,X
          ASL   2,X
```

```
        BCC   BAK
        LDA   #1
        ORA   1,X
        STA   1,X
BAK     CLC
        ROR   0,X
        BCC   BAK1
        CLC
        LDA   2,X
        ADC   4,X
        STA   4,X
        LDA   1,X
        ADC   3,X
        STA   3,X
BAK1    DEY
        BEQ   OUT
        JMP   OVER
OUT     RTS
*
*
*
*  ***********************************************************
*  * 32/16 BIT DIVISION FOR MC6809 MICROPROCESSOR           *
*  ***********************************************************
        NAM   DIVISION
        ORG   $0000
START   LDX   #MLTR
        LDY   #MLTN
        LDU   #PROD1
        CLR    ,U
        CLR   1,U
        LDA   1,X
        LDB   1,Y
        MUL
        STD   2,U
        LDA    ,X
        LDB   1,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKIP3
        INC    ,U
SKIP3   LDA   1,X
        LDB    ,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKIP4
        INC    ,U
SKIP4   LDA    ,X
        LDB    ,Y
        MUL
        ADDD   ,U
        STD    ,U
        ORG   $0023
MPLR    FCB   0
MCND1   FCB   0
MCND2   FCB   0
TEMP1   FCB   0
TEMP2   FCB   0
MPR     FDB   0
MND     FDB   0
PROD1   FCB   0
PROD2   FCB   0
PROD3   FCB   0
PROD4   FCB   0
PROD5   FCB   0
PROD6   FCB   0
PROD7   FCB   0
PROD8   FCB   0
TMP1    FCB   0
TMP2    FCB   0
TMP3    FCB   0
        END   START
*
*  ********************************
*  * 32-BIT PRODUCT IN            *
*  * PROD1:PROD2:PROD3:PROD4      *
*  * 32 BIT / 16 BIT UNSIGNED     *
*  * DIVISION                     *
*  ********************************
*
        LDD   PROD1
        STD   DVND2
        LDD   PROD3
        STD   DVND4
        LDD   #0
        STA   DVND1
        STD   QUOT1
        LDA   #16
        STA   COUNT
DIVICE  ASL   DVND5
        ROL   DVND4
        ROL   DVND3
        ROL   DVND2
        ROL   DVND1
        LDA   DVND1
        BNE   SKIP
        LDD   DVND2
        CMPD  DVSR2
        BCS   CHECK
SKIP    LDA   DVND3
        SUBA  DVSR3
        STA   DVND3
```

```
        LDA   DVND2          |DVSR3   FCB   00
        SBCA  DVSR2          |REM    FCB   00
        STA   DVND2          |QUOT1  FCB   00
        LDA   DVND1          |QUOT2  FDB   00
        SBCA  DVSR1          |MLTR   FDB   00
        STA   DVND1          |MLTN   FCB   00
        ASL   QUOT2          |PROD1  FCB   00
        ROL   QUOT1          |PROD2  FCB   00
        INC   QUOT2          |PROD3  FCB   00
CHECK   DEC   COUNT          |PROD4  FCB   00
        BNE   DIVIDE         |DVND1  FCB   00
        LDD   DVND2          |DVND2  FCB   00
        STD   REM            |DVND3  FCB   00
        JMP   $D283          |DVND4  FCB   00
COUNT   FCB   00             |DVND5  FCB   00
DVSR1   FCB   00             |       END
DVSR2   FCB   00             |
```

## Appendix-B

Assembler program source listing for a 15-point WFTA (TMS9900)

FORTRAN program source listing for a 15-point WFTA

```
*       ***************************************************
*       *      15-POINT WINOGRAD ALGORITHM (WFTA) TMS9900   *
*       ***************************************************
        IDT     'WIN015'
        OPTION XREF,SYMT
        AORG    >6000
START   LWPI  WSP
        LI      R4,YREG
        LI      R5,XREG
*
* ********************************
* *  INPUT SHUFFLE             *
* ********************************
*
        MOV   *R4,*R5
        MOV   @6(R4),@2(R5)
        MOV   @12(R4),@4(R5)
        MOV   @18(R4),@6(R5)
        MOV   @24(R4),@8(R5)
        MOV   @10(R4),@10(R5)
        MOV   @16(R4),@12(R5)
        MOV   @22(R4),@14(R5)
        MOV   @28(R4),@16(R5)
        MOV   @4(R4),@18(R5)
        MOV   @20(R4),@20(R5)
        MOV   @26(R4),@22(R5)
        MOV   @2(R4),@24(R5)
        MOV   @8(R4),@26(R5)
        MOV   @14(R4),@28(R5)
*
* ********************************
* * 3 POINT PREWEAVE           *
* ********************************
*
        LI    R5,XREG
*
LOOP1   MOV   @10(R5),R0
        MOV   @20(R5),R1
        BL    @ADDSUB
        MOV   R2,@10(R5)
        MOV   R3,@20(R5)
        MOV   *R5,R3
        BL    @ADD
        MOV   R3,*R5
*
        MOV   @12(R5),R0
        MOV   @22(R5),R1
        BL    @ADDSUB
        MOV   R2,@12(R5)
        MOV   R3,@22(R5)
        MOV   @2(R5),R3
        BL    @ADD
        MOV   R3,@2(R5)
*
        MOV   @14(R5),R0
        MOV   @24(R5),R1
        BL    @ADDSUB
        MOV   R2,@14(R5)
        MOV   R3,@24(R5)
        MOV   @4(R5),R3
        BL    @ADD
        MOV   R3,@4(R5)
*
        MOV   @16(R5),R0
        MOV   @26(R5),R1
        BL    @ADDSUB
        MOV   R2,@16(R5)
        MOV   R3,@26(R5)
        MOV   @6(R5),R3
        BL    @ADD
        MOV   R3,@6(R5)
*
        MOV   @18(R5),R0
        MOV   @28(R5),R1
        BL    @ADDSUB
        MOV   R2,@18(R5)
        MOV   R3,@28(R5)
        MOV   @8(R5),R3
        BL    @ADD
        MOV   R3,@8(R5)
*
* ********************************
* * 5 POINT PREWEAVE           *
* ********************************
*
        LI    R6,ZREG
        MOV   @2(R5),R0
        MOV   @8(R5),R1
        BL    @ADDSUB
        MOV   R2,@2(R5)
        MOV   R3,@6(R6)
*
        MOV   @6(R5),R0
        MOV   @4(R5),R1
        BL    @ADDSUB
        MOV   R2,@4(R5)
        MOV   R3,@10(R6)
        MOV   @6(R6),R2
        BL    @ADD
        MOV   R3,@8(R6)
*
        MOV   @2(R5),R0
        MOV   @4(R5),R1
```

```
        BL    @ADDSUB
        MOV   R2,@2(R6)
        MOV   R3,@4(R6)
        MOV   *R5,R3
        BL    @ADD
        MOV   R3,*R6
*
*
        MOV   @12(R5),R0
        MOV   @18(R5),R1
        BL    @ADDSUB
        MOV   R2,@12(R5)
        MOV   R3,@18(R6)
*
        MOV   @16(R5),R0
        MOV   @14(R5),R1
        BL    @ADDSUB
        MOV   R2,@14(R5)
        MOV   R3,@22(R6)
        MOV   @18(R6),R2
        BL    @ADD
        MOV   R3,@20(R6)
*
        MOV   @12(R5),R0
        MOV   @14(R5),R1
        BL    @ADDSUB
        MOV   R2,@14(R6)
        MOV   R3,@16(R6)
        MOV   @10(R5),R3
        BL    @ADD
        MOV   R3,@12(R6)
*
        MOV   @22(R5),R0
        MOV   @28(R5),R1
        BL    @ADDSUB
        MOV   R2,@22(R5)
        MOV   R3,@30(R6)
*
        MOV   @26(R5),R0
        MOV   @24(R5),R1
        BL    @ADDSUB
        MOV   R2,@24(R5)
        MOV   R3,@34(R6)
        MOV   @30(R6),R2
        BL    @ADD
        MOV   R3,@32(R6)
*
        MOV   @22(R5),R0
        MOV   @24(R5),R1
        BL    @ADDSUB
        MOV   R2,@26(R6)
        MOV   R3,@28(R6)
        MOV   @20(R5),R3
        BL    @ADD
        MOV   R3,@24(R6)
*
* ********************************
* * MULTIPLICATION              *
* ********************************
*
        MOV   @FWD,R1
        JEQ   FRWD
        LI    R7,COEFR
*
        JMP   OVER
FRWD    LI    R7,COEFF
*
OVER    LI    R4,0
        LI    R8,65521
LOOP    MOV   *R7+,R1
        MOV   @ZREG(R4),R2
        MPY   R1,R2
        DIV   R8,R2
        MOV   R3,@ZREG(R4)
        INCT  R4
        CI    R4,36
        JNE   LOOP
*
* ********************************
* * 5 POINT POSTWEAVE           *
* ********************************
*
        MOV   *R6,R3
        MOV   R3,*R5
        MOV   @2(R6),R2
        BL    @ADD
        MOV   R3,@2(R6)
        MOV   @6(R6),R0
        MOV   @8(R6),R1
        BL    @SUB
        MOV   R3,@6(R6)
        MOV   @8(R6),R2
        MOV   @10(R6),R3
        BL    @ADD
        MOV   R3,@10(R6)
        MOV   @2(R6),R0
        MOV   @4(R6),R1
        BL    @ADDSUB
        MOV   R2,@2(R6)
        MOV   R3,@4(R6)
        MOV   @2(R6),R0
        MOV   @6(R6),R1
        BL    @ADDSUB
        MOV   R2,@2(R5)
        MOV   R3,@8(R5)
        MOV   @4(R6),R0
        MOV   @10(R6),R1
        BL    @ADDSUB
```

```
        MOV   R2,@4(R5)
        MOV   R3,@6(R5)
*
        MOV   @12(R6),R3
        MOV   R3,@10(R5)
        MOV   @14(R6),R2
        BL    @ADD
        MOV   R3,@14(R6)
        MOV   @18(R6),R0
        MOV   @20(R6),R1
        BL    @SUB
        MOV   R3,@18(R6)
        MOV   @20(R6),R2
        MOV   @22(R6),R3
        BL    @ADD
        MOV   R3,@22(R6)
        MOV   @14(R6),R0
        MOV   @16(R6),R1
        BL    @ADDSUB
        MOV   R2,@14(R6)
        MOV   R3,@16(R6)
        MOV   @14(R6),R0
        MOV   @18(R6),R1
        BL    @ADDSUB
        MOV   R2,@12(R5)
        MOV   R3,@18(R5)
        MOV   @16(R6),R0
        MOV   @22(R6),R1
        BL    @ADDSUB
        MOV   R2,@14(R5)
        MOV   R3,@16(R5)
*
        MOV   @24(R6),R3
        MOV   R3,@20(R5)
        MOV   @26(R6),R2
        BL    @ADD
        MOV   R3,@26(R6)
        MOV   @34(R6),R2
        MOV   @32(R6),R3
        BL    @ADD
        MOV   R3,@34(R6)
        MOV   @30(R6),R0
        MOV   @32(R6),R1
        BL    @SUB
        MOV   R3,@30(R6)
        MOV   @26(R6),R0
        MOV   @28(R6),R1
        BL    @ADDSUB
        MOV   R2,@26(R6)
        MOV   R3,@28(R6)
        MOV   @26(R6),R0
        MOV   @30(R6),R1
        BL    @ADDSUB
        MOV   R2,@22(R5)
        MOV   R3,@28(R5)
        MOV   @28(R6),R0
        MOV   @34(R6),R1
        BL    @ADDSUB
        MOV   R2,@24(R5)
        MOV   R3,@26(R5)
*
* ******************************
* * 3 POINT POSTWEAVE           *
* ******************************
*
        MOV   *R5,R3
        MOV   @10(R5),R2
        BL    @ADD
        MOV   R3,@10(R5)
*
        MOV   @2(R5),R3
        MOV   @12(R5),R2
        BL    @ADD
        MOV   R3,@12(R5)
*
        MOV   @4(R5),R3
        MOV   @14(R5),R2
        BL    @ADD
        MOV   R3,@14(R5)
*
        MOV   @6(R5),R3
        MOV   @16(R5),R2
        BL    @ADD
        MOV   R3,@16(R5)
*
        MOV   @8(R5),R3
        MOV   @18(R5),R2
        BL    @ADD
        MOV   R3,@18(R5)
*
        MOV   @10(R5),R0
        MOV   @20(R5),R1
        BL    @ADDSUB
        MOV   R2,@10(R5)
        MOV   R3,@20(R5)
*
        MOV   @12(R5),R0
        MOV   @22(R5),R1
        BL    @ADDSUB
        MOV   R2,@12(R5)
        MOV   R3,@22(R5)
*
        MOV   @14(R5),R0
        MOV   @24(R5),R1
        BL    @ADDSUB
        MOV   R2,@14(R5)
        MOV   R3,@24(R5)
*
```

```
          MOV   @16(R5),R0
          MOV   @26(R5),R1
          BL    @ADDSUB
          MOV   R2,@16(R5)
          MOV   R3,@26(R5)
*
          MOV   @18(R5),R0
          MOV   @28(R5),R1
          BL    @ADDSUB
          MOV   R2,@18(R5)
          MOV   R3,@28(R5)
*
* ********************************
* * OUTPUT SHUFFLE             *
* ********************************
*
          MOV   *R5,*R6
          MOV   @12(R5),@2(R6)
          MOV   @24(R5),@4(R6)
          MOV   @6(R5),@6(R6)
          MOV   @18(R5),@8(R6)
          MOV   @20(R5),@10(R6)
          MOV   @2(R5),@12(R6)
          MOV   @14(R5),@14(R6)
          MOV   @26(R5),@16(R6)
          MOV   @8(R5),@18(R6)
          MOV   @10(R5),@20(R6)
          MOV   @22(R5),@22(R6)
          MOV   @4(R5),@24(R6)
          MOV   @16(R5),@26(R6)
          MOV   @28(R5),@28(R6)
*
          B     @>0800
*
* ********************************
* * ADD & SUBTRACT SUBROUTINE*
* ********************************
*
ADDSUB    MOV   R1,R2
          A     R0,R2
          JOC   PLUS
          CI    R2,65521
          JL    SUB
PLUS      AI    R2,15
SUB       MOV   R0,R3
          S     R1,R3
          C     R1,R0
          JL    FIN
          AI    R3,65521
FIN       RT
*
* ********************************
* * ADDITION SUBROUTINE        *
* ********************************
*
ADD       A     R2,R3
          JOC   PLUS1
          CI    R3,65521
          JL    TAG
PLUS1     AI    R3,15
TAG       RT
*
*
* ********************************
* *      SHUFFLE     VECTORS    *
* ********************************
*
COEFF DATA     1, 16379, 13376,
      DATA 19136, 18005, 48647,
      DATA 32759,  8192, 45457,
      DATA 36817,  5753, 25311,
      DATA 16087, 29032,  8748,
      DATA 23174, 43615,  1465,
COEFR DATA 61153,  5460, 18364,
      DATA 46773, 20640,  5493,
      DATA  6552, 57331, 37975,
      DATA 28122, 34561, 24521,
      DATA 29504, 28641, 12521,
      DATA  5913, 24748, 21938,
*
WSP       BSS   32
YREG      BSS   30
XREG      BSS   30
ZREG      BSS   36
LIM       BSS   2
FWD       BSS   2
LAST      END   START
```

```
C
C    ***************************************************************
C    *     15-POINT WINOGRAD ALGORITHM (WFTA)                      *
C    ***************************************************************
C
      IMPLICIT REAL*8(A - H,O - Z)
      DIMENSION X(15), Y(15), Z(18), OUT(15)
      DIMENSION COEF(18), COEFR(18)
      INTEGER IRF(15), IRFI(15)
      REAL*8 MODO
C
C     INPUT SHUFFLE VECTORS
C
      DATA IRF /0, 3, 6, 9, 12, 5, 8, 11, 14,
     1     2, 10, 13, 1, 4, 7/
C
C     OUTPUT SHUFFLE VECTORS
C
      DATA IRFI /0, 6, 12, 3, 9, 10, 1, 7, 13,
     1     4, 5, 11, 2, 8, 14/
C
C     FORWARD TRANSFORM COEFFICIENTS
C
      DATA COEF /1.D0, 16379.D0, 13376.D0, 19136.D0,
     1     18005.D0, 48647.D0, 32759.D0, 8192.D0,
     2     45457.D0, 36817.D0, 5753.D0, 25311.D0,
     3     16087.D0, 29032.D0, 8748.D0, 23174.D0,
     4     43615.D0, 1465.D0/
      DATA COEFR /61153.D0, 5460.D0, 18364.D0, 46773.D0,
     1     20640.D0,  5493.D0,  6552.D0, 57331.D0,
     2     37975.D0, 28122.D0, 34561.D0, 24521.D0,
     3     29504.D0, 28641.D0,12521.D0, 5913.D0,
     4     24748.D0, 21938.D0/
C
C     READ INPUT DATA ARRAY
C
      FRD = 0.0
      READ (5,*) (Y(I),I=1,15)
      DO 10 I = 1, 15
   10 X(I) = Y(IRF(I) + 1)
      DO 20 I = 1, 5
        T = MODO(X(5 + I) + X(10 + I))
        X(I) = MODO(X(I) + T)
        X(10 + I) = MODO(X(5 + I) - X(10 + I))
        X(5 + I) = T
   20 CONTINUE
      J = 1
      DO 30 I = 1, 3
        IND = 5 * (I - 1)
        S1 = MODO(X(IND + 2) + X(IND + 5))
        S2 = MODO(X(IND + 2) - X(IND + 5))
        S3 = MODO(X(IND + 4) + X(IND + 3))
        S4 = MODO(X(IND + 4) - X(IND + 3))
        S5 = MODO(S1 + S3)
```

```
        S6 = MODD(S1 - S3)
        S7 = MODD(S2 + S4)
        S8 = MODD(S5 + X(IND + 1))
        Z(J) = S8
        Z(J + 1) = S5
        Z(J + 2) = S6
        Z(J + 3) = S2
        Z(J + 4) = S7
        Z(J + 5) = S4
        J = J + 6
   30 CONTINUE
      IF (FRD .EQ. 1.D0) GO TO 50
      DO 40 I = 1, 18
   40 Z(I) = MODD(Z(I)*COEF(I))
      GO TO 70
   50 DO 60 I = 1, 18
   60 Z(I) = MODD(Z(I)*COEFR(I))
   70 J = 1
      DO 80 I = 1, 3
        IND = 5 * (I - 1)
        S9 = MODD(Z(J) + Z(J + 1))
        S10 = MODD(S9 + Z(J + 2))
        S11 = MODD(S9 - Z(J + 2))
        S12 = MODD(Z(J + 3) - Z(J + 4))
        S13 = MODD(Z(J + 4) + Z(J + 5))
        S14 = MODD(S10 + S12)
        S15 = MODD(S10 - S12)
        S16 = MODD(S11 + S13)
        S17 = MODD(S11 - S13)
        X(IND + 1) = Z(J)
        X(IND + 2) = S14
        X(IND + 3) = S16
        X(IND + 4) = S17
        X(IND + 5) = S15
        J = J + 6
   80 CONTINUE
      DO 90 I = 1, 5
        T = MODD(X(I) + X(5 + I))
        T2 = MODD(T + X(10 + I))
        X(10 + I) = MODD(T - X(10 + I))
        X(5 + I) = T2
   90 CONTINUE
      DO 100 I = 1, 15
        OUT(IRFI(I) + 1) = X(I)
  100 CONTINUE
      WRITE (6,110) (Y(I),I=1,15)
  110 FORMAT (' ', 5F10.2)
      WRITE (6,120)
  120 FORMAT (' ', //)
      WRITE (6,130) (OUT(I),I=1,15)
  130 FORMAT (' ', 5F10.2)
      STOP
      END
C
```

```
C
C
C
      DOUBLE PRECISION FUNCTION MODD(F)
      REAL*8 F, MOD
      MOD = 65521.D0
      IF (F .LT. 0.0D0) GO TO 10
      MODD = DMOD(F,MOD)
      GO TO 20
   10 MODD = MOD - DMOD(-F,MOD)
   20 RETURN
      END
```

## Appendix-C

FORTH program source listing for a 60-point WFTA (TMS9900)

```
   ( THIS PROGRAM PERFORMS WINOGRAD LENGTH 60
     FORWARD AND REVERSE TRANSFORM )
   ( INPUT ARRAY IS Y AND THE RESULT OF TRANSFORM
     IS ALSO STORED IN ARRAY Y      )

:S
   DECIMAL  ( VARIABLES USED FOR TEMPORARY STORAGE )
   0 INTEGER S0  0 INTEGER S1  0 INTEGER S2  0 INTEGER S3
   0 INTEGER S4  0 INTEGER S5  0 INTEGER T1  0 INTEGER T2
   0 INTEGER T3  0 INTEGER T4  0 INTEGER T5  0 INTEGER TM0
   0 INTEGER TM1 0 INTEGER TM2 0 INTEGER TM3 0 INTEGER TM4
   0 INTEGER TM
            ( ARRAYS USED FOR COMPUTATION )
   144 ARRAY FCOEF 144 ARRAY RCOEF 120 ARRAY X 144 ARRAY Y
   120 ARRAY RF 120 ARRAY RFI
: SINT 0 S0 ! 2 S1 ! 4 S2 ! 6 S3 ! 8 S4 ! 10 S5 ! ;
: INTZ 0 TM0 ! 2 TM ! 4 TM1 ! 6 TM2 ! 8 TM3 ! 10 TM4 ! ;
: 1CHG TM0 @ 10 + TM0 ! TM @ 10 + TM ! TM1 @ 10 + TM1 !
       TM2 @ 10 + TM2 ! TM3 @ 10 + TM3 ! TM4 @ 10 + TM4 ! ;
: 2CHG S0 @ 12 + S0 ! S1 @ 12 + S1 ! S2 @ 12 + S2
       ! S3 @ 12 + S3 ! S4 @ 12 + S4 ! S5 @ 12 + S5 ! ;
:S

   ( INPUT SHUFFLE VECTORS )     RF FILL
    0  72  24  96  48  90  42 114  66  18  60  12  84  36 108
   30 102  54   6  78  80  32 104  56   8  50   2  74  26  98
   20  92  44 116  68 110  62  14  86  38  40 112  64  16  88
   10  82  34 106  58 100  52   4  76  28  70  22  94  46 118

   ( OUTPUT SHUFFLE VECTORS )    RFI FILL
    0  24  48  72  96  30  54  78 102   6  60  84 108  12  36
   90 114  18  42  66  40  64  88 112  16  70  94 118  22  46
  100   4  28  52  76  10  34  58  82 106  80 104   8  32  56
  110  14  38  62  86  20  44  68  92 116  50  74  98   2  26

:S
  ( COEFFICIENTS FOR FORWARD TRANSFORM )
 FCOEF FILL
      1  16379  13376  64390  46385  48647
      1  16379  13376  64390  46385  48647
      1  16379  13376  64390  46385  48647
  41224  13991  53009  26608  10376  22681
  32759   8192  45457  34457  28704  25311
  32759   8192  45457  34457  28704  25311
  32759   8192  45457  34457  28704  25311
   3685  11774  18768  25609  49957  64260
  49434  36489  56773  45080  23174  64056
  49434  36489  56773  45080  23174  64056
  49434  36489  56773  45080  23174  64056
  33074  56939     32   5797  28796  17202

:S
( COEFFICIENTS FOR REVERSE TRANSFORM )
```

```
RCOEF FILL
  64429   1365   4591   9847   4687  50514
  64429   1365   4591   9847   4687  50514
  64429   1365   4591   9847   4687  50514
   3681  11779  30785  35388   4541  64807
   1638  30713  25874  17990  25730  55271
   1638  30713  25874  17990  25730  55271
   1638  30713  25874  17990  25730  55271
  27239  15092  52104  12439  25949   1071
  58145   9220  13250  44432  50619  27276
  58145   9220  13250  44432  50619  27276
  58145   9220  13250  44432  50619  27276
  50784   2041  30577  40308  60673  45578

:S
 ( MODULAR MULTIPLICATION ROUTINE FOR THE EXTERNAL
   HARDWARE MODULAR MULTIPLIER )
HEX  CODE ALOAD  3FF2 2 LI 3FF4 3 LI 3FF6 4 LI RETURN
DECIMAL
CODE 1CALC   8 POP 9 POP 0 8 1 2 MOV 0 9 1 3
             MOV 1 4 0 7 MOV 7 PUSH RETURN
: AMULT  ALOAD    144 0 DO I RCOEF + @ I Y + @ 1CALC
                  I Y + ! 2 +LOOP ;
: BMULT  ALOAD    144 0 DO I RCOEF + @ I Y + @ 1CALC
                  I Y + ! 2 +LOOP ;
: CMULT  FLAG 0 = IF AMULT ELSE BMULT THEN ;
:S
  ( MODULAR ADDITION )   HEX
CODE MOD 1 POP 2 POP 0 1 0 2 A FNC IF ELSE F 1 AI
         THEN FFF1 1 CI FH IF F 1 AI 1 PUSH ELSE
         1 PUSH THEN RETURN
  ( MODULAR MULTIPLICATION )
CODE D/ 7 POP 5 POP FFF1 4 LI 5 0 7 MPY
        5 0 4 DIV 6 PUSH RETURN
                  ( REG4 CONTAINS DIVISOR )
  ( MODULAR SUBTRACTION )
CODE SBT 2 POP 1 POP 0 3 0 1 MOV 0 1 0 2 S 0 2 0 3 C
     FLT IF FFF1 1 AI 1 PUSH ELSE 1 PUSH THEN RETURN
  ( MODULAR HARDWARE MULTPLIER )
HEX  CODE CREG 0 7 CLR 0 8 CLR 0 9 CLR RETURN
CODE ALOAD 3FF2 2 LI 3FF4 3 LI 3FF6 4 LI RETURN
CODE CALC 0 8 1 2 MOV 0 9 1 3 MOV 1 4 0 7 MOV 7 PUSH RETURN
:S
   ( 3 POINT PRE-WEAVE )  DECIMAL
: 3AD 40 0 DO I 40 + X + @ I 80 + X + @ OVER OVER MOD I
      40 + Y + ! SBT I 80 + Y + ! 2 +LOOP ;
: 3DAD 40 0 DO I 40 + Y + @ I X + @ MOD I Y +
       ! 2 +LOOP ;
: I3PT 3AD 3DAD ;

   ( 4 POINT PRE-WEAVE )
: 41AD 10 0 DO I Y + @ I 20 + Y + @ MOD I X + ! 2 +LOOP ;
: 42AD 10 0 DO I 10 + Y + @ I 30 + Y + @ OVER OVER MOD
       I 10 + X + ! SBT I 30 + X + ! 2 +LOOP ;
```

```
: 42SB 10 0 DO I Y + @ I 20 + Y + @ SBT I 20 + X + ! 2
       +LOOP ;
: 43AD 10 0 DO I X + @ I 10 + X + @ MOD TM ! I X + @ I 10 +
       X + @ SBT I 10 + X + ! TM @ I X + ! 2 +LOOP ;
:S
: 44AD 10 0 DO I 40 + Y + @ I 60 + Y + @ OVER OVER MOD
       I 40 + X + ! SBT I 60 + X + ! 2 +LOOP ;
: 45AD 10 0 DO I 50 + Y + @ I 70 + Y + @ OVER OVER MOD
       I 50 + X + ! SBT I 70 + X + ! 2 +LOOP ;
: 48AD 10 0 DO I 40 + X + @ I 50 + X + @ MOD TM ! I 40 + X +
       @ I 50 + X + @ SBT I 50 + X + ! TM @ I 40 + X + ! 2 +LOOP ;
: 49AD 10 0 DO I 80 + Y + @ I 100 + Y + @ OVER OVER MOD
       I 80 + X + ! SBT I 100 + X + ! 2 +LOOP ;
: 4AAD 10 0 DO I 90 + Y + @ I 110 + Y + @ OVER OVER MOD
       I 90 + X + ! SBT I 110 + X + ! 2 +LOOP ;
: 4DAD 10 0 DO I 80 + X + @ I 90 + X + @ MOD TM ! I 80 + X +
       @ I 90 + X + @ SBT I 90 + X + ! TM @ I 80 + X + ! 2 +LOOP ;
: I4PT 41AD 42AD 42SB 43AD 44AD 45AD 48AD 49AD 4AAD 4DAD ;

:S
  ( MULTIPLICATION WITH COEFFICIENTS )
  0 INTEGER FLAG
: FMULT 144 0 DO I FCOEF + @ I Y + @ D/ I Y + ! 2 +LOOP ;
: RMULT 144 0 DO I RCOEF + @ I Y + @ D/ I Y + ! 2 +LOOP ;
: MULT FLAG @ 0 =  IF FMULT ELSE RMULT THEN ;
  ( 5 POINT PRE-WEAVE )
: I15PT TM @ X + @ TM3 @ X + @ OVER OVER MOD S1 @ Y + !
        SBT S5 @ Y + ! ;
: I25PT TM1 @ X + @ TM2 @ X + @ MOD S2 @ Y + ! TM2 @ X
        + @ TM1 @ X + @ SBT S4 @ Y + ! ;
: I35PT S1 @ Y + @ S2 @ Y + @ OVER OVER MOD S1 @ Y + !
        SBT S2 @ Y + ! TM0 @ X + @ S1 @ Y + @ MOD S0 @ Y + ! ;
: I45PT S5 @ Y + @ S4 @ Y + @ MOD S3  @ Y + !  ;
: I5PT INTZ SINT 24 0 DO I15PT I25PT I35PT I45PT 2CHG
       1CHG 2 +LOOP ;
:S
  ( 5 POINT POST-WEAVE )
: FVPT S0 @ Y + @ DUP S1 @ Y + @ MOD T1 ! TM0 @ X + ! ;
: 1FVPT S3 @ Y + @  S5 @ Y + @ MOD T5 ! ;
: 2FVPT S3 @ Y + @  S4 @ Y + @ SBT T3 ! ;
: 3FVPT T1 @ S2 @ Y + @ OVER OVER MOD T2 ! SBT T4 ! ;
: 4FVPT T2 @ T3 @ OVER OVER MOD TM @ X + ! SBT
        TM3 @ X + ! ;
: 5FVPT T4 @ T5 @ OVER OVER MOD TM1 @ X + ! SBT
        TM2 @ X + ! ;
: 05PT INTZ SINT 24 0 DO FVPT 1FVPT 2FVPT 3FVPT
        4FVPT 5FVPT 2CHG 1CHG 2 +LOOP ;

:S
  ( 4 POINT POST-WEAVE )
: 401 10 0 DO I X + @ I Y + ! 2 +LOOP ;
: 14D 10 0 DO I 20 + X + @ I 30 + X + @ OVER OVER
      MOD I 10 + Y + ! SBT I 30 + Y + ! I 10 + X + @ I 20
      + Y + ! 2 + LOOP ;
```

```
: 402 10  0 DO I 40 + X + @ I 40 + Y + ! 2 +LOOP ;
: 24D 10 0 DO I 60 + X + @ I 70 + X + @ OVER OVER MOD I
      50 + Y + ! SBT I 70 + Y + ! I 50 + X + @ I 60 + Y + !
      2 + LOOP ;
: 403 10  0 DO I 90 + X + @ I 80 + Y + ! 2 +LOOP ;
: 34D 10  0 DO I 100 + X + @ I 110 + Y + @ OVER OVER MOD I
      90 + Y + ! SBT I 110 + Y + ! I 90 + X + @
      I 100 + Y + ! 2 + LOOP ;
: 04PT 401 14D 402 24D 403 34D ;

:S
  ( 3 POINT POST-WEAVE )
: 03PT 40 0 DO I Y + @ I 40 + Y + @ MOD
       I 80 + Y + @ OVER OVER MOD I 40 + X + ! SBT I 80
       + X + ! I Y + @ I X + ! 2 +LOOP ;
  ( INPUT RE-ORDERING VECTOR RF )
: IORD 120 0 DO I RF + @ Y + @ I X + ! 2 +LOOP ;
  ( OUTPUT RE-ORDERING VECTOR RFI )
: OORD 120 0 DO I X + @ I RFI + @ Y + ! 2 +LOOP ;

:S

: TRANSFORM     IORD I3PT I4PT I5PT MULT O5PT
                O4PT O3PT OORD ;
: 1TRANSFORM    IORD I3PT I4PT I5PT CMULT O5PT
                O4PT O3PT OORD ;

 ( FRD FOR FOWARD AND INV FOR INVERSE TRANSFORM
   USING MULTIPLY AND DIVIDE INSTRUCTION )
: FRD 0 FLAG ! TRANSFORM ; : INV 1 FLAG ! TRANSFORM ;
 ( 1FRD FOR FOWARD AND 1INV FOR INVERSE TRANSFORM
   USING EXTERNAL HARDWARE MODULAR MULTIPLIER )
: 1FRD 0 FLAG ! 1TRANSFORM ; : 1INV 1 FLAG ! 1TRANSFORM ;
       X EMPTY Y EMPTY
:S
```

## Appendix-D

Assembler program source listings for the slave microprocessors (1 to 18).

Assembler program source listing for the master microprocessor

Assembler program source listing for a 15-point WFTA (MC6809)

```
*       ************************************************************************
*       *                        PROCESSOR NUMBER 1                            *
*       ************************************************************************
            NAM   68091
OUTPUT      EQU   $0400
STATUS      EQU   $0402
T6          EQU   $0403
T2          EQU   $0405
INPUT       EQU   $0410
R6          EQU   $0412
R2          EQU   $0414
SEM         EQU   $0416
*
            ORG    $F800
            NOP
            ORCC   #%01010000
            LDU    #PROD1
BEGIN       CLRA
            STA    FLAG
            LDA    SEM
            BEQ    FRD
START       LDA    #1
            STA    FLAG
FRD         LDY    #MCND
            LDX    #MLTPR
            LDA    #1
            STA    STATUS
            SYNC
            CLRA
            STA    STATUS
            LDD    INPUT
            BRA    OVER
NEXT        LDY    #MCND
            LDX    #MLTPR
            SYNC
            LDD    SAVE
*
OVER        SYNC
            SYNC
            ADDD   R6
            BCS    SKP12
            CMPD   #65521
            BLO    SKP13
SKP12       ADDD   #15
SKP13       SYNC
            SYNC
            SYNC
            ADDD   R2
            BCS    SKP14
            CMPD   #65521
            BLO    SKP15
SKP14       ADDD   #15
*
SKP15       STD    MCND
            CLR    ,U
            CLR    1,U
            LDA    1,X
            LDB    1,Y
            MUL
            STD    2,U
            LDA    ,X
            LDB    1,Y
            MUL
            ADDD   1,U
            STD    1,U
            BCC    SKP16
            INC    ,U
SKP16       LDA    1,X
            LDB    ,Y
            MUL
            ADDD   1,U
            STD    1,U
            BCC    SKP19
            INC    ,U
SKP19       LDA    ,X
            LDB    ,Y
            MUL
            ADDD   ,U
            STD    ,U
*
            LDA    1,U
            LDB    #15
            MUL
            ADDD   2,U
            BCS    SKP20
            CMPD   #65521
            BLO    SKP21
SKP20       ADDD   #15
SKP21       STD    2,U
*
            LDA    ,U
            LDX    #TEMP
            CLR    ,X
            CLR    1,X
            CLR    2,X
            LDB    #15
            MUL
            STD    ,X
            LDA    ,X
            LDB    #15
            MUL
            ADDD   1,X
            ADDD   2,U
```

```
        BCS     SKP22
        CMPD    #65521
        BLO     SKP23
SKP22   ADDD    #15
SKP23   SYNC
*
        STD     T2
        SYNC
        SYNC
        SYNC
        STD     T6
        SYNC
        SYNC
*
        STD     SAVE
        LDA     FLAG
        CMPA    #1
        BEQ     MULT
        CMPA    #2
        BEQ     CONV
        LDD     SAVE
        STD     RES
        LBRA    BEGIN
CONV    LDD     SAVE
        STD     OUTPUT
        LBRA    BEGIN
*
MULT    INC     FLAG
        LDX     #SAVE
        LDY     #RES
        CLR     ,U
        CLR     1,U
        LDA     1,X
        LDB     1,Y
        MUL
        STD     2,U
        LDA     ,X
        LDB     1,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP16
        INC     ,U
LOP16   LDA     1,X
        LDB     ,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP19
        INC     ,U
LOP19   LDA     ,X
        LDB     ,Y
        MUL
        ADDD    ,U
        STD     ,U
        LDA     1,U
        LDB     #15
        MUL
        ADDD    2,U
        BCS     LOP20
        CMPD    #65521
        BLO     LOP21
LOP20   ADDD    #15
LOP21   STD     2,U
*
        LDA     ,U
        LDX     #TEMP
        CLR     ,X
        CLR     1,X
        CLR     2,X
        LDB     #15
        MUL
        STD     ,X
        LDA     ,X
        LDB     #15
        MUL
        ADDD    1,X
        ADDD    2,U
        BCS     LOP22
        CMPD    #65521
        BLO     LOP23
LOP22   ADDD    #15
LOP23   STD     SAVE
        SYNC
        SYNC
        SYNC
        SYNC
        LBRA    NEXT
*
MLTER   FDB     1
MLTRR   FDB     61153
*
        ORG     $0000
MCND    FDB     0
PROD1   FCB     0
PROD2   FCB     0
PROD3   FCB     0
PROD4   FCB     0
TEMP    FCB     0
TEMP1   FCB     0
TEMP2   FCB     0
SAVE    FDB     0
FLAG    FCB     0
RES     FDB     0
*
        ORG     $FFFE
STRT    EQU     $F800
        END     BEGIN
```

```
*       ************************************************************************
*       *                    PROCESSOR NUMBER 2                                *
*       ************************************************************************
            NAM   68092          |SKP14    ADDD  #15
OUTPUT      EQU   $0400          |SKP15    STD   T3
STATUS      EQU   $0402          |         SYNC
T7          EQU   $0403          |         ADDD  R3
T5          EQU   $0405          |         BCS   SKP16
T3          EQU   $0407          |         CMPD  #65521
T1          EQU   $0409          |         BLO   SKP17
INPUT       EQU   $0410          |SKP16    ADDD  #15
R7          EQU   $0412          |SKP17    STD   T1
R5          EQU   $0414          |         SYNC
R3          EQU   $0416          |         STD   MCND
R1          EQU   $0418          |         CLR    ,U
SEM         EQU   $041A          |         CLR   1,U
*                                |         LDA   1,X
            ORG    $F800         |         LDB   1,Y
            NOP                  |         MUL
            ORCC  #%01010000     |         STD   2,U
            LDU   #PROD1         |         LDA    ,X
BEGIN       CLRA                 |         LDB   1,Y
            STA   FLAG           |         MUL
            LDA   SEM            |         ADDD  1,U
            BEQ   FRD            |         STD   1,U
START       LDA   #1             |         BCC   SKP18
            STA   FLAG           |         INC    ,U
FRD         LDY   #MCND          |SKP18    LDA   1,X
            LDX   #MLTPR         |         LDB    ,Y
            LDA   #1             |         MUL
            STA   STATUS         |         ADDD  1,U
            SYNC                 |         STD   1,U
            CLRA                 |         BCC   SKP21
            STA   STATUS         |         INC    ,U
            LDD   INPUT          |SKP21    LDA    ,X
            BRA   OVER           |         LDB    ,Y
NEXT        LDY   #MCND          |         MUL
            LDX   #MLTRR         |         ADDD   ,U
            SYNC                 |         STD    ,U
            LDD   SAVE           |*
*                                |         LDA   1,U
OVER        SYNC                 |         LDB   #15
            SYNC                 |         MUL
            ADDD  R7             |         ADDD  2,U
            BCS   SKP12          |         BCS   SKP22
            CMPD  #65521         |         CMPD  #65521
            BLO   SKP13          |         BLO   SKP23
SKP12       ADDD  #15            |SKP22    ADDD  #15
SKP13       STD   T5             |SKP23    STD   2,U
            SYNC                 |*
            ADDD  R5             |         LDA    ,U
            BCS   SKP14          |         LDX   #TEMP
            CMPD  #65521         |         CLR    ,X
            BLO   SKP15          |         CLR   1,X
```

```
        CLR   2,X
        LDB   #15
        MUL
        STD    ,X
        LDA    ,X
        LDB   #15
        MUL
        ADDD  1,X
        ADDD  2,U
        BCS   SKP24
        CMPD  #65521
        BLO   SKP25
SKP24   ADDD  #15
SKP25   SYNC
*
        SYNC
        ADDD  R1
        BCS   SKP26
        CMPD  #65521
        BLO   SKP27
SKP26   ADDD  #15
SKP27   STD   T3
        SYNC
        ADDD  R3
        BCS   SKP28
        CMPD  #65521
        BLO   SKP29
SKP28   ADDD  #15
SKP29   STD   T5
        SYNC
        ADDD  R5
        BCS   SKP30
        CMPD  #65521
        BLO   SKP31
SKP30   ADDD  #15
SKP31   STD   T7
        SYNC
        SYNC
*
        STD    SAVE
        LDA    FLAG
        CMPA   #1
        BEQ    MULT
        CMPA   #2
        BEQ    CONV
        LDD    SAVE
        STD    RES
        LBRA   BEGIN
CONV    LDD    SAVE
        STD    OUTPUT
        LBRA   BEGIN
*
MULT    INC    FLAG
        LDX    #SAVE
        LDY    #RES
LOP15   CLR    ,U
        CLR   1,U
        LDA   1,X
        LDB   1,Y
        MUL
        STD   2,U
        LDA    ,X
        LDB   1,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   LOP16
        INC    ,U
LOP16   LDA   1,X
        LDB    ,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   LOP19
        INC    ,U
LOP19   LDA    ,X
        LDB    ,Y
        MUL
        ADDD   ,U
        STD    ,U
*
        LDA   1,U
        LDB   #15
        MUL
        ADDD  2,U
        BCS   LOP20
        CMPD  #65521
        BLO   LOP21
LOP20   ADDD  #15
LOP21   STD   2,U
*
        LDA    ,U
        LDX   #TEMP
        CLR    ,X
        CLR   1,X
        CLR   2,X
        LDB   #15
        MUL
        STD    ,Y
        LDA    ,X
        LDB   #15
        MUL
        ADDD  1,Y
        ADDD  2,U
        BCS   LOP22
        CMPD  #65521
        BLO   LOP23
LOP22   ADDD  #15
```

```
*                                   |           ORG     $0000
LUP23     STD     T3                |MCND       FDB     0
          SYNC                      |PROD1      FCB     0
          LDD     R3                |PROD2      FCB     0
          STD     T5                |PROD3      FCB     0
          SYNC                      |PROD4      FCB     0
          SYNC                      |TEMP       FCB     0
          SYNC                      |TEMP1      FCB     0
          LDD     R3                |TEMP3      FCB     0
          STD     SAVE              |SAVE       FDB     0
          LBRA    NEXT              |FLAG       FCB     0
*                                   |RES        FDB     0
MLTFR     FDB     16379             |           ORG     $FFFE
MLTRR     FDB      5460             |START      EQU     $F800
*                                   |           END     BEGIN
*     ********************************************************************************
*     *                        PROCESSOR NUMBER 3                                    *
*     ********************************************************************************
          NAM     68093             |OVER       SYNC
          EQU     $0400             |           SYNC
STATUS    EQU     $0402             |           ADDD    R8
T8        EQU     $0403             |           BCS     SKP12
T4        EQU     $0405             |           CMPD    #65521
T2        EQU     $0407             |           BLO     SKP13
INPUT     EQU     $0410             |SKP12      ADDD    #15
R8        EQU     $0412             |SKP13      STD     T4
R4        EQU     $0414             |           SYNC
R2        EQU     $0416             |           ADDD    R4
SEM       EQU     $0418             |           BCS     SKP14
*                                   |           CMPD    #65521
          ORG      $F800            |           BLO     SKP15
          NOP                       |SKP14      ADDD    #15
          ORCC    #%01010000        |SKP15      STD     T2
          LDU     #PROD1            |           SYNC
BEGIN     CLRA                      |           STD     SAVE
          STA     FLAG              |           LDD     R2
          LDA     SEM               |           SUBD    SAVE
          BEQ     FRD               |           BCC     SKP16
START     LDA     #1                |           ADDD    #65521
          STA     FLAG              |SKP16      SYNC
FRD       LDY     #MCND             |*
          LDX     #MLTFR            |           STD     MCND
          LDA     #1                |           CLR      ,U
          STA     STATUS            |           CLR     1,U
          SYNC                      |           LDA     1,X
          CLRA                      |           LDB     1,Y
          STA     STATUS            |           MUL
          LDD     INPUT             |           STD     2,U
          BRA     OVER              |           LDA      ,X
NEXT      LDY     #MCND             |           LDB     1,Y
          LDX     #MLTRR            |           MUL
          SYNC                      |           ADDD    1,U
          LDD     SAVE              |           STD     1,U
*                                   |           BCC     SKP18
```

```
         INC    ,U
SKP18    LDA    1,X
         LDB    ,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    SKP21
         INC    ,U
SKP21    LDA    ,X
         LDB    ,Y
         MUL
         ADDD   ,U
         STD    ,U
*
         LDA    1,U
         LDB    #15
         MUL
         ADDD   2,U
         BCS    SKP22
         CMPD   #65521
         BLO    SKP23
SKP22    ADDD   #15
SKP23    STD    2,U
*
         LDA    ,U
         LDX    #TEMP
         CLR    ,X
         CLR    1,X
         CLR    2,X
         LDB    #15
         MUL
         STD    ,X
         LDA    ,X
         LDB    #15
         MUL
         ADDD   1,X
         ADDD   2,U
         BCS    SKP24
         CMPD   #65521
         BLO    SKP25
SKP24    ADDD   #15
SKP25    SYNC
*
         SYNC
         STD    T2
         SYNC
         STD    SAVE
         LDD    R2
         SUBD   SAVE
         BCC    SKP26
         ADDD   #65521
SKP26    STD    T4
         SYNC
         ADDD   R4
         BCS    SKP28
         CMPD   #65521
         BLO    SKP29
SKP28    ADDD   #15
SKP29    STD    T8
         SYNC
         SYNC
*
         STD    SAVE
         LDA    FLAG
         CMPA   #1
         BEQ    MULT
         CMPA   #2
         BEQ    CONV
         LDD    SAVE
         STD    RES
         LBRA   BEGIN
CONV     LDD    SAVE
         STD    OUTPUT
         LBRA   BEGIN
*
MULT     INC    FLAG
         LDX    #SAVE
         LDY    #RES
LOP15    CLR    ,U
         CLR    1,U
         LDA    1,X
         LDB    1,Y
         MUL
         STD    2,U
         LDA    ,X
         LDB    1,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    LOP16
         INC    ,U
LOP16    LDA    1,X
         LDB    ,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    LOP19
         INC    ,U
LOP19    LDA    ,X
         LDB    ,Y
         MUL
         ADDD   ,U
         STD    ,U
*
         LDA    1,U
         LDB    #15
         MUL
         ADDD   2,U
```

```
          BCS    LOP20          |          SYNC
          CMPD   #65521         |          SYNC
          BLO    LOP21          |          LDD    R4
LOP20     ADDD   #15            |          STD    T2
LOP21     STD    2,U            |          SYNC
*                                |          LBRA   NEXT
          LDA     ,U            |*
          LDX    #TEMP          |MLTER     FDB    13376
          CLR     ,X            |MLTRR     FDB    13364
          CLR    1,X            |*
          CLR    2,X            |          ORG     $0000
          LDB    #15            |MCND      FDB    0
          MUL                   |PROD1     FCB    0
          STD     ,X            |PROD2     FCB    0
          LDA     ,X            |PROD3     FCB    0
          LDB    #15            |PROD4     FCB    0
          MUL                   |TEMP      FCB    0
          ADDD   1,X            |TEMP1     FCB    0
          ADDD   2,U            |TEMP3     FCB    0
          BCS    LOP22          |SAVE      FDB    0
          CMPD   #65521         |FLAG      FCB    0
          BLO    LOP23          |RES       FDB    0
LOP22     ADDD   #15            |*
LOP23     STD    T2             |          ORG    $EEEE
          SYNC                  |STRT      EQU    $F800
          LDD    R2             |          END    BEGIN
          STD    SAVE           |*
*     ************************************************************************
*     *                        PROCESSOR NUMBER 4                            *
*     ************************************************************************
          NAM    68094          |          LDA    #1
OUTPUT    EQU    $0400          |          STA    STATUS
STATUS    EQU    $0402          |          SYNC
T9        EQU    $0403          |          CLRA
T3        EQU    $0405          |          STA    STATUS
T16       EQU    $0407          |          LDD    INPUT
INPUT     EQU    $0410          |          BRA    OVER
R9        EQU    $0412          |NEXT      LDY    #MCND
R3        EQU    $0414          |          LDX    #MLTRR
R16       EQU    $0416          |          SYNC
SEM       EQU    $0418          |          LDD    SAVE
*                                |*
          ORG     $F800         |OVER      SYNC
          NOP                   |          SYNC
          ORCC   #%01010000     |          ADDD   R9
          LDU    #PROD1         |          BCS    SKP12
BEGIN     CLRA                  |          CMPD   #65521
          STA    FLAG           |          BLO    SKP13
          LDA    SEM            |SKP12     ADDD   #15
          BEQ    FRD            |SKP13     STD    T3
START     LDA    #1             |          SYNC
          STA    FLAG           |          SUBD   R3
FRD       LDY    #MCND          |          BCC    SKP14
          LDX    #MLTER         |          ADDD   #65521
```

```
SKP14   STD   T16
        SYNC
        SYNC
*
        STD   MCND
        CLR    ,U
        CLR   1,U
        LDA   1,X
        LDB   1,Y
        MUL
        STD   2,U
        LDA    ,X
        LDB   1,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKP16
        INC    ,U
SKP16   LDA   1,X
        LDB    ,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKP19
        INC    ,U
SKP19   LDA    ,X
        LDB    ,Y
        MUL
        ADDD   ,U
        STD    ,U
*
        LDA   1,U
        LDB   #15
        MUL
        ADDD  2,U
        BCS   SKP20
        CMPD  #65521
        BLO   SKP21
SKP20   ADDD  #15
SKP21   STD   2,U
*
        LDA    ,U
        LDX   #TEMP
        CLR    ,X
        CLR   1,X
        CLR   2,X
        LDB   #15
        MUL
        STD    ,X
        LDA    ,X
        LDB   #15
        MUL
        ADDD  1,X
        ADDD  2,U
        BCS   SKP22
        CMPD  #65521
        BLO   SKP23
SKP22   ADDD  #15
SKP23   SYNC
*
        SYNC
        ADDD  P16
        BCS   SKP24
        CMPD  #65521
        BLO   SKP25
SKP24   ADDD  #15
SKP25   SYNC
        STD   T3
        SYNC
        STD   SAVE
        LDD   R3
        SUBD  SAVE
        BCC   SKP26
        ADDD  #65521
SKP26   STD   T9
        SYNC
        SYNC
*
        STD    SAVE
        LDA    FLAG
        CMPA   #1
        BEQ    MULT
        CMPA   #2
        BEQ    CONV
        LDD    SAVE
        STD    RES
        LBRA   BEGIN
CONV    LDD    SAVE
        STD    OUTPUT
        LBRA   BEGIN
*
MULT    INC    FLAG
        LDX    #SAVE
        LDY    #RES
LOP15   CLR    ,U
        CLR   1,U
        LDA   1,X
        LDB   1,Y
        MUL
        STD   2,U
        LDA    ,X
        LDB   1,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   LOP16
        INC    ,U
LOP16   LDA   1,X
```

```
          LDB     ,Y              |          ADDD    2,U
          MUL                     |          BCS     LOP22
          ADDD    1,U             |          CMPD    #65521
          STD     1,U             |          BLO     LOP23
          BCC     LOP19           |LOP22     ADDD    #15
          INC     ,U              |LOP23     SYNC
LOP19     LDA     ,X              |          SYNC
          LDB     ,Y              |          STD     T3
          MUL                     |          LDD     R16
          ADDD    ,U              |          STD     SAVE
          STD     ,U              |          SYNC
*                                 |          SYNC
          LDA     1,U             |          LSRA    NEXT
          LDB     #15             |*
          MUL                     |MLTER     FDB     48647
          ADDD    -2,U            |MLTRR     FDB      5493
          BCS     LOP20           |*
          CMPD    #65521          |*
          BLO     LOP21           |          ORG     $0000
LOP20     ADDD    #15             |MCND      FDB     0
LOP21     STD     2,U             |PROD1     FCB     0
*                                 |PROD2     FCB     0
          LDA     ,U              |PROD3     FCB     0
          LDX     #TEMP           |PROD4     FCB     0
          CLR     ,X              |TEMP      FCB     0
          CLR     1,X             |TEMP1     FCB     0
          CLR     2,X             |TEMP3     FCB     0
          LDB     #15             |SAVE      FDB     0
          MUL                     |FLAG      FCB     0
          STD     ,X              |RES       FDB     0
          LDA     ,X              |*
          LDB     #15             |          ORG     $FFFE
          MUL                     |STRT      EQU     $F800
          ADDD    1,X             |          END     BEGIN
*     ************************************************************************
*     *                        PROCESSOR NUMBER 5                            *
*     ************************************************************************
          NAM     68095           |          STA     FLAG
OUTPUT    EQU     $0400           |          LDA     SEM
STATUS    EQU     $0402           |          BEQ     FRD
T10       EQU     $0403           |START     LDA     #1
T2        EQU     $0405           |          STA     FLAG
T16       EQU     $0407           |FRD       LDY     #MCND
INPUT     EQU     $0410           |          LDX     #MLTER
R10       EQU     $0412           |          LDA     #1
R2        EQU     $0414           |          STA     STATUS
R16       EQU     $0416           |          SYNC
SEM       EQU     $0418           |          CLRA
*                                 |          STA     STATUS
          ORG     $F800           |          LDD     INPUT
          NOP                     |          BRA     OVER
          ORCC    #%01010000      |NEXT      LDY     #MCND
          LDU     #PROD1          |          LDX     #MLTRR
BEGIN     CLRA                    |          SYNC
```

```
         LDD    SAVE
*
OVER     SYNC
         SYNC
         ADDD   R10
         BCS    SKP12
         CMPD   #65521
         BLO    SKP13
SKP12    ADDD   #15
SKP13    STD    T2
         SYNC
         STD    SAVE
         LDD    R2
         SUBD   SAVE
         BCC    SKP14
         ADDD   #65521
SKP14    STD    T16
         SYNC
         SYNC
*
         STD    MCND
         CLR     ,U
         CLR    1,U
         LDA    1,X
         LDB    1,Y
         MUL
         STD    2,U
         LDA     ,X
         LDB    1,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    SKP16
         INC     ,U
SKP16    LDA    1,X
         LDB     ,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    SKP19
         INC     ,U
SKP19    LDA     ,X
         LDB     ,Y
         MUL
         ADDD    ,U
         STD     ,U
*
         LDA    1,U
         LDB    #15
         MUL
         ADDD   2,U
         BCS    SKP20
         CMPD   #65521
         BLO    SKP21
SKP20    ADDD   #15
SKP21    STD    2,U
*
         LDA     ,U
         LDX    #TEMP
         CLR     ,X
         CLR    1,X
         CLR    2,Y
         LDB    #15
         MUL
         STD     ,X
         LDA     ,X
         LDB    #15
         MUL
         ADDD   1,Y
         ADDD   2,U
         BCS    SKP22
         CMPD   #65521
         BLO    SKP23
SKP22    ADDD   #15
SKP23    SYNC
*
         SYNC
         SUBD   R16
         BCC    SKP24
         ADDD   #65521
SKP24    SYNC
         STD    T2
         SYNC
         STD    SAVE
         LDD    R2
         SUBD   SAVE
         BCC    SKP26
         ADDD   #65521
SKP26    STD    T10
         SYNC
         SYNC
*
         STD     SAVE
         LDA     FLAG
         CMPA    #1
         BEQ     MULT
         CMPA    #2
         BEQ     CONV
         LDD     SAVE
         STD     RES
         LBRA    BEGIN
CONV     LDD     SAVE
         STD     OUTPUT
         LBRA    BEGIN
*
MULT     INC     FLAG
         LDX     #SAVE
         LDY     #RES
```

```
LOP15   CLR    ,U          |        LDB   #15
        CLR   1,U          |        MUL
        LDA   1,X          |        STD    ,X
        LDB   1,Y          |        LDA    ,X
        MUL                |        LDB   #15
        STD   2,U          |        MUL
        LDA    ,X          |        ADDD  1,X
        LDB   1,Y          |        ADDD  2,U
        MUL                |        BCS   LOP22
        ADDD  1,U          |        CMPD  #65521
        STD   1,U          |        BLO   LOP23
        BCC   LOP16        |LOP22   ADDD  #15
        INC    ,U          |LOP23   STD   T16
LOP16   LDA   1,X          |        SYNC
        LDB    ,Y          |        SYNC
        MUL                |        LDD   R2
        ADDD  1,U          |        STD   SAVE
        STD   1,U          |        SYNC
        BCC   LOP19        |        SYNC
        INC    ,U          |        LBRA  NEXT
LOP19   LDA    ,X          |************************
        LDB    ,Y          |MLTFR   FDB   19136
        MUL                |MLTRR   FDB   46773
        ADDD   ,U          |*
        STD    ,U          |        ORG   $0000
*                          |MCND    FDB   0
        LDA   1,U          |PROD1   FCB   0
        LDB   #15          |PROD2   FCB   0
        MUL                |PROD3   FCB   0
        ADDD  2,U          |PROD4   FCB   0
        BCS   LOP20        |TEMP    FCB   0
        CMPD  #65521       |TEMP1   FCB   0
        BLO   LOP21        |TEMP3   FCB   0
LOP20   ADDD  #15          |SAVE    FDB   0
LOP21   STD   2,U          |FLAG    FCB   0
*                          |RES     FDB   0
        LDA    ,U          |*
        LDX   #TEMP        |        ORG   $FFFE
        CLR    ,X          |STRT    EQU   $F800
        CLR   1,X          |        END   BEGIN
        CLR   2,X          |*
*     ***************************************************************************
*     *                    PROCESSOR NUMBER 6                                   *
*     ***************************************************************************
        NAM   68096        |SEM     EQU   $0418
OUTPUT  EQU   $0400         |*
STATUS  EQU   $0402         |        ORG    $F800
T11     EQU   $0403         |        NOP
T1      EQU   $0405         |        ORCC   #%01010000
T7      EQU   $0407         |        LDU    #PROD1
INPUT   EQU   $0410         |BEGIN   CLRA
R11     EQU   $0412         |        STA    FLAG
R1      EQU   $0414         |        LDA    SEM
R7      EQU   $0416         |        BEQ    FDD
```

```
START   LDA    #1          |         INC    ,U
        STA    FLAG        |SKP19    LDA    ,X
FRD     LDY    #MCND       |         LDB    ,Y
        LDX    #MLTPR      |         MUL
        LDA    #1          |         ADDD   ,U
        STA    STATUS      |         STD    ,U
        SYNC               |*
        CLRA               |         LDA    1,U
        STA    STATUS      |         LDB    #15
        LDD    INPUT       |         MUL
        BRA    OVER        |         ADDD   2,U
NEXT    LDY    #MCND       |         BCS    SKP20
        LDX    #MLTPR      |         CMPD   #65521
        SYNC               |         BLO    SKP21
        LDD    SAVE        |SKP20    ADDD   #15
*                          |SKP21    STD    2,U
OVER    STD    T11         |*
        SYNC               |         LDA    ,U
        ADDD   R11         |         LDX    #TEMP
        BCS    SKP12       |         CLR    ,X
        CMPD   #65521      |         CLR    1,X
        BLO    SKP13       |         CLR    2,X
SKP12   ADDD   #15         |         LDB    #15
SKP13   STD    T1          |         MUL
        SYNC               |         STD    ,X
        SYNC               |         LDA    ,X
        SYNC               |         LDB    #15
        SYNC               |         MUL
        ADDD   R7          |         ADDD   1,X
        BCS    SKP14       |         ADDD   2,U
        CMPD   #65521      |         BCS    SKP22
        BLO    SKP15       |         CMPD   #65521
SKP14   ADDD   #15         |         BLO    SKP23
*                          |SKP22    ADDD   #15
SKP15   STD    MCND        |SKP23    SYNC
        CLR    ,U          |*
        CLR    1,U         |         STD    T7
        LDA    1,X         |         SYNC
        LDB    1,Y         |         SYNC
        MUL                |         SYNC
        STD    2,U         |         SYNC
        LDA    ,X          |         ADDD   R1
        LDB    1,Y         |         BCS    SKP24
        MUL                |         CMPD   #65521
        ADDD   1,U         |         BLO    SKP25
        STD    1,U         |SKP24    ADDD   #15
        BCC    SKP16       |SKP25    STD    T11
        INC    ,U          |         SYNC
SKP16   LDA    1,X         |         ADDD   R11
        LDB    ,Y          |         BCS    SKP26
        MUL                |         CMPD   #65521
        ADDD   1,U         |         BLO    SKP27
        STD    1,U         |SKP26    ADDD   #15
        BCC    SKP19       |*
```

```
SKP27   STD     SAVE
        LDA     FLAG
        CMPA    #1
        BEQ     MULT
        CMPA    #2
        BEQ     CONV
        LDD     SAVE
        STD     RES
        LBRA    BEGIN
CONV    LDD     SAVE
        STD     OUTPUT
        LBRA    BEGIN
*
MULT    INC     FLAG
        LDX     #SAVE
        LDY     #RES
LOP15   CLR     ,U
        CLR     1,U
        LDA     1,X
        LDB     1,Y
        MUL
        STD     2,U
        LDA     ,X
        LDB     1,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP16
        INC     ,U
LOP16   LDA     1,X
        LDB     ,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP19
        INC     ,U
LOP19   LDA     ,X
        LDB     ,Y
        MUL
        ADDD    ,U
        STD     ,U
*
        LDA     1,U
        LDB     #15
        MUL
        ADDD    2,U
        BCS     LOP20
        CMPD    #65521
        BLO     LOP21
*
*
*
*
*
LOP20   ADDD    #15
LOP21   STD     2,U
*
        LDA     ,U
        LDX     #TEMP
        CLR     ,X
        CLR     1,X
        CLR     2,X
        LDB     #15
        MUL
        STD     ,X
        LDA     ,X
        LDB     #15
        MUL
        ADDD    1,X
        ADDD    2,U
        BCS     LOP22
        CMPD    #65521
        BLO     LOP23
LOP22   ADDD    #15
LOP23   STD     T11
        SYNC
        LDD     P11
        STD     SAVE
        SYNC
        SYNC
        SYNC
        LBRA    NEXT
*************************
MLTPR   FDB     32759
MLTRR   FDB      6552
*
        ORG     $0000
MCND    FDB     0
PROD1   FCB     0
PROD2   FCB     0
PROD3   FCB     0
PROD4   FCB     0
TEMP    FCB     0
TEMP1   FCB     0
TEMP3   FCB     0
SAVE    FDB     0
FLAG    FCB     0
RES     FDB     0
*
        ORG     $FFFF
STRT    EQU     $F800
        END     BEGIN
*
*
*
*
*
*
```

```
*       ************************************************************************
*       *                        PROCESSOR NUMBER 7                            *
*       ************************************************************************
          NAM   68097
OUTPUT    EQU   $0400
STATUS    EQU   $0402
T12       EQU   $0403
T2        EQU   $0405
T10       EQU   $0407
T8        EQU   $0409
T6        EQU   $040B
INPUT     EQU   $0410
R12       EQU   $0412
R2        EQU   $0414
R10       EQU   $0416
R8        EQU   $0418
R6        EQU   $041A
SEM       EQU   $041C
*
          ORG    $F800
          NOP
          ORCC   #%01010000
          LDU    #PROD1
BEGIN     CLRA
          STA    FLAG
          LDA    SEM
          BEQ    FRD
START     LDA    #1
          STA    FLAG
FRD       LDY    #MCND
          LDX    #MLTFR
          LDA    #1
          STA    STATUS
          SYNC
          CLRA
          STA    STATUS
          LDD    INPUT
          BRA    OVER
NEXT      LDY    #MCND
          LDX    #MLTRR
          SYNC
          LDD    SAVE
*
OVER      STD    T12
          SYNC
          ADDD   R12
          BCS    SKP12
          CMPD   #65521
          BLO    SKP13
SKP12     ADDD   #15
SKP13     STD    T2
          SYNC
          STD    T10
          SYNC
          ADDD   R10
          BCS    SKP14
          CMPD   #65521
          BLO    SKP15
SKP14     ADDD   #15
SKP15     STD    T8
          SYNC
          ADDD   R8
          BCS    SKP16
          CMPD   #65521
          BLO    SKP17
SKP16     ADDD   #15
SKP17     STD    T6
          SYNC
*
          STD    MCND
          CLR    ,U
          CLR    1,U
          LDA    1,X
          LDB    1,Y
          MUL
          STD    2,U
          LDA    ,X
          LDB    1,Y
          MUL
          ADDD   1,U
          STD    1,U
          BCC    SKP18
          INC    ,U
SKP18     LDA    1,X
          LDB    ,Y
          MUL
          ADDD   1,U
          STD    1,U
          BCC    SKP21
          INC    ,U
SKP21     LDA    ,X
          LDB    ,Y
          MUL
          ADDD   ,U
          STD    ,U
*
          LDA    1,U
          LDB    #15
          MUL
          ADDD   2,U
          BCS    SKP22
          CMPD   #65521
          BLO    SKP23
SKP22     ADDD   #15
SKP23     STD    2,U
```

```
*
          LDA    ,U
          LDX    #TEMP
          CLR    ,X
          CLR    1,X
          CLR    2,X
          LDB    #15
          MUL
          STD    ,X
          LDA    ,X
          LDB    #15
          MUL
          ADDD   1,X
          ADDD   2,U
          BCS    SKP24
          CMPD   #65521
          BLO    SKP25
SKP24     ADDD   #15
SKP25     SYNC
*
          SYNC
          ADDD   R6
          BCS    SKP26
          CMPD   #65521
          BLO    SKP27
SKP26     ADDD   #15
SKP27     STD    T8
          SYNC
          ADDD   R8
          BCS    SKP28
          CMPD   #65521
          BLO    SKP29
SKP28     ADDD   #15
SKP29     STD    T10
          SYNC
          ADDD   R10
          BCS    SKP30
          CMPD   #65521
          BLO    SKP31
SKP30     ADDD   #15
SKP31     SYNC
          ADDD   R2
          BCS    SKP32
          CMPD   #65521
          BLO    SKP33
SKP32     ADDD   #15
SKP33     STD    T12
          SYNC
          ADDD   R12
          BCS    SKP34
          CMPD   #65521
          BLO    SKP35
SKP34     ADDD   #15
*
SKP35     STD    SAVE
          LDA    FLAG
          CMPA   #1
          BEQ    MULT
          CMPA   #2
          BEQ    CONV
          LDD    SAVE
          STD    RES
          LBRA   BEGIN
CONV      LDD    SAVE
          STD    OUTPUT
          LBRA   BEGIN
*
MULT      INC    FLAG
          LDX    #SAVE
          LDY    #RES
LOP15     CLR    ,U
          CLR    1,U
          LDA    1,Y
          LDB    1,Y
          MUL
          STD    2,U
          LDA    ,X
          LDB    1,Y
          MUL
          ADDD   1,U
          STD    1,U
          BCC    LOP16
          INC    ,U
LOP16     LDA    1,X
          LDB    ,Y
          MUL
          ADDD   1,U
          STD    1,U
          BCC    LOP19
          INC    ,U
LOP19     LDA    ,X
          LDB    ,Y
          MUL
          ADDD   ,U
          STD    ,U
*
          LDA    1,U
          LDB    #15
          MUL
          ADDD   2,U
          BCS    LOP20
          CMPD   #65521
          BLO    LOP21
LOP20     ADDD   #15
LOP21     STD    2,U
*
          LDA    ,U
          LDX    #TEMP
```

```
        CLR     ,X
        CLR     1,X
        CLR     2,X
        LDB     #15
        MUL
        STD     ,X
        LDA     ,X
        LDB     #15
        MUL
        ADDD    1,X
        ADDD    2,U
        BCS     LOP22
        CMPD    #65521
        BLO     LOP23
LOP22   ADDD    #15
LOP23   STD     T12
        SYNC
        LDD     R12
        STD     T8
        SYNC
        LDD     R8
        STD     T12
        SYNC
        LDD     R12
        STD     SAVE
        SYNC
        LBRA    NEXT
*
MLTFR   FDB     8192
MLTRR   FDB     57331
*
        ORG     $0000
MCND    FDB     0
PROD1   FCB     0
PROD2   FCB     0
PROD3   FCB     0
PROD4   FCB     0
TEMP    FCB     0
TEMP1   FCB     0
TEMP3   FCB     0
SAVE    FDB     0
FLAG    FCB     0
RES     FDB     0
*
        ORG     $FFFE
STRT    EQU     $F800
        END     BEGIN
*
*****************************************************************
*                     PROCESSOR NUMBER 8                        *
*****************************************************************
        NAM     68098
OUTPUT  EQU     $0400
STATUS  EQU     $0402
T13     EQU     $0403
T3      EQU     $0405
T9      EQU     $0407
T7      EQU     $0409
INPUT   EQU     $0410
R13     EQU     $0412
R3      EQU     $0414
R9      EQU     $0416
R7      EQU     $0418
SEM     EQU     $041A
*
        ORG     $F800
        NOP
        ORCC    #%01010000
        LDU     #PROD1
BEGIN   CLRA
        STA     FLAG
        LDA     SEM
        BEQ     FRD
START   LDA     #1
        STA     FLAG
FRD     LDY     #MCND
        LDX     #MLTFR
        LDA     #1
        STA     STATUS
        SYNC
        CLRA
        STA     STATUS
        LDD     INPUT
        BRA     OVER
NEXT    LDY     #MCND
        LDX     #MLTRR
        SYNC
        LDD     SAVE
*
OVER    STD     T13
        SYNC
        ADDD    R13
        BCS     SKP12
        CMPD    #65521
        BLO     SKP13
SKP12   ADDD    #15
SKP13   STD     T3
        SYNC
        STD     T9
        SYNC
        ADDD    R9
        BCS     SKP14
        CMPD    #65521
        BLO     SKP15
SKP14   ADDD    #15
```

```
SKP15   STD   T7
        SYNC
        STD   SAVE
        LDD   R7
        SUBD  SAVE
        BCC   SKP16
        ADDD  #65521
SKP16   SYNC
*
        STD   MCND
        CLR    ,U
        CLR   1,U
        LDA   1,X
        LDB   1,Y
        MUL
        STD   2,U
        LDA    ,X
        LDB   1,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKP18
        INC    ,U
SKP18   LDA   1,X
        LDB    ,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKP21
        INC    ,U
SKP21   LDA    ,X
        LDB    ,Y
        MUL
        ADDD   ,U
        STD    ,U
*
        LDA   1,U
        LDB   #15
        MUL
        ADDD  2,U
        BCS   SKP22
        CMPD  #65521
        BLO   SKP23
SKP22   ADDD  #15
SKP23   STD   2,U
*
        LDA    ,U
        LDX   #TEMP
        CLR    ,X
        CLR   1,X
        CLR   2,X
        LDB   #15
        MUL
        STD    ,X
        LDA    ,X
        LDB   #15
        MUL
        ADDD  1,X
        ADDD  2,U
        BCS   SKP24
        CMPD  #65521
        BLO   SKP25
SKP24   ADDD  #15
SKP25   SYNC
*
        SYNC
        STD   T7
        SYNC
        STD   SAVE
        LDD   R7
        SUBD  SAVE
        BCC   SKP26
        ADDD  #65521
SKP26   STD   T9
        SYNC
        ADDD  R9
        BCS   SKP28
        CMPD  #65521
        BLO   SKP29
SKP28   ADDD  #15
SKP29   SYNC
        ADDD  R3
        BCS   SKP30
        CMPD  #65521
        BLO   SKP31
SKP30   ADDD  #15
SKP31   STD   T13
        SYNC
        ADDD  R13
        BCS   SKP32
        CMPD  #65521
        BLO   SKP33
SKP32   ADDD  #15
*
SKP33   STD    SAVE
        LDA    FLAG
        CMPA   #1
        BEQ    MULT
        CMPA   #2
        BEQ    CONV
        LDD    SAVE
        STD    RES
        LBRA   BEGIN
CONV    LDD    SAVE
        STD    OUTPUT
        LBRA   BEGIN
*
MULT    INC    FLAG
```

```
          LDX    #SAVE
          LDY    #RES
LOP15     CLR    ,U
          CLR    1,U
          LDA    1,X
          LDB    1,Y
          MUL
          STD    2,U
          LDA    ,X
          LDB    1,Y
          MUL
          ADDD   1,U
          STD    1,U
          BCC    LOP16
          INC    ,U
LOP16     LDA    1,X
          LDB    ,Y
          MUL
          ADDD   1,U
          STD    1,U
          BCC    LOP19
          INC    ,U
LOP19     LDA    ,X
          LDB    ,Y
          MUL
          ADDD   ,U
          STD    ,U
*
          LDA    1,U
          LDB    #15
          MUL
          ADDD   2,U
          BCS    LOP20
          CMPD   #65521
          BLO    LOP21
LOP20     ADDD   #15
LOP21     STD    2,U
*
          LDA    ,U
          LDX    #TEMP
          CLR    ,X
          CLR    1,X
          CLR    2,X
          LDB    #15
          MUL
          STD    ,X
          LDA    ,X
          LDB    #15
          MUL
          ADDD   1,X
          ADDD   2,U
          BCS    LOP22
          CMPD   #65521
          BLO    LOP23
LOP22     ADDD   #15
LOP23     STD    T13
          SYNC
          LDD    R9
          STD    T7
          LDD    R13
          STD    T9
          SYNC
          LDD    R7
          STD    SAVE
          SYNC
          SYNC
          LBRA    NEXT
*
MLTER     FDB    45457
MLTRR     FDB    37975
*
          ORG    $0000
MCND      FDB    0
PROD1     FCB    0
PROD2     FCB    0
PROD3     FCB    0
PROD4     FCB    0
TEMP      FCB    0
TEMP1     FCB    0
TEMP3     FCB    0
SAVE      FDB    0
FLAG      FCB    0
RES       FDB    0
*
          ORG    $FFFE
STRT      EQU    $F800
          END    BEGIN
*
*     ************************************************************
*     *                  PROCESSOR NUMBER 9                      *
*     ************************************************************
          NAM    68099
OUTPUT    EQU    $0400
STATUS    EQU    $0402
T14       EQU    $0403
T4        EQU    $0405
T8        EQU    $0407
T17       EQU    $0409
INPUT     EQU    $0410
R14       EQU    $0412
R4        EQU    $0414
R8        EQU    $0416
R17       EQU    $0418
SEM       EQU    $041A
*
```

```
INPUT   EQU   $0410
R14     EQU   $0412
R4      EQU   $0414
R8      EQU   $0416
R17     EQU   $0418
SEM     EQU   $041A
*
        ORG    $F800
        NOP
        ORCC   #%01010000
        LDU    #PROD1
BEGIN   CLRA
        STA    FLAG
        LDA    SEM
        BEQ    FRD
START   LDA    #1
        STA    FLAG
FRD     LDY    #MCND
        LDX    #MLTPR
        LDA    #1
        STA    STATUS
        SYNC
        CLRA
        STA    STATUS
        LDD    INPUT
        BRA    OVER
NEXT    LDY    #MCND
        LDX    #MLTRR
        SYNC
        LDD    SAVE
*
OVER    STD    T14
        SYNC
        ADDD   R14
        BCS    SKP12
        CMPD   #65521
        BLO    SKP13
SKP12   ADDD   #15
SKP13   STD    T4
        SYNC
        STD    T8
        SYNC
        SUBD   R8
        BCC    SKP14
        ADDD   #65521
SKP14   STD    T17
        SYNC
        SYNC
*
        STD    MCND
        CLR    ,U
        CLR    1,U
        LDA    1,X
        LDB    1,Y
        MUL
        STD    2,U
        LDA    ,X
        LDB    1,Y
        MUL
        ADDD   1,U
        STD    1,U
        BCC    SKP16
        INC    ,U
SKP16   LDA    1,X
        LDB    ,Y
        MUL
        ADDD   1,U
        STD    1,U
        BCC    SKP19
        INC    ,U
SKP19   LDA    ,X
        LDB    ,Y
        MUL
        ADDD   ,U
        STD    ,U
*
        LDA    1,U
        LDB    #15
        MUL
        ADDD   2,U
        BCS    SKP20
        CMPD   #65521
        BLO    SKP21
SKP20   ADDD   #15
SKP21   STD    2,U
*
        LDA    ,U
        LDX    #TEMP
        CLR    ,X
        CLR    1,X
        CLR    2,X
        LDB    #15
        MUL
        STD    ,X
        LDA    ,X
        LDB    #15
        MUL
        ADDD   1,X
        ADDD   2,U
        BCS    SKP22
        CMPD   #65521
        BLO    SKP23
SKP22   ADDD   #15
SKP23   SYNC
*
        SYNC
        ADDD   R17
        BCS    SKP24
```

```
        CMPD   #65521
        BLO    SKP25
SKP24   ADDD   #15
SKP25   SYNC
        STD    T8
        SYNC
        STD    SAVE
        LDD    R8
        SUBD   SAVE
        BCC    SKP26
        ADDD   #65521
SKP26   SYNC
        ADDD   R4
        BCS    SKP28
        CMPD   #65521
        BLO    SKP29
SKP28   ADDD   #15
SKP29   STD    T14
        SYNC
        ADDD   R14
        BCS    SKP30
        CMPD   #65521
        BLO    SKP31
SKP30   ADDD   #15
*
SKP31   STD     SAVE
        LDA     FLAG
        CMPA    #1
        BEQ     MULT
        CMPA    #2
        BEQ     CONV
        LDD     SAVE
        STD     RES
        LBRA    BEGIN
CONV    LDD     SAVE
        STD     OUTPUT
        LBRA    BEGIN
*
MULT    INC     FLAG
        LDX     #SAVE
        LDY     #RES
LOP15   CLR     ,U
        CLR     1,U
        LDA     1,X
        LDB     1,Y
        MUL
        STD     2,U
        LDA      ,X
        LDB     1,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP16
        INC      ,U
LOP16   LDA     1,X
        LDB      ,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP19
        INC      ,U
LOP19   LDA      ,X
        LDB      ,Y
        MUL
        ADDD     ,U
        STD      ,U
*
        LDA     1,U
        LDB     #15
        MUL
        ADDD    2,U
        BCS     LOP20
        CMPD    #65521
        BLO     LOP21
LOP20   ADDD    #15
LOP21   STD     2,U
*
        LDA      ,U
        LDX     #TEMP
        CLR      ,X
        CLR     1,X
        CLR     2,X
        LDB     #15
        MUL
        STD      ,X
        LDA      ,X
        LDB     #15
        MUL
        ADDD    1,X
        ADDD    2,U
        BCS     LOP22
        CMPD    #65521
        BLO     LOP23
LOP22   ADDD    #15
LOP23   STD     T8
        SYNC
        SYNC
        LDD     R17
        STD     T14
        LDD     R8
        STD     T17
        SYNC
        LDD     R14
        STD     SAVE
        SYNC
        LBRA    NEXT
*
MLTFR   FDB     25311
```

```
MLTRR    FDB   24521          |TEMP1    FCB   0
*                             |TEMP3    FCB   0
         ORG   $0000           |SAVE     FDB   0
MCND     FDB   0               |FLAG     FCB   0
PROD1    FCB   0               |RES      FDB   0
PROD2    FCB   0               |*
PROD3    FCB   0               |         ORG   $FFFE
PROD4    FCB   0               |STRT     EQU   *$F800
TEMP     FCB   0               |         END   BEGIN
*    ************************************************************************
*    *                    PROCESSOR NUMBER 10                                *
*    ************************************************************************
         NAM   680910          |         CMPD  #65521
OUTPUT   EQU   $0400           |         BLO   SKP13
STATUS   EQU   $0402           |SKP12    ADDD  #15
T15      EQU   $0403           |SKP13    STD   T5
T5       EQU   $0405           |         SYNC
T7       EQU   $0407           |         STD   T7
T17      EQU   $0409           |         SYNC
INPUT    EQU   $0410           |         STD   SAVE
R15      EQU   $0412           |         LDD   R7
R5       EQU   $0414           |         SUBD  SAVE
R7       EQU   $0416           |         BCC   SKP14
R17      EQU   $0418           |         ADDD  #65521
SEM      EQU   $041A           |SKP14    STD   T17
*                              |         SYNC
         ORG    $F800          |         SYNC
         NOP                   |*
         ORCC  #%01010000      |         STD   MCND
         LDU   #PROD1          |         CLR    ,U
BEGIN    CLRA                  |         CLR   1,U
         STA   FLAG            |         LDA   1,X
         LDA   SEM             |         LDB   1,Y
         BEQ   FRD             |         MUL
START    LDA   #1              |         STD   2,U
         STA   FLAG            |         LDA    ,X
FRD      LDY   #MCND           |         LDB   1,Y
         LDX   #MLTFR          |         MUL
         LDA   #1              |         ADDD  1,U
         STA   STATUS          |         STD   1,U
         SYNC                  |         BCC   SKP16
         CLRA                  |         INC    ,U
         STA   STATUS          |SKP16    LDA   1,X
         LDD   INPUT           |         LDB    ,Y
         BRA   OVER            |         MUL
NEXT     LDY   #MCND           |         ADDD  1,U
         LDX   #MLTRR          |         STD   1,U
         SYNC                  |         BCC   SKP19
         LDD   SAVE            |         INC    ,U
*                              |SKP19    LDA    ,X
OVER     STD   T15             |         LDB    ,Y
         SYNC                  |         MUL
         ADDD  R15             |         ADDD   ,U
         BCS   SKP12           |         STD    ,U
```

```
*
        LDA     1,U
        LDB     #15
        MUL
        ADDD    2,U
        BCS     SKP20
        CMPD    #65521
        BLO     SKP21
SKP20   ADDD    #15
SKP21   STD     2,U
*
        LDA      ,U
        LDX     #TEMP
        CLR      ,X
        CLR     1,X
        CLR     2,X
        LDB     #15
        MUL
        STD      ,X
        LDA      ,X
        LDB     #15
        MUL
        ADDD    1,X
        ADDD    2,U
        BCS     SKP22
        CMPD    #65521
        BLO     SKP23
SKP22   ADDD    #15
*
SKP23   SYNC
        SYNC
        SUBD    R17
        BCC     SKP24
        ADDD    #65521
SKP24   SYNC
        STD     T7
        SYNC
        STD     SAVE
        LDD     R7
        SUBD    SAVE
        BCC     SKP26
        ADDD    #65521
SKP26   SYNC
        ADDD    R5
        BCS     SKP28
        CMPD    #65521
        BLO     SKP29
SKP28   ADDD    #15
SKP29   STD     T15
        SYNC
        ADDD    R15
        BCS     SKP30
        CMPD    #65521
        BLO     SKP31
SKP30   ADDD    #15
*
SKP31   STD      SAVE
        LDA      FLAG
        CMPA     #1
        BEQ      MULT
        CMPA     #2
        BEQ      CONV
        LDD      SAVE
        STD      RES
        LBRA     BEGIN
CONV    LDD      SAVE
        STD      OUTPUT
        LBRA     BEGIN
*
MULT    INC      FLAG
        LDX      #SAVE
        LDY      #RES
LOP15   CLR      ,U
        CLR     1,U
        LDA     1,X
        LDB     1,Y
        MUL
        STD     2,U
        LDA      ,X
        LDB     1,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP16
        INC      ,U
LOP16   LDA     1,X
        LDB      ,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP19
        INC      ,U
LOP19   LDA      ,X
        LDB      ,Y
        MUL
        ADDD     ,U
        STD      ,U
*
        LDA     1,U
        LDB     #15
        MUL
        ADDD    2,U
        BCS     LOP20
        CMPD    #65521
        BLO     LOP21
LOP20   ADDD    #15
LOP21   STD     2,U
*
```

```
         LDA    ,U          |         STD    SAVE
         LDX    #TEMP       |         LBRA   NEXT
         CLR    ,X          |*
         CLR    1,X         |MLTFR    FDB    36917
         CLR    2,X         |MLTRR    FDB    28122
         LDB    #15         |*
         MUL                |         ORG    $0000
         STD    ,X          |MCND     FDB    0
         LDA    ,X          |PROD1    FCB    0
         LDB    #15         |PROD2    FCB    0
         MUL                |PROD3    FCB    0
         ADDD   1,X         |PROD4    FCB    0
         ADDD   2,U         |TEMP     FCB    0
         BCS    LOP22       |TEMP1    FCB    0
         CMPD   #65521      |TEMP3    FCB    0
         BLO    LOP23       |SAVE     FDB    0
LOP22    ADDD   #15         |FLAG     FCB    0
LOP23    STD    T17         |RES      FDB    0
         SYNC               |*
         SYNC               |         ORG    $FFFE
         SYNC               |STRT     EQU    $F800
         SYNC               |         END    BEGIN
         LDD    R17         |*
*     ***************************************************************
*     *                    PROCESSOR NUMBER 11                      *
*     ***************************************************************
         NAM    680911      |         BRA    OVER
OUTPUT   EQU    $0400       |NEXT     LDY    #MCND
STATUS   EQU    $0402       |         LDX    #MLTRR
T6       EQU    $0403       |         SYNC
T12      EQU    $0405       |         LDD    SAVE
INPUT    EQU    $0410       |*
R6       EQU    $0412       |OVER     STD    T6
R12      EQU    $0414       |         SYNC
SEM      EQU    $0416       |         STD    SAVE
*                           |         LDD    R6
         ORG    $F800       |         SUBD   SAVE
         NOP                |         BCC    SKP12
         ORCC   #%01010000  |         ADDD   #65521
         LDU    #PROD1      |SKP12    SYNC
BEGIN    CLRA               |         SYNC
         STA    FLAG        |         SYNC
         LDA    SEM         |         SYNC
         BEQ    FRD         |         ADDD   P12
START    LDA    #1          |         BCS    SKP14
         STA    FLAG        |         CMPD   #65521
FRD      LDY    #MCND       |         BLO    SKP15
         LDX    #MLTFR      |SKP14    ADDD   #15
         LDA    #1          |*
         STA    STATUS      |SKP15    STD    MCND
         SYNC               |         CLR    ,U
         CLRA               |         CLR    1,U
         STA    STATUS      |         LDA    1,X
         LDD    INPUT       |         LDB    1,Y
```

```
         MUL
         STD    2,U
         LDA     ,X
         LDB    1,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    SKP16
         INC     ,U
SKP16    LDA    1,X
         LDB     ,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    SKP19
         INC     ,U
SKP19    LDA     ,X
         LDB     ,Y
         MUL
         ADDD    ,U
         STD     ,U
*
         LDA    1,U
         LDB    #15
         MUL
         ADDD   2,U
         BCS    SKP20
         CMPD   #65521
         BLO    SKP21
SKP20    ADDD   #15
SKP21    STD    2,U
*
         LDA     ,U
         LDX    #TEMP
         CLR     ,X
         CLR    1,X
         CLR    2,X
         LDB    #15
         MUL
         STD     ,X
         LDA     ,X
         LDB    #15
         MUL
         ADDD   1,X
         ADDD   2,U
         BCS    SKP22
         CMPD   #65521
         BLO    SKP23
SKP22    ADDD   #15
SKP23    SYNC
*
         STD    T12
         SYNC
         SYNC
         SYNC
         SYNC
         STD    T6
         SYNC
         STD    SAVE
         LDD    R6
         SUBD   SAVE
         BCC    SKP24
         ADDD   #65521
*
SKP24    STD     SAVE
         LDA     FLAG
         CMPA    #1
         BEQ     MULT
         CMPA    #2
         BEQ     CONV
         LDD     SAVE
         STD     RES
         LBRA    BEGIN
CONV     LDD     SAVE
         STD     OUTPUT
         LBRA    BEGIN
*
MULT     INC     FLAG
         LDX     #SAVE
         LDY     #RES
LOP15    CLR     ,U
         CLR    1,U
         LDA    1,X
         LDB    1,Y
         MUL
         STD    2,U
         LDA     ,X
         LDB    1,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    LOP16
         INC     ,U
LOP16    LDA    1,X
         LDB     ,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    LOP19
         INC     ,U
LOP19    LDA     ,X
         LDB     ,Y
         MUL
         ADDD    ,U
         STD     ,U
*
         LDA    1,U
         LDB    #15
```

```
          MUL                    |          LDD    R6
          ADDD  2,U              |          STD    SAVE
          BCS   LOP20            |          SYNC
          CMPD  #65521           |          SYNC
          BLO   LOP21            |          SYNC
LOP20     ADDD  #15              |          LBRA   NEXT
LOP21     STD   2,U              |*
*                                |MLTFR     FDB   16087
          LDA    ,U              |MLTRR     FDB   29504
          LDX   #TEMP            |*
          CLR    ,X              |          ORG   $0000
          CLR   1,X              |MCND      FDB   0
          CLR   2,X              |PROD1     FCB   0
          LDB   #15              |PROD2     FCB   0
          MUL                    |PROD3     FCB   0
          STD    ,X              |PROD4     FCB   0
          LDA    ,X              |TEMP      FCB   0
          LDB   #15              |TEMP1     FCB   0
          MUL                    |TEMP3     FCB   0
          ADDD  1,X              |SAVE      FDB   0
          ADDD  2,U              |FLAG      FCB   0
          BCS   LOP22            |RES       FDB   0
          CMPD  #65521           |*
          BLO   LOP23            |          ORG   $FFFE
LOP22     ADDD  #15              |STRT      EQU   $F800
LOP23     STD   T6               |          END   BEGIN
          SYNC                   |*
*     ***************************************************************************
*     *                      PROCESSOR NUMBER 12                                  *
*     ***************************************************************************
          NAM   680912           |FRD       LDY    #MCND
OUTPUT    EQU   $0400            |          LDX    #MLTFR
STATUS    EQU   $0402            |          LDA    #1
T7        EQU   $0403            |          STA    STATUS
T15       EQU   $0405            |          SYNC
T13       EQU   $0407            |          CLRA
T11       EQU   $0409            |          STA    STATUS
INPUT     EQU   $0410            |          LDD    INPUT
R7        EQU   $0412            |          BRA    OVER
R15       EQU   $0414            |NEXT      LDY    #MCND
R13       EQU   $0416            |          LDX    #MLTRR
R11       EQU   $0418            |          SYNC
SEM       EQU   $041A            |          LDD    SAVE
*                                |*
          ORG    $F800           |OVER      STD   T7
          NOP                    |          SYNC
          ORCC  #%01010000       |          STD   SAVE
          LDU   #PROD1           |          LDD   R7
BEGIN     CLRA                   |          SUBD  SAVE
          STA   FLAG             |          BCC   SKP12
          LDA   SEM              |          ADDD  #65521
          BEQ   FRD              |SKP12     SYNC
START     LDA   #1               |          STD   T15
          STA   FLAG             |          SYNC
```

```
FRD     LDY   #MCND      |        LDB   ,Y
        LDX   #MLTRR     |        MUL
        LDA   #1         |        ADDD  1,U
        STA   STATUS     |        STD   1,U
        SYNC             |        BCC   SKP21
        CLRA             |        INC   ,U
        STA   STATUS     |SKP21   LDA   ,X
        LDD   INPUT      |        LDB   ,Y
        BRA   OVER       |        MUL
NEXT    LDY   #MCND      |        ADDD  ,U
        LDX   #MLTRR     |        STD   ,U
        SYNC             |*
        LDD   SAVE       |        LDA   1,U
*                        |        LDB   #15
OVER    STD   T7         |        MUL
        SYNC             |        ADDD  2,U
        STD   SAVE       |        BCS   SKP22
        LDD   R7         |        CMPD  #65521
        SUBD  SAVE       |        BLO   SKP23
        BCC   SKP12      |SKP22   ADDD  #15
        ADDD  #65521     |SKP23   STD   2,U
SKP12   SYNC             |*
        STD   T15        |        LDA   ,U
        SYNC             |        LDX   #TEMP
        ADDD  R15        |        CLR   ,X
        BCS   SKP14      |        CLR   1,X
        CMPD  #65521     |        CLR   2,X
        BLO   SKP15      |        LDB   #15
SKP14   ADDD  #15        |        MUL
SKP15   STD   T13        |        STD   ,X
        SYNC             |        LDA   ,X
        ADDD  R13        |        LDB   #15
        BCS   SKP16      |        MUL
        CMPD  #65521     |        ADDD  1,X
        BLO   SKP17      |        ADDD  2,U
SKP16   ADDD  #15        |        BCS   SKP24
SKP17   STD   T11        |        CMPD  #65521
        SYNC             |        BLO   SKP25
*                        |SKP24   ADDD  #15
        STD   MCND       |SKP25   SYNC
        CLR   ,U         |*
        CLR   1,U        |        SYNC
        LDA   1,X        |        ADDD  R11
        LDB   1,Y        |        BCS   SKP26
        MUL              |        CMPD  #65521
        STD   2,U        |        BLO   SKP27
        LDA   ,X         |SKP26   ADDD  #15
        LDB   1,Y        |SKP27   STD   T13
        MUL              |        SYNC
        ADDD  1,U        |        ADDD  R13
        STD   1,U        |        BCS   SKP28
        BCC   SKP18      |        CMPD  #65521
        INC   ,U         |        BLO   SKP29
SKP18   LDA   1,X        |SKP28   ADDD  #15
```

```
SKP29   STD   T15
        SYNC
        ADDD  R15
        BCS   SKP30
        CMPD  #65521
        BLO   SKP31
SKP30   ADDD  #15
SKP31   SYNC
        STD   T7
        SYNC
        STD   SAVE
        LDD   R7
        SUBD  SAVE
        BCC   SKP32
        ADDD  #65521
*
SKP32   STD    SAVE
        LDA    FLAG
        CMPA   #1
        BEQ    MULT
        CMPA   #2
        BEQ    CONV
        LDD    SAVE
        STD    RES
        LBRA   BEGIN
CONV    LDD    SAVE
        STD    OUTPUT
        LBRA   BEGIN
*
MULT    INC    FLAG
        LDX    #SAVE
        LDY    #RES
LOP15   CLR    ,U
        CLR    1,U
        LDA    1,X
        LDB    1,Y
        MUL
        STD    2,U
        LDA    ,X
        LDB    1,Y
        MUL
        ADDD   1,U
        STD    1,U
        BCC    LOP16
        INC    ,U
LOP16   LDA    1,X
        LDB    ,Y
        MUL
        ADDD   1,U
        STD    1,U
        BCC    LOP19
        INC    ,U
LOP19   LDA    ,X
        LDB    ,Y
        MUL
        ADDD   ,U
        STD    ,U
*
        LDA    1,U
        LDB    #15
        MUL
        ADDD   2,U
        BCS    LOP20
        CMPD   #65521
        BLO    LOP21
LOP20   ADDD   #15
LOP21   STD    2,U
*
        LDA    ,U
        LDX    #TEMP
        CLR    ,X
        CLR    1,X
        CLR    2,X
        LDB    #15
        MUL
        STD    ,X
        LDA    ,X
        LDB    #15
        MUL
        ADDD   1,X
        ADDD   2,U
        BCS    LOP22
        CMPD   #65521
        BLO    LOP23
LOP22   ADDD   #15
LOP23   STD    T7
        SYNC
        LDD    R7
        STD    T13
        SYNC
        LDD    R13
        STD    T7
        SYNC
        LDD    R7
        STD    SAVE
        SYNC
        LBRA   NEXT
*
MLTFR   FDB    29032
MLTRR   FDB    28641
*
        ORG    $0000
MCND    FDB    0
PROD1   FCB    0
PROD2   FCB    0
PROD3   FCB    0
PROD4   FCB    0
TEMP    FCB    0
```

```
TEMP1    FCB   0
TEMP3    FCB   0
SAVE     FDB   0
FLAG     FCB   0
RES      FDB   0
*    ***************************************************************
*    *                    PROCESSOR NUMBER 13                      *
*    ***************************************************************
         NAM   680913
OUTPUT   EQU   $0400
STATUS   EQU   $0402
T8       EQU   $0403
T14      EQU   $0405
T12      EQU   $0407
INPUT    EQU   $0410
R8       EQU   $0412
R14      EQU   $0414
R12      EQU   $0416
SEM      EQU   $0418
*
         ORG    $F800
         NOP
         ORCC   #%01010000
         LDU    #PROD1
BEGIN    CLRA
         STA    FLAG
         LDA    SEM
         BEQ    FRD
START    LDA    #1
         STA    FLAG
FRD      LDY    #MCND
         LDX    #MLTFR
         LDA    #1
         STA    STATUS
         SYNC
         CLRA
         STA    STATUS
         LDD    INPUT
         BRA    OVER
NEXT     LDY    #MCND
         LDX    #MLTRR
         SYNC
         LDD    SAVE
*
OVER     STD    T8
         SYNC
         STD    SAVE
         LDD    R8
         SUBD   SAVE
         BCC    SKP12
         ADDD   #65521
SKP12    SYNC
         STD    T14
         SYNC
*
         ORG    $FFFE
STRT     EQU    $F800
         END    BEGIN
*
         ADDD   R14
         BCS    SKP14
         CMPD   #65521
         BLO    SKP15
SKP14    ADDD   #15
SKP15    STD    T12
         SYNC
         STD    SAVE
         LDD    R12
         SUBD   SAVE
         BCC    SKP16
         ADDD   #65521
SKP16    SYNC
*
         STD    MCND
         CLR     ,U
         CLR    1,U
         LDA    1,X
         LDB    1,Y
         MUL
         STD    2,U
         LDA     ,X
         LDB    1,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    SKP18
         INC     ,U
SKP18    LDA    1,X
         LDB     ,Y
         MUL
         ADDD   1,U
         STD    1,U
         BCC    SKP21
         INC     ,U
SKP21    LDA     ,X
         LDB     ,Y
         MUL
         ADDD    ,U
         STD     ,U
*
         LDA    1,U
         LDB    #15
         MUL
         ADDD   2,U
         BCS    SKP22
```

```
         CMPD   #65521
         BLO    SKP23
SKP22    ADDD   #15
SKP23    STD    2,U
*
         LDA     ,U
         LDX    #TEMP
         CLR     ,X
         CLR    1,X
         CLR    2,X
         LDB    #15
         MUL
         STD     ,X
         LDA     ,X
         LDB    #15
         MUL
         ADDD   1,X
         ADDD   2,U
         BCS    SKP24
         CMPD   #65521
         BLO    SKP25
SKP24    ADDD   #15
SKP25    SYNC
*
         SYNC
         STD    T12
         SYNC
         STD    SAVE
         LDD    R12
         SUBD   SAVE
         BCC    SKP26
         ADDD   #65521
SKP26    STD    T14
         SYNC
         ADDD   R14
         BCS    SKP28
         CMPD   #65521
         BLO    SKP29
SKP28    ADDD   #15
SKP29    SYNC
         STD    T8
         SYNC
         STD    SAVE
         LDD    R8
         SUBD   SAVE
         BCC    SKP30
         ADDD   #65521
*
SKP30    STD     SAVE
         LDA     FLAG
         CMPA    #1
         BEQ     MULT
         CMPA    #2
         BEQ     CONV
         LDD     SAVE
         STD     RES
         LBRA    BEGIN
CONV     LDD     SAVE
         STD     OUTPUT
         LBRA    BEGIN
*
MULT     INC     FLAG
         LDX     #SAVE
         LDY     #RES
LOP15    CLR      ,U
         CLR     1,U
         LDA     1,X
         LDB     1,Y
         MUL
         STD     2,U
         LDA      ,X
         LDB     1,Y
         MUL
         ADDD    1,U
         STD     1,U
         BCC     LOP16
         INC      ,U
LOP16    LDA     1,X
         LDB      ,Y
         MUL
         ADDD    1,U
         STD     1,U
         BCC     LOP19
         INC      ,U
LOP19    LDA      ,X
         LDB      ,Y
         MUL
         ADDD     ,U
         STD      ,U
*
         LDA     1,U
         LDB     #15
         MUL
         ADDD    2,U
         BCS     LOP20
         CMPD    #65521
         BLO     LOP21
LOP20    ADDD    #15
LOP21    STD     2,U
*
         LDA      ,U
         LDX     #TEMP
         CLR      ,X
         CLR     1,X
         CLR     2,X
         LDB     #15
         MUL
         STD      ,X
```

```
          LDA    ,X              |*
          LDB    #15             |MLTFR    FDB     8748
          MUL                    |MLTRR    FDB    12521
          ADDD   1,X             |*
          ADDD   2,U             |         ORG    $0000
          BCS    LOP22           |MCND     FDB    0
          CMPD   #65521          |PROD1    FCB    0
          BLO    LOP23           |PROD2    FCB    0
LOP22     ADDD   #15             |PROD3    FCB    0
LOP23     STD    T8              |PROD4    FCB    0
          SYNC                   |TEMP     FCB    0
          LDD    R8              |TEMP1    FCB    0
          STD    T14             |TEMP3    FCB    0
          LDD    R14             |SAVE     FDB    0
          STD    T12             |FLAG     FCB    0
          SYNC                   |RES      FDB    0
          LDD    R12             |*
          STD    SAVE            |         ORG    $FFFE
          SYNC                   |STRT     EQU    $F800
          SYNC                   |         END    BEGIN
          LBRA    NEXT           |*
*     ***************************************************************************
*     *                        PROCESSOR NUMBER 14                                *
*     ***************************************************************************
          NAM    680914          |         BRA     OVER
OUTPUT    EQU    $0400           |NEXT     LDY     #MCND
STATUS    EQU    $0402           |         LDX     #MLTRR
T9        EQU    $0403           |         SYNC
T13       EQU    $0405           |         LDD     SAVE
T18       EQU    $0407           |*
INPUT     EQU    $0410           |OVER     STD    T9
R9        EQU    $0412           |         SYNC
R13       EQU    $0414           |         STD    SAVE
R18       EQU    $0416           |         LDD    P9
SEM       EQU    $0418           |         SUBD   SAVE
*                                |         BCC    SKP12
          ORG     $F800          |         ADDD   #65521
          NOP                    |SKP12    SYNC
          ORCC   #%01010000      |         STD    T13
          LDU    #PROD1          |         SYNC
BEGIN     CLRA                   |         SUBD   R13
          STA    FLAG            |         BCC    SKP14
          LDA    SEM             |         ADDD   #65521
          BEQ    FRD             |SKP14    STD    T18
START     LDA    #1              |         SYNC
          STA    FLAG            |         SYNC
FRD       LDY    #MCND           |*
          LDX    #MLTFR          |         STD    MCND
          LDA    #1              |         CLR     ,U
          STA    STATUS          |         CLR    1,U
          SYNC                   |         LDA    1,X
          CLRA                   |         LDB    1,Y
          STA    STATUS          |         MUL
          LDD    INPUT           |         STD    2,U
```

```
        LDA     ,X
        LDB     1,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     SKP16
        INC     ,U
SKP16   LDA     1,X
        LDB     ,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     SKP19
        INC     ,U
SKP19   LDA     ,X
        LDB     ,Y
        MUL
        ADDD    ,U
        STD     ,U
*
        LDA     1,U
        LDB     #15
        MUL
        ADDD    2,U
        BCS     SKP20
        CMPD    #65521
        BLO     SKP21
SKP20   ADDD    #15
SKP21   STD     2,U
*
        LDA     ,U
        LDX     #TEMP
        CLR     ,X
        CLR     1,X
        CLR     2,X
        LDB     #15
        MUL
        STD     ,X
        LDA     ,X
        LDB     #15
        MUL
        ADDD    1,X
        ADDD    2,U
        BCS     SKP22
        CMPD    #65521
        BLO     SKP23
SKP22   ADDD    #15
SKP23   SYNC
*
        SYNC
        ADDD    R18
        BCS     SKP24
        CMPD    #65521
        BLO     SKP25
SKP24   ADDD    #15
SKP25   SYNC
        STD     T13
        SYNC
        STD     SAVE
        LDD     R13
        SUBD    SAVE
        BCC     SKP26
        ADDD    #65521
SKP26   SYNC
        STD     T9
        SYNC
        STD     SAVE
        LDD     R9
        SUBD    SAVE
        BCC     SKP28
        ADDD    #65521
*
SKP28   STD     SAVE
        LDA     FLAG
        CMPA    #1
        BEQ     MULT
        CMPA    #2
        BEQ     CONV
        LDD     SAVE
        STD     RES
        LBRA    BEGIN
CONV    LDD     SAVE
        STD     OUTPUT
        LBRA    BEGIN
*
MULT    INC     FLAG
        LDX     #SAVE
        LDY     #RES
LOP15   CLR     ,U
        CLR     1,U
        LDA     1,X
        LDB     1,Y
        MUL
        STD     2,U
        LDA     ,X
        LDB     1,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP16
        INC     ,U
LOP16   LDA     1,X
        LDB     ,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC     LOP19
        INC     ,U
```

```
LOP19    LDA     ,X          |LOP23    STD     T13
         LDB     ,Y          |        SYNC
         MUL                 |        SYNC
         ADDD    ,U          |        LDD     R13
         STD     ,U          |        STD     T18
*                            |        LDD     R18
         LDA     1,U         |        STD     T9
         LDB     #15         |        SYNC
         MUL                 |        LDD     R9
         ADDD    2,U         |        STD     SAVE
         BCS     LOP20       |        SYNC
         CMPD    #65521      |        LBRA    NEXT
         BLO     LOP21       |*
LOP20    ADDD    #15         |MLTFR    FDB    1465
LOP21    STD     2,U         |MLTRR    FDB    21938
*                            |*
         LDA     ,U          |        ORG     $0000
         LDX     #TEMP       |MCND     FDB    0
         CLR     ,X          |PROD1    FCB    0
         CLR     1,X         |PROD2    FCB    0
         CLR     2,X         |PROD3    FCB    0
         LDB     #15         |PROD4    FCB    0
         MUL                 |TEMP     FCB    0
         STD     ,X          |TEMP1    FCB    0
         LDA     ,X          |TEMP3    FCB    0
         LDB     #15         |SAVE     FDB    0
         MUL                 |FLAG     FCB    0
         ADDD    1,X         |RES      FDB    0
         ADDD    2,U         |*
         BCS     LOP22       |        ORG     $FFFE
         CMPD    #65521      |STRT     EQU    $F800
         BLO     LOP23       |        END     BEGIN
LOP22    ADDD    #15         |*
*     ***************************************************************************
*     *                      PROCESSOR NUMBER 15                                  *
*     ***************************************************************************
         NAM     680915      |        STA     FLAG
OUTPUT   EQU     $0400       |        LDA     SEM
STATUS   EQU     $0402       |        BEQ     FRD
T10      EQU     $0403       |START    LDA     #1
T12      EQU     $0405       |        STA     FLAG
T18      EQU     $0407       |FRD      LDY     #MCND
INPUT    EQU     $0410       |        LDX     #MLTFR
R10      EQU     $0412       |        LDA     #1
R12      EQU     $0414       |        STA     STATUS
R18      EQU     $0416       |        SYNC
SEM      EQU     $0418       |        CLRA
         ORG     $F800       |        STA     STATUS
*                            |        LDD     INPUT
         ORG     $F800       |        BRA     OVER
         NOP                 |NEXT     LDY     #MCND
         ORCC    #%01010000  |        LDX     #MLTRR
         LDU     #PROD1      |        SYNC
BEGIN    CLRA                 |        LDD     SAVE
```

```
*
OVER    STD   T10
        SYNC
        STD   SAVE
        LDD   R10
        SUBD  SAVE
        BCC   SKP12
        ADDD  #65521
SKP12   SYNC
        STD   T12
        SYNC
        STD   SAVE
        LDD   R12
        SUBD  SAVE
        BCC   SKP14
        ADDD  #65521
SKP14   STD   T18
        SYNC
        SYNC
*
        STD   MCND
        CLR    ,U
        CLR   1,U
        LDA   1,X
        LDB   1,Y
        MUL
        STD   2,U
        LDA    ,X
        LDB   1,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKP16
        INC    ,U
SKP16   LDA   1,X
        LDB    ,Y
        MUL
        ADDD  1,U
        STD   1,U
        BCC   SKP19
        INC    ,U
SKP19   LDA    ,X
        LDB    ,Y
        MUL
        ADDD   ,U
        STD    ,U
*
        LDA   1,U
        LDB   #15
        MUL
        ADDD  2,U
        BCS   SKP20
        CMPD  #65521
        BLO   SKP21
SKP20   ADDD  #15
SKP21   STD   2,U
*
        LDA    ,U
        LDX   #TEMP
        CLR    ,X
        CLR   1,X
        CLR   2,X
        LDB   #15
        MUL
        STD    ,X
        LDA    ,X
        LDB   #15
        MUL
        ADDD  1,X
        ADDD  2,U
        BCS   SKP22
        CMPD  #65521
        BLO   SKP23
SKP22   ADDD  #15
SKP23   SYNC
*
        SYNC
        SUBD  R19
        BCC   SKP24
        ADDD  #65521
SKP24   SYNC
        STD   T12
        SYNC
        STD   SAVE
        LDD   R12
        SUBD  SAVE
        BCC   SKP26
        ADDD  #65521
SKP26   SYNC
        STD   T10
        SYNC
        STD   SAVE
        LDD   R10
        SUBD  SAVE
        BCC   SKP28
        ADDD  #65521
*
SKP28   STD    SAVE
        LDA    FLAG
        CMPA   #1
        BEQ    MULT
        CMPA   #2
        BEQ    CONV
        LDD    SAVE
        STD    RES
        LBRA   BEGIN
CONV    LDD    SAVE
        STD    OUTPUT
```

```
         LBRA    BEGIN
*
MULT     INC     FLAG
         LDX     #SAVE
         LDY     #RES
LOP15    CLR     ,U
         CLR     1,U
         LDA     1,X
         LDB     1,Y
         MUL
         STD     2,U
         LDA     ,X
         LDB     1,Y
         MUL
         ADDD    1,U
         STD     1,U
         BCC     LOP16
         INC     ,U
LOP16    LDA     1,X
         LDB     ,Y
         MUL
         ADDD    1,U
         STD     1,U
         BCC     LOP19
         INC     ,U
LOP19    LDA     ,X
         LDB     ,Y
         MUL
         ADDD    ,U
         STD     ,U
*
         LDA     1,U
         LDB     #15
         MUL
         ADDD    2,U
         BCS     LOP20
         CMPD    #65521
         BLO     LOP21
LOP20    ADDD    #15
LOP21    STD     2,U
*
         LDA     ,U
         LDX     #TEMP
         CLR     ,X
         CLR     1,X
         CLR     2,X
         LDB     #15
         MUL
         STD     ,X
         LDA     ,X
         LDB     #15
         MUL
         ADDD    1,X
         ADDD    2,U
         BCS     LOP22
         CMPD    #65521
         BLO     LOP23
LOP22    ADDD    #15
LOP23    STD     T18
         SYNC
         SYNC
         SYNC
         SYNC
         LDD     R18
         STD     SAVE
         LBRA    NEXT
*
MLTFR    FDB     23174
MLTRR    FDB     5913
*
         ORG     $0000
MCND     FDB     0
PROD1    FCB     0
PROD2    FCB     0
PROD3    FCB     0
PROD4    FCB     0
TEMP     FCB     0
TEMP1    FCB     0
TEMP3    FCB     0
SAVE     FDB     0
FLAG     FCB     0
RES      FDB     0
*
         ORG     $FFFE
STRT     EQU     $F800
         END     BEGIN
*********************************************************************
*                    PROCESSOR NUMBER 16                            *
*********************************************************************
         NAM     680916
T4       EQU     $0410
T5       EQU     $0412
R4       EQU     $0414
R5       EQU     $0416
SEM      EQU     $0418
*
         ORG     $F800
         NOP
         ORCC    #%01010000
         LDU     #PROD1
BEGIN    CLRA
         STA     FLAG
         LDA     SEM
         BEQ     FRD
START    LDA     #1
```

```
         STA   FLAG
FRD      LDY   #MCND
         LDX   #MLTFR
         BRA   OVER
NEXT     LDY   #MCND
         LDX   #MLTRR
OVER     SYNC
         SYNC
         SYNC
         SYNC
         SYNC
         LDD   R4
         ADDD  R5
         BCS   SKP12
         CMPD  #65521
         BLO   SKP13
SKP12    ADDD  #15
SKP13    SYNC
*
         STD   MCND
         CLR    ,U
         CLR   1,U
         LDA   1,X
         LDB   1,Y
         MUL
         STD   2,U
         LDA    ,X
         LDB   1,Y
         MUL
         ADDD  1,U
         STD   1,U
         BCC   SKP14
         INC    ,U
SKP14    LDA   1,X
         LDB    ,Y
         MUL
         ADDD  1,U
         STD   1,U
         BCC   SKP17
         INC    ,U
SKP17    LDA    ,X
         LDB    ,Y
         MUL
         ADDD   ,U
         STD    ,U
*
         LDA   1,U
         LDB   #15
         MUL
         ADDD  2,U
         BCS   SKP18
         CMPD  #65521
         BLO   SKP19
SKP18    ADDD  #15
SKP19    STD   2,U
*
         LDA    ,U
         LDX   #TEMP
         CLR    ,X
         CLR   1,X
         CLR   2,X
         LDB   #15
         MUL
         STD    ,X
         LDA    ,X
         LDB   #15
         MUL
         ADDD  1,X
         ADDD  2,U
         BCS   SKP20
         CMPD  #65521
         BLO   SKP21
SKP20    ADDD  #15
SKP21    SYNC
*
         STD   T5
         STD   T4
         SYNC
         SYNC
         SYNC
         SYNC
         SYNC
         LDA    FLAG
         CMPA   #1
         BEQ    SKP
         LBRA   BEGIN
SKP      INC    FLAG
         SYNC
         LDD    R5
         STD    T4
         SYNC
         SYNC
         SYNC
         LBRA   NEXT
*
MLTFR    FDB    18005
MLTRR    FDB     5493
*
         ORG   $0000
MCND     FDB   0
PROD1    FCB   0
PROD2    FCB   0
PROD3    FCB   0
PROD4    FCB   0
TEMP     FCB   0
TEMP1    FCB   0
TEMP3    FCB   0
SAVE     FDB   0
```

```
FLAG    FCB   0                 |        ORG   $FFFE
*                               |STRT    EQU   $F800
*                               |        END   BEGIN
*     ***************************************************************
*     *                    PROCESSOR NUMBER 17                      *
*     ***************************************************************
        NAM   680917            |        INC    ,U
T9      EQU   $0410             |SKP14   LDA   1,X
T10     EQU   $0412             |        LDB    ,Y
R9      EQU   $0414             |        MUL
R10     EQU   $0416             |        ADDD  1,U
SEM     EQU   $0418             |        STD   1,U
*                               |        BCC   SKP17
        ORG   $F800             |        INC    ,U
        NOP                     |SKP17   LDA    ,X
        ORCC  #%01010000        |        LDB    ,Y
        LDU   #PROD1            |        MUL
BEGIN   CLRA                    |        ADDD   ,U
        STA   FLAG              |        STD    ,U
        LDA   SEM               |*
        BEQ   FRD               |        LDA   1,U
STAPT   LDA   #1                |        LDB   #15
        STA   FLAG              |        MUL
FRD     LDY   #MCND             |        ADDD  2,U
        LDX   #MLTFR            |        BCS   SKP18
        BRA   OVER              |        CMPD  #65521
NEXT    LDY   #MCND             |        BLO   SKP19
        LDX   #MLTRR            |SKP18   ADDD  #15
OVER    SYNC                    |SKP19   STD   2,U
        SYNC                    |*
        SYNC                    |        LDA    ,U
        SYNC                    |        LDX   #TEMP
        SYNC                    |        CLR    ,X
        LDD   R9                |        CLR   1,X
        ADDD  R10               |        CLR   2,X
        BCS   SKP12             |        LDB   #15
        CMPD  #65521            |        MUL
        BLO   SKP13             |        STD    ,X
SKP12   ADDD  #15               |        LDA    ,X
SKP13   SYNC                    |        LDB   #15
*                               |        MUL
        STD   MCND              |        ADDD  1,X
        CLR    ,U               |        ADDD  2,U
        CLR   1,U               |        BCS   SKP20
        LDA   1,X               |        CMPD  #65521
        LDB   1,Y               |        BLO   SKP21
        MUL                     |SKP20   ADDD  #15
        STD   2,U               |SKP21   SYNC
        LDA    ,X               |*
        LDB   1,Y               |        STD   T10
        MUL                     |        STD   T9
        ADDD  1,U               |        SYNC
        STD   1,U               |        SYNC
        BCC   SKP14             |        SYNC
```

```
            SYNC                    |MLTRR     FDB    34561
            SYNC                    |*
            LDA     FLAG            |          ORG    $0000
            CMPA    #1              |MCND      FDB    0
            BEQ     SKP             |PROD1     FCB    0
            LBRA    BEGIN           |PROD2     FCB    0
SKP         INC     FLAG            |PROD3     FCB    0
            SYNC                    |PROD4     FCB    0
            LDD     R10             |TEMP      FCB    0
            STD     T9              |TEMP1     FCB    0
            SYNC                    |TEMP3     FCB    0
            SYNC                    |SAVE      FDB    0
            LDD     R9              |FLAG      FCB    0
            STD     T10             |*
            SYNC                    |          ORG    $FFFE
            LBRA    NEXT            |STRT      EQU    $F800
*                                   |          END    BEGIN
MLTFR       FDB     5753            |*
*       ***********************************************************************
*       *                      PROCESSOR NUMBER 18                            *
*       ***********************************************************************
            NAM     680918          |SKP13     SYNC
T14         EQU     $0410           |*
T15         EQU     $0412           |          STD    MCND
R14         EQU     $0414           |          CLR     ,U
R15         EQU     $0416           |          CLR    1,U
SEM         EQU     $0418           |          LDA    1,X
*                                   |          LDB    1,Y
            ORG     $F800           |          MUL
            NOP                     |          STD    2,U
            ORCC    #%01010000      |          LDA     ,X
            LDU     #PROD1          |          LDB    1,Y
BEGIN       CLRA                    |          MUL
            STA     FLAG            |          ADDD   1,U
            LDA     SEM             |          STD    1,U
            BEQ     FRD             |          BCC    SKP14
START       LDA     #1              |          INC     ,U
            STA     FLAG            |SKP14     LDA    1,X
FRD         LDY     #MCND           |          LDB     ,Y
            LDX     #MLTFR          |          MUL
            BRA     OVER            |          ADDD   1,U
NEXT        LDY     #MCND           |          STD    1,U
            LDX     #MLTRR          |          BCC    SKP17
OVER        SYNC                    |          INC     ,U
            SYNC                    |SKP17     LDA     ,X
            SYNC                    |          LDB     ,Y
            SYNC                    |          MUL
            SYNC                    |          ADDD    ,U
            LDD     R14             |          STD     ,U
            ADDD    R15             |*
            BCS     SKP12           |          LDA    1,U
            CMPD    #65521          |          LDB    #15
            BLO     SKP13           |          MUL
SKP12       ADDD    #15             |          ADDD   2,U
```

```
          BCS    SKP18          |          CMPA   #1
          CMPD   #65521         |          BEQ    SKP
          BLO    SKP19          |          LBRA   BEGIN
SKP18     ADDD   #15            |SKP       INC    FLAG
SKP19     STD    2,U            |          SYNC
*                                |          LDD    R15
          LDA     ,U            |          STD    T14
          LDX    #TEMP          |          SYNC
          CLR     ,X            |          SYNC
          CLR    1,X            |          LDD    R14
          CLR    2,X            |          STD    T15
          LDB    #15            |          SYNC
          MUL                   |          LBRA   NEXT
          STD     ,X            |*
          LDA     ,X            |MLTFR     FDB    43615
          LDB    #15            |MLTRR     FDB    24748
          MUL                   |*
          ADDD   1,X            |          ORG    $0000
          ADDD   2,U            |MCND      FDB    0
          BCS    SKP20          |PROD1     FCB    0
          CMPD   #65521         |PROD2     FCB    0
          BLO    SKP21          |PROD3     FCB    0
SKP20     ADDD   #15            |PROD4     FCB    0
SKP21     SYNC                  |TEMP      FCB    0
*                                |TEMP1     FCB    0
          STD    T15            |TEMP3     FCB    0
          STD    T14            |SAVE      FDB    0
          SYNC                  |FLAG      FCB    0
          SYNC                  |*
          SYNC                  |          ORG    $FFFE
          SYNC                  |STRT      EQU    $F800
          SYNC                  |          END    BEGIN
          LDA     FLAG          |*
*
*       ************************************************************************
*       *                        CONTROL MICROPROCESSOR                        *
*       ************************************************************************
*
*         NAM     CONTROL       |          ORG    $F800
*                               |START     NOP
ACIACR    EQU     $1040         |          ORCC   #%01010000
ACIASR    EQU     $1040         |          LDS    #$80
ACIARX    EQU     $1041         |          CLRA
ACIATX    EQU     $1041         |          STA    CONTRL
ARYOUT    EQU     $1020         |          STA    STATUS
INPUT     EQU     $1000         |          LDA    #%00000101
OUT1M     EQU     $1050         |          STA    CONTRL
OUT1L     EQU     $1051         |          LDA    #$13
OUT2M     EQU     $1052         |          STA    ACIACR
OUT2L     EQU     $1053         |          LDA    #$11
DATA      EQU     $1054         |          STA    ACIACR
CONTRL    EQU     $101E         |          LDA    #%00000111
*                               |          STA    CONTRL
          LDA     #%00000110    |          STA    TMP
```

```
         STA     CONTRL          |         JSR     RCX
         LDA     #%00000111      |         JSR     TXR
         STA     CONTRL          |         CMPA    #$0D
         STA     STATUS          |         BEQ     SK5
         BRA     BEGIN           |         JSR     VALID
INIT     LDS     #$80            |         ADDA    TMP
         LDA     #$07            |         BRA     SK6
         JSR     TXR             |SK5      LDA     TMP
         JSR     CRLF            |         LSRA
BEGIN    JSR     CRLF            |         LSRA
         JSR     PFX             |         LSRA
         JSR     RCX             |         LSRA
         CMPA    #'1             |         BRA     SK6
         BEQ     SKP1            |ZERO      CLRA
         CMPA    #'2             |SK6       STA     CNT
         BEQ     SKP2            |         CMPA    #30
         LBRA    DSP             |         LBHS    ERMSG1
*                                |LOOP5     JSR     CRLF
SKP1     LDA     #1              |*
         STA     FLAG            |WRITE     LDA     CNT
         LDX     #MSG3           |         LSRA
         LDA     #15             |         LSRA
         STA     CNT             |         LSRA
         JSR     DSPLY           |         LSRA
         LDY     #IN1            |         JSR     CONVA
         LDX     #ARYIN          |         JSR     TXR
         JSR     EXG             |         LDA     CNT
         BRA     MODFY           |         ANDA    #$0F
SKP2     LDA     #2              |         JSR     CONVA
         STA     FLAG            |         JSR     TXR
         LDY     #IN2            |         LDA     #'=
         LDX     #ARYIN          |         JSR     TXR
         JSR     EXG             |         LDA     #$20
         LDX     #MSG8           |         JSR     TXR
         LDA     #13             |         LDB     CNT
         STA     CNT             |         LDA     B,X
         JSR     DSPLY           |         LSRA
         JSR     CRLF            |         LSRA
         JSR     PFX             |         LSRA
*                                |         LSRA
MODFY    JSR     CRLF            |         JSR     CONVA
         LDX     #ARYIN          |         JSR     TXR
         LDA     #$20            |         LDB     CNT
         JSR     TXR             |         LDA     B,X
         JSR     RCX             |         ANDA    #$0F
         JSR     TXR             |         JSR     CONVA
         CMPA    #$0D            |         JSR     TXR
         BEQ     ZERO            |         LDA     #$20
         JSR     VALID           |         JSR     TXR
         LSLA                    |*
         LSLA                    |READ      JSR     RCX
         LSLA                    |         JSR     TXR
         LSLA                    |         CMPA    #$0D
         BEQ     MOVE            |         LDA     STATUS
```

```
        CMPA    #'-
        BEQ     DECR
        CMPA    #$20
        BEQ     INCR
        JSR     VALID
        LSLA
        LSLA
        LSLA
        LSLA
        STA     TMP
        JSR     RCX
        JSR     TXR
        JSR     VALID
        ADDA    TMP
        LDB     CNT
        STA     B,X
INCR    LDA     CNT
        INCA
        CMPA    #30
        BHS     MOVE
        STA     CNT
        BRA     LOOP5
DECR    LDA     CNT
        DECA
        BLT     MOVE
        STA     CNT
        LBRA    LOOP5
*
MOVE    JSR     CRLF
        JSR     PFX
        JSR     CRLF
        LDY     #ARYIN
        LDA     FLAG
        CMPA    #1
        BNE     SKP3
        LDA     STATUS
        ANDA    #%10111111
        STA     CONTRL
        STA     STATUS
        LDX     #IN1
        JSR     EXG
        LDX     #INPUT
        JSR     EXG
        LBRA    BEGIN
SKP3    LDA     STATUS
        ORA     #%01000000
        STA     CONTRL
        STA     STATUS
        LDX     #IN2
        JSR     EXG
        LDX     #INPUT
        JSR     EXG
*
        RTS
        ANDA    #%11111110
        STA     CONTRL
        ORA     #%00001001
        STA     CONTRL
        SYNC
        ANDA    #%11110111
        STA     CONTRL
        STA     STATUS
*
CONV    LDX     #OUT
        JSR     EXCHG
        JSR     CRLF
        LDX     #MSG4
        LDA     #11
        STA     CNT
        JSP     DSPLY
        JSR     CRLF
        LDX     #OUT
        JSR     ARAY
        JSR     CRLF
SKP4    LDA     #15
        STA     CNT
        LDX     #OUT
SKP5    LDD     ,X++
        STD     OUT2M
        DEC     CNT
        BNE     SKP5
        LDA     ACIASR
        LSRA
        BCC     SKP4
        LDA     ACIARX
        ANDA    #$7F
        CMPA    #'G
        LBEQ    GET
        LBRA    BEGIN
*
DSPLY   LDA     ,X+
        JSR     TXR
        DEC     CNT
        BNE     DSPLY
        RTS
*
TXR     LDB     #$02
WAIT    BITB    ACIASR
        BEQ     WAIT
        STA     ACIATX
        RTS     RETURN
*
RCX     LDA     ACIASR
        LSRA
        BCC     RCX
        LDA     ACIARX
        ANDA    #$7F
        STA     STATUS
```

```
*
CONVA   CMPA   #9
        BLS    OMIT
        ADDA   #'A-'9-1
OMIT    ADDA   #'0
        ANDA   #$7F
        RTS
*
CRLF    LDA    #$0D
        JSR    TXR
        LDA    #$0A
        JSR    TXR
        RTS
*
EXCHG   LDA    STATUS
        ORA    #%00001000
        STA    CONTRL
        STA    STATUS
        LDY    #ARYOUT
        SYNC
*
EXG     LDD     ,Y
        STD     ,X
        LDD    2,Y
        STD    2,X
        LDD    4,Y
        STD    4,X
        LDD    6,Y
        STD    6,X
        LDD    8,Y
        STD    8,X
        LDD   10,Y
        STD   10,X
        LDD   12,Y
        STD   12,X
        LDD   14,Y
        STD   14,X
        LDD   16,Y
        STD   16,X
        LDD   18,Y
        STD   18,X
        LDD   20,Y
        STD   20,X
        LDD   22,Y
        STD   22,X
        LDD   24,Y
        STD   24,X
        LDD   26,Y
        STD   26,X
        LDD   28,Y
        STD   28,X
        ANDA   #%11110111
        STA    CONTRL
        ANDA   #$7F
        RTS
*
*
GET     JSR    CRLF
        JSR    PFX
        LDX    #MSG4
        LDA    #12
        STA    CNT
        JSR    DSPLY
        JSR    CRLF
        JSR    PFX
LOOPW   LDA    STATUS
        ORA    #%00010000
        STA    STATUS
        STA    CONTRL
LOOPX   LDA    #15
        STA    CNT
        LDY    #ARYIN
        LDX    #INPUT
        LDU    #IN2
        LDD    DATA
LOOPY   SYNC
        LDD    DATA
        STD    ,X++
        STD    ,Y++
        STD    ,U++
        DEC    CNT
        BNE    LOOPY
*
        LDA    STATUS
        ANDA   #%11111110
        STA    CONTRL
        ORA    #%00001001
        STA    CONTRL
        SYNC
        ANDA   #%11110111
        STA    CONTRL
        STA    STATUS
*
        LDX    #OUT
        JSR    EXCHG
        LDY    #IN2
        LDA    #15
        STA    CNT
LOOPZ   LDD    ,X++
        STD    OUT2M
        LDD    ,Y++
        STD    OUT1M
        DEC    CNT
        BNE    LOOPZ
        LDA    ACIASR
        LSRA
        BCC    LOOPX
        STD    OUT2M
```

```
        CMPA    #'E
        LBEQ    BEGIN
        JSR     CRLF
        JSR     PFX
REPEAT  LDX     #OUT
        LDY     #IN2
        LDA     #15
        STA     CNT
LOOP    LDD     ,Y++
        STD     OUT1M
        LDD     ,X++
        STD     OUT2M
        DEC     CNT
        BNE     LOOP
        LDA     ACIASR
        LSRA
        BCC     REPEAT
        LDA     ACIARX
        ANDA    #$7F
        CMPA    #'1
        LBEQ    SKP1
        CMPA    #'2
        LBEQ    SKP2
        LBRA    LOOPX
*
DSP     JSR     CRLF
        LDX     #MSG5
        LDA     #7
        STA     CNT
        JSR     DSPLY
        JSR     CRLF
        JSR     PFX
        JSR     CRLF
        LDX     #IN1
        JSR     ARAY
        JSR     CRLF
        JSR     CRLF
        LDX     #MSG6
        LDA     #7
        STA     CNT
        JSR     DSPLY
        JSR     CRLF
        LDX     #IN2
        JSR     ARAY
        JSR     CRLF
        JSR     PFX
SK9     LDX     #IN1
        LDY     #IN2
        LDA     #15
        STA     CNT
LOOPE   LDD     ,X++
        STD     OUT1M
        LDD     ,Y++
OK      RTS

        DEC     CNT
        BNE     LOOPE
        LDA     ACIASR
        LSRA
        BCC     SK9
        LDA     ACIARX
        ANDA    #$7F
        CMPA    #'1
        LBEQ    SKP1
        LBRA    SKP2
*
ARAY    LDA     #30
        STA     CNT
AGAIN   CLR     CNT1
LOOP2   CLR     CNT2
LOOP1   LDA     ,X
        INC     CNT2
        LSRA
        LSRA
        LSRA
        LSRA
        JSR     CONVA
        JSR     TXR
        LDA     ,X+
        ANDA    #$0F
        JSR     CONVA
        JSR     TXR
        INC     CNT2
        DEC     CNT
        BEQ     SET
        LDA     CNT2
        CMPA    #4
        BEQ     CHKT
        BRA     LOOP1
OVER    JSR     CRLF
        BRA     AGAIN
CHKT    LDA     CNT1
        CMPA    #4
        BEQ     OVER
        LDA     #$20
        JSR     TXR
        INC     CNT1
        BRA     LOOP2
SET     RTS
*
VALID   SUBA    #'0
        CMPA    #9
        BHI     CHK1
        RTS
CHK1    SUBA    #7
        CMPA    #$0F
        BLS     OK
        BRA     ERMSG3
MSG1    FCC     'Address Too Large'
```

```
*
PFX       LDX    #MSG7
          LDA    #6
          STA    CNT
          JSR    DSPLY
          RTS
*
ERMSG1    JSR    CRLF
          LDX    #MSG1
          LDA    #17
          STA    CNT
          JSR    CRLF
          JSR    DSPLY
          JSR    CRLF
          LBRA   INIT
ERMSG3    JSR    CRLF
          LDX    #MSG2
          LDA    #17
          STA    CNT
          JSR    CRLF
          JSR    DSPLY
          JSR    CRLF
          LBRA   INIT
*
          NAM    WIN015
*
*     ***************************************************************
*     *     WINOGRAD'S 15 POINT ALGORITHM                           *
*     ***************************************************************
          LDX    #AX
          LDY    #ARYIN
*    ************************
*    * INPUT REORDERING   *
*    ************************
          LDD     ,Y
          STD     ,X
          LDD    6,Y
          STD    2,X
          LDD   12,Y
          STD    4,X
          LDD   18,Y
          STD    6,X
          LDD   24,Y
          STD    8,X
          LDD   10,Y
          STD   10,X
          LDD   16,Y
          STD   12,X
          LDD   22,Y
          STD   14,X
          LDD   28,Y
          STD   16,X
          LDD    4,Y
*
|MSG2     FCC    'Invalid HEX Digit'
|MSG3     FCC    'Enter Response'
|MSG4     FCC    'Convolution '
|MSG5     FCC    'Array 1'
|MSG6     FCC    'Array 2'
|MSG7     FCC    'CONV: '
|MSG8     FCC    'Enter Values'
|*
|         ORG    $0081
|STACK    RMB    1
|TMP      RMB    2
|CNT      RMB    1
|CNT1     RMB    1
|CNT2     RMB    1
|STATUS   RMB    1
|FLAG     RMB    1
|IN1      RMB    30
|IN2      RMB    30
|OUT      RMB    30
|APYIN    RMB    30
|*
|         ORG    $FFFE
|STRT     EQU    $F800
|         END    START
|
|         LDD    20,Y
|         STD    20,X
|         LDD    26,Y
|         STD    22,X
|         LDD     2,Y
|         STD    24,X
|         LDD     8,Y
|         STD    26,X
|         LDD    14,Y
|         STD    28,X
|*
|*    ************************
|*    * 3-POINT PRE-WEAVE  *
|*    ************************
|SKP2     LDD    10,X
|         ADDD   20,X
|         BCS    JMP1
|         CMPD   #65521
|         BLO    JMP2
|JMP1     ADDD   #15
|JMP2     STD    TMP1
|         ADDD     ,X
|         BCS    JMP3
|         CMPD   #65521
|*
```

```
        BLO   JMP4
JMP3    ADDD  #15
JMP4    STD     ,X
        LDD   10,X
        SUBD  20,X
        BCC   JMP5
        ADDD  #65521
JMP5    STD   20,X
        LDD   TMP1
        STD   10,X
        LDD   12,X
        ADDD  22,X
        BCS   JMP6
        CMPD  #65521
        BLO   JMP7
JMP6    ADDD  #15
JMP7    STD   TMP1
        ADDD  2,X
        BCS   JMP8
        CMPD  #65521
        BLO   JMP9
JMP8    ADDD  #15
JMP9    STD   2,X
        LDD   12,X
        SUBD  22,X
        BCC   JMP10
        ADDD  #65521
JMP10   STD   22,X
        LDD   TMP1
        STD   12,X
        LDD   14,X
        ADDD  24,X
        BCS   JMP11
        CMPD  #65521
        BLO   JMP12
JMP11   ADDD  #15
JMP12   STD   TMP1
        ADDD  4,X
        BCS   JMP13
        CMPD  #65521
        BLO   JMP14
JMP13   ADDD  #15
JMP14   STD   4,X
        LDD   14,X
        SUBD  24,X
        BCC   JMP15
        ADDD  #65521
JMP15   STD   24,X
        LDD   TMP1
        STD   14,X
        LDD   16,X
        ADDD  26,X
        BCS   JMP16
        CMPD  #65521
        BLO   JMP17
JMP16   ADDD  #15
JMP17   STD   TMP1
        ADDD  6,X
        BCS   JMP18
        CMPD  #65521
        BLO   JMP19
JMP18   ADDD  #15
JMP19   STD   6,X
        LDD   16,X
        SUBD  26,X
        BCC   JMP20
        ADDD  #65521
JMP20   STD   26,X
        LDD   TMP1
        STD   16,X
        LDD   18,X
        ADDD  28,X
        BCS   JMP21
        CMPD  #65521
        BLO   JMP22
JMP21   ADDD  #15
JMP22   STD   TMP1
        ADDD  8,X
        BCS   JMP23
        CMPD  #65521
        BLO   JMP24
JMP23   ADDD  #15
JMP24   STD   8,X
        LDD   18,X
        SUBD  28,X
        BCC   JMP25
        ADDD  #65521
JMP25   STD   28,X
        LDD   TMP1
        STD   18,X
*       **************************
*       * 5-POINT PRE-WEAVE     *
*       **************************
        LDY   #Z
        LDD   2,X
        ADDD  8,X
        BCS   JMP26
        CMPD  #65521
        BLO   JMP27
JMP26   ADDD  #15
JMP27   STD   2,Y
        LDD   2,X
        SUBD  8,X
        BCC   JMP28
        ADDD  #65521
JMP28   STD   6,Y
        LDD   4,X
        ADDD  6,X
```

```
        BCS   JMP29
        CMPD  #65521
        BLO   JMP30
JMP29   ADDD  #15
JMP30   STD   4,Y
        LDD   6,X
        SUBD  4,X
        BCC   JMP31
        ADDD  #65521
JMP31   STD   10,Y
        ADDD  6,Y
        BCS   JMP32
        CMPD  #65521
        BLO   JMP33
JMP32   ADDD  #15
JMP33   STD   8,Y
        LDD   4,Y
        ADDD  2,Y
        BCS   JMP34
        CMPD  #65521
        BLO   JMP35
JMP34   ADDD  #15
JMP35   STD   TMP1
        ADDD    ,X
        BCS   JMP36
        CMPD  #65521
        BLO   JMP37
JMP36   ADDD  #15
JMP37   STD     ,Y
        LDD    2,Y
        SUBD   4,Y
        BCC   JMP38
        ADDD  #65521
JMP38   STD    4,Y
        LDD   TMP1
        STD    2,Y
*
        LDD   12,X
        ADDD  18,X
        BCS   JMP39
        CMPD  #65521
        BLO   JMP40
JMP39   ADDD  #15
JMP40   STD   14,Y
        LDD   12,X
        SUBD  18,X
        BCC   JMP41
        ADDD  #65521
JMP41   STD   18,Y
        LDD   14,X
        ADDD  16,X
        BCS   JMP42
        CMPD  #65521
        BLO   JMP43
JMP42   ADDD  #15
JMP43   STD   16,Y
        LDD   16,X
        SUBD  14,X
        BCC   JMP44
        ADDD  #65521
JMP44   STD   22,Y
        ADDD  18,Y
        BCS   JMP45
        CMPD  #65521
        BLO   JMP46
JMP45   ADDD  #15
JMP46   STD   20,Y
        LDD   16,Y
        ADDD  14,Y
        BCS   JMP47
        CMPD  #65521
        BLO   JMP48
JMP47   ADDD  #15
JMP48   STD   TMP1
        ADDD  10,X
        BCS   JMP49
        CMPD  #65521
        BLO   JMP50
JMP49   ADDD  #15
JMP50   STD   12,Y
        LDD   14,Y
        SUBD  16,Y
        BCC   JMP51
        ADDD  #65521
JMP51   STD   16,Y
        LDD   TMP1
        STD   14,Y
*
        LDD   22,X
        ADDD  28,X
        BCS   JMP52
        CMPD  #65521
        BLO   JMP53
JMP52   ADDD  #15
JMP53   STD   26,Y
        LDD   22,X
        SUBD  28,X
        BCC   JMP54
        ADDD  #65521
JMP54   STD   30,Y
        LDD   24,X
        ADDD  26,X
        BCS   JMP56
        CMPD  #65521
        BLO   JMP57
JMP56   ADDD  #15
JMP57   STD   28,Y
        LDD   26,X
```

```
        SUBD   24,X
        BCC    JMP58
        ADDD   #65521
JMP58   STD    34,Y
        ADDD   30,Y
        BCS    JMP59
        CMPD   #65521
        BLO    JMP60
JMP59   ADDD   #15
JMP60   STD    32,Y
        LDD    26,Y
        ADDD   28,Y
        BCS    JMP61
        CMPD   #65521
        BLO    JMP62
JMP61   ADDD   #15
JMP62   STD    TMP1
        ADDD   20,X
        BCS    JMP63
        CMPD   #65521
        BLO    JMP64
JMP63   ADDD   #15
JMP64   STD    24,Y
        LDD    26,Y
        SUBD   28,Y
        BCC    JMP65
        ADDD   #65521
JMP65   STD    28,Y
        LDD    TMP1
        STD    26,Y
*   ************************
*   * MULTIPLICATION     *
*   ************************
        CLRA
        STA    IND
        LDS    #Z
LOOP    LDA    FRD
        BEQ    OVER1
        LDY    #COEFR
        BRA    OVER2
OVER1   LDY    #COEFF
OVER2   LDA    IND
        LDD    A,Y
        STD    MLTR
        LDD      ,S
        STD    MLTN
        LDX    #MLTR
        LDY    #MLTN
        LDU    #PROD1
        CLR      ,U
        CLR     1,U
        LDA     1,X
        LDB     1,Y
        MUL
        STD     2,U
        LDA      ,X
        LDB     1,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC    SKIP3
        INC      ,U
SKIP3   LDA     1,X
        LDB      ,Y
        MUL
        ADDD    1,U
        STD     1,U
        BCC    SKIP4
        INC      ,U
SKIP4   LDA      ,X
        LDB      ,Y
        MUL
        ADDD     ,U
        STD      ,U
*
        LDA     1,U
        LDB    #15
        MUL
        ADDD    2,U
        BCS    SKIP6
        CMPD   #65521
        BLO    SKIP7
SKIP6   ADDD   #15
SKIP7   STD     2,U
        LDA      ,U
        LDY    #TEMP
        CLR      ,Y
        CLR     1,Y
        CLR     2,Y
        LDB    #15
        MUL
SKIPA   STD      ,Y
        LDA      ,Y
        BEQ    SKIPE
        LDB    #15
        MUL
        ADDD    1,Y
        BRA    SKIPD
SKIPE   LDD     1,Y
SKIPD   ADDD    2,U
        BCS    SKIPB
        CMPD   #65521
        BLO    SKIPC
SKIPB   ADDD   #15
SKIPC   STD      ,S++
        LDA    IND
        ADDA   #2
        STA    IND
```

```
          CMPA  #34
          LBLS  LOOP
*    **************************
*    * 5-POINT POST-WEAVE *
*    **************************
          LDX   #AX
          LDY   #Z
          LDD     ,Y
          STD     ,X
          ADDD   2,Y
          BCS   JMP67
          CMPD  #65521
          BLO   JMP68
JMP67     ADDD  #15
JMP68     STD   2,X
          LDD   8,Y
          ADDD  10,Y
          BCS   JMP69
          CMPD  #65521
          BLO   JMP70
JMP69     ADDD  #15
JMP70     STD   10,Y
          LDD   6,Y
          SUBD  8,Y
          BCC   JMP71
          ADDD  #65521
JMP71     STD   8,X
          LDD   2,X
          ADDD  4,Y
          BCS   JMP72
          CMPD  #65521
          BLO   JMP73
JMP72     ADDD  #15
JMP73     STD   TMP1
          LDD   2,X
          SUBD  4,Y
          BCC   JMP74
          ADDD  #65521
JMP74     STD   4,X
          SUBD  10,Y
          BCC   JMP75
          ADDD  #65521
JMP75     STD   6,X
          LDD   TMP1
          STD   2,X
          LDD   4,X
          ADDD  10,Y
          BCS   JMP76
          CMPD  #65521
          BLO   JMP77
JMP76     ADDD  #15
JMP77     STD   4,X
          LDD   2,X
          ADDD  8,X
          BCS   JMP78
          CMPD  #65521
          BLO   JMP79
JMP78     ADDD  #15
JMP79     STD   TMP1
          LDD   2,X
          SUBD  8,X
          BCC   JMP80
          ADDD  #65521
JMP80     STD   8,X
          LDD   TMP1
          STD   2,X
*
          LDD   12,Y
          STD   10,X
          ADDD  14,Y
          BCS   JUP67
          CMPD  #65521
          BLO   JUP68
JUP67     ADDD  #15
JUP68     STD   12,X
          LDD   20,Y
          ADDD  22,Y
          BCS   JUP69
          CMPD  #65521
          BLO   JUP70
JUP69     ADDD  #15
JUP70     STD   22,Y
          LDD   18,Y
          SUBD  20,Y
          BCC   JUP71
          ADDD  #65521
JUP71     STD   18,X
          LDD   12,X
          ADDD  16,Y
          BCS   JUP72
          CMPD  #65521
          BLO   JUP73
JUP72     ADDD  #15
JUP73     STD   TMP1
          LDD   12,X
          SUBD  16,Y
          BCC   JUP74
          ADDD  #65521
JUP74     STD   14,X
          SUBD  22,Y
          BCC   JUP75
          ADDD  #65521
JUP75     STD   16,X
          LDD   TMP1
          STD   12,X
          LDD   14,X
          ADDD  22,Y
          BCS   JUP76
```

```
        CMPD   #65521
        BLO    JUP77
JUP76   ADDD   #15
JUP77   STD    14,X
        LDD    12,X
        ADDD   18,X
        BCS    JUP78
        CMPD   #65521
        BLO    JUP79
JUP78   ADDD   #15
JUP79   STD    TMP1
        LDD    12,X
        SUBD   18,X
        BCC    JUP80
        ADDD   #65521
JUP80   STD    18,X
        LDD    TMP1
        STD    12,X
*
        LDD    24,Y
        STD    20,X
        ADDD   26,Y
        BCS    SKP67
        CMPD   #65521
        BLO    SKP68
SKP67   ADDD   #15
SKP68   STD    22,X
        LDD    32,Y
        ADDD   34,Y
        BCS    SKP69
        CMPD   #65521
        BLO    SKP70
SKP69   ADDD   #15
SKP70   STD    34,Y
        LDD    30,Y
        SUBD   32,Y
        BCC    SKP71
        ADDD   #65521
SKP71   STD    28,X
        LDD    22,X
        ADDD   28,Y
        BCS    SKP72
        CMPD   #65521
        BLO    SKP73
SKP72   ADDD   #15
SKP73   STD    TMP1
        LDD    22,X
        SUBD   28,Y
        BCC    SKP74
        ADDD   #65521
SKP74   STD    24,X
        SUBD   34,Y
        BCC    SKP75
        ADDD   #65521
SKP75   STD    26,X
        LDD    TMP1
        STD    22,X
        LDD    24,X
        ADDD   34,Y
        BCS    SKP76
        CMPD   #65521
        BLO    SKP77
SKP76   ADDD   #15
SKP77   STD    24,X
        LDD    22,X
        ADDD   28,X
        BCS    SKP78
        CMPD   #65521
        BLO    SKP79
SKP78   ADDD   #15
SKP79   STD    TMP1
        LDD    22,X
        SUBD   28,X
        BCC    SKP80
        ADDD   #65521
SKP80   STD    28,X
        LDD    TMP1
        STD    22,X
*   ************************
*   * 3-POINT POST-WEAVE *
*   ************************
        LDD     ,X
        ADDD   10,X
        BCS    JMP81
        CMPD   #65521
        BLO    JMP82
JMP81   ADDD   #15
JMP82   STD    10,X
        LDD     2,X
        ADDD   12,X
        BCS    JMP83
        CMPD   #65521
        BLO    JMP84
JMP83   ADDD   #15
JMP84   STD    12,X
        LDD     4,X
        ADDD   14,X
        BCS    JMP85
        CMPD   #65521
        BLO    JMP86
JMP85   ADDD   #15
JMP86   STD    14,X
        LDD     6,X
        ADDD   16,X
        BCS    JMP87
        CMPD   #65521
        BLO    JMP88
JMP87   ADDD   #15
```

```
JMP88    STD   16,X
         LDD    8,X
         ADDD  18,X
         BCS   JMP89
         CMPD  #65521
         BLO   JMP90
JMP89    ADDD  #15
JMP90    STD   18,X
         LDD   10,X
         ADDD  20,X
         BCS   JMP91
         CMPD  #65521
         BLO   JMP92
JMP91    ADDD  #15
JMP92    STD   TMP1
         LDD   10,X
         SUBD  20,X
         BCC   JMP911
         ADDD  #65521
JMP911   STD   20,X
         LDD   TMP1
         STD   10,X
         LDD   12,X
         ADDD  22,X
         BCS   JMP922
         CMPD  #65521
         BLO   JMP93
JMP922   ADDD  #15
JMP93    STD   TMP1
         LDD   12,X
         SUBD  22,X
         BCC   JMP94
         ADDD  #65521
JMP94    STD   22,X
         LDD   TMP1
         STD   12,X
         LDD   14,X
         ADDD  24,X
         BCS   JMP95
         CMPD  #65521
         BLO   JMP96
JMP95    ADDD  #15
JMP96    STD   TMP1
         LDD   14,X
         SUBD  24,X
         BCC   JMP97
         ADDD  #65521
JMP97    STD   24,X
         LDD   TMP1
         STD   14,X
         LDD   16,X
         ADDD  26,X
         BCS   JMP98
         CMPD  #65521
         BLO   JMP99
JMP98    ADDD  #15
JMP99    STD   TMP1
         LDD   16,X
         SUBD  26,X
         BCC   JMP100
         ADDD  #65521
JMP100   STD   26,X
         LDD   TMP1
         STD   16,X
         LDD   18,X
         ADDD  28,X
         BCS   JMP101
         CMPD  #65521
         BLO   JMP102
JMP101   ADDD  #15
JMP102   STD   TMP1
         LDD   18,X
         SUBD  28,X
         BCC   JMP103
         ADDD  #65521
JMP103   STD   28,X
         LDD   TMP1
         STD   18,X
*    ************************
*    *  OUTPUT SHUFFLE     *
*    ************************
         LDX   #AX
         LDY   #OUT
         LDD     ,X
         STD     ,Y
         LDD   12,X
         STD    2,Y
         LDD   24,X
         STD    4,Y
         LDD    6,X
         STD    6,Y
         LDD   18,X
         STD    8,Y
         LDD   20,X
         STD   10,Y
         LDD    2,X
         STD   12,Y
         LDD   14,X
         STD   14,Y
         LDD   26,X
         STD   16,Y
         LDD    8,X
         STD   18,Y
         LDD   10,X
         STD   20,Y
         LDD   22,X
         STD   22,Y
         LDD    4,X
```

```
          STD    24,Y
          LDD    16,X
          STD    26,Y
          LDD    28,X
          STD    28,Y
*
COEFF     FDB      1,6379,13376
          FDB  19136,18005,48647
          FDB  32759,8192,45457
          FDB  36817,5753,25311
*
COEFR     FDB  16087,29032,8748
          FDB  23174,43615,1465
          FDB  61153,5460,18364
          FDB  46773,20640,5493
          FDB  6552,57331,37975
          FDB  28122,34561,24521
          FDB  29504,28641,12521
          FDB  5913,24748,21938
          END  STRT
```

## Appendix-E

Backplane wiring connections for the parallel microprocessor system

Backplane pin connections for tne parallel microprocessor system.
Flow of data from SOURCE (TX) -> DESTINATION (RX)

BOARD NO A - ( PROCESSOR NO 1 2 3)

SIDE A

| PIN# | FUNCTION | PROCESSOR# |
|---|---|---|
| 1-8 | DATA IN | 1 2 3 |
| 9-14 | CLOCK | 1 2 3 |
| 15-22 | DATA OUT | 1 2 3 |
| 23-28 | OE | 1 2 3 |
| 29-36 | TX | 1 -> 6 |
| 37-38 | CLOCK | 1 -> 6 |
| 39-46 | RX | 6 -> 1 |
| 47-48 | CLOCK | 6 -> 1 |
| 49-56 | TX | 2 -> 7 |
| 57-60 | CLOCK | 2 -> 7 |
| 49-56 | TX | 2 -> 5 |
| 57-60 | CLOCK | 2 -> 5 |
| 61-68 | RX | 7 -> 2 |
| 69-70 | CLOCK | 7 -> 2 |
| 71-78 | RX | 5 -> 2 |
| 79-80 | CLOCK | 5 -> 2 |
| 81-88 | TX | 3 -> 8 |
| 89-92 | CLOCK | 3 -> 8 |
| 81-88 | TX | 3 -> 4 |
| 89-92 | CLOCK | 3 -> 4 |

SIDE B

| PIN# | FUNCTION | PROCESSOR# |
|---|---|---|
| 1-8 | RX | 8 -> 3 |
| 9-10 | CLOCK | 8 -> 3 |
| 11-18 | RX | 4 -> 3 |
| 19-20 | CLOCK | 4 -> 3 |
| 74 | STATUS | OUT |
| 75 | SYNC | OUT |
| 76 | SYNC | IN |
| 77 | SYSTEM | CLOCK |
| 78 | HALT | |
| 79 | RESET | |
| 29-61-93 | +VCC | |
| 32-64-96 | GROUND | |

BOARD NO B - (PROCESSOR 4 5 16)

SIDE A

| PIN# | FUNCTION | PROCESSOR# |
|---|---|---|
| 1-8 | DATA IN | 4 5 |
| 9-12 | CLOCK | 4 5 |
| 13-20 | DATA OUT | 4 5 |
| 21-24 | OE | 4 5 |
| 25-32 | TX | 4 -> 9 |
| 33-36 | CLOCK | 4 -> 9 |
| 25-32 | TX | 4 -> 3 |
| 33-36 | CLOCK | 4 -> 3 |
| 37-44 | RX | 9 -> 4 |
| 45-46 | CLOCK | 9 -> 4 |
| 47-54 | RX | 3 -> 4 |
| 55-56 | CLOCK | 3 -> 4 |
| 57-64 | TX | 5 -> 10 |
| 65-68 | CLOCK | 5 -> 10 |

SIDE B

| PIN# | FUNCTION |
|---|---|
| 74 | STATUS OUT |
| 75 | SYNC OUT |
| 76 | SYNC IN |
| 77 | SYSTEM CLOCK |
| 78 | HALT |
| 79 | RESET |
| 29-61-93 | +VCC |
| 32-64-96 | GROUND |

| | | | |
|---|---|---|---|
| 57-64 | TX | 5 -> 2 | |
| 65-68 | CLOCK | 5 -> 2 | |
| 69-76 | RX | 10 -> 5 | |
| 77-78 | CLOCK | 10 -> 5 | |
| 79-86 | RX | 2 -> 5 | |
| 87-88 | CLOCK | 2 -> 5 | |

BOARD NO C - (PROCESSOR 6 7 8)

| SIDE A | | | SIDE B | | |
|---|---|---|---|---|---|
| PIN# | FUNCTION | PROCESSOR# | PIN# | FUNCTION | PROCESSOR# |
| 1-8 | DATA IN | 6 7 8 | 1-8 | RX | 10 -> 7 |
| 9-14 | CLOCK | 6 7 8 | 9-10 | CLOCK | 10 -> 7 |
| 15-22 | DATA OUT | 6 7 8 | 11-18 | TX | 8 -> 13 |
| 23-28 | OE | 6 7 8 | 19-24 | CLOCK | 8 -> 13 |
| 29-36 | TX | 6 -> 11 | 11-18 | TX | 8 -> 3 |
| 37-40 | CLOCK | 6 -> 11 | 19-24 | CLOCK | 8 -> 3 |
| 29-36 | TX | 6 -> 1 | 11-18 | TX | 8 -> 9 |
| 37-40 | CLOCK | 6 -> 1 | 19-24 | CLOCK | 8 -> 9 |
| 41-48 | RX | 11 -> 6 | 25-32 | RX | 13 -> 8 |
| 49-50 | CLOCK | 11 -> 6 | 33-34 | CLOCK | 13 -> 8 |
| 51-58 | RX | 1 -> 6 | 35-42 | RX | 2 -> 8 |
| 59-60 | CLOCK | 1 -> 6 | 43-44 | CLOCK | 3 -> 8 |
| 61-68 | TX | 7 -> 12 | 45-52 | RX | 9 -> 8 |
| 69-74 | CLOCK | 7 -> 12 | 53-54 | CLOCK | 9 -> 8 |
| 61-68 | TX | 7 -> 2 | | | |
| 69-74 | CLOCK | 7 -> 2 | 74 | STATUS OUT | |
| 61-68 | TX | 7 -> 10 | 75 | SYNC OUT | |
| 69-74 | CLOCK | 7 -> 10 | 76 | SYNC IN | |
| 75-52 | RX | 12 -> 7 | 77 | SYSTEM CLOCK | |
| 83-84 | CLOCK | 12 -> 7 | 78 | HALT | |
| 85-92 | RX | 2 -> 7 | 79 | RESET | |
| 93-94 | CLOCK | 2 -> 7 | 29-61-93 | +VCC | |
| | | | 32-64-96 | GROUND | |

BOARD NO D - (PROCESSOR 9 10 17)

| SIDE A | | | SIDE B | | |
|---|---|---|---|---|---|
| PIN# | FUNCTION | PROCESSOR# | PIN# | FUNCTION | PROCESSOR# |
| 1-8 | DATA IN | 9 10 | 1-8 | RX | 5 -> 10 |
| 9-12 | CLOCK | 9 10 | 9-10 | CLOCK | 5 -> 10 |
| 13-20 | DATA OUT | 9 10 | 11-18 | RX | 7 -> 10 |
| 21-24 | OE | 9 10 | 19-20 | CLOCK | 7 -> 10 |
| 25-32 | TX | 9 -> 14 | | | |
| 33-38 | CLOCK | 9 -> 14 | 74 | STATUS OUT | |
| 25-32 | TX | 9 -> 4 | 75 | SYNC OUT | |
| 33-38 | CLOCK | 9 -> 4 | 76 | SYNC IN | |
| 25-32 | TX | 9 -> 8 | 77 | SYSTEM CLOCK | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 33-38 | CLOCK | 9 | -> | 8 | 78 | HALT | |
| 39-46 | RX | 14 | -> | 9 | 79 | RESET | |
| 47-48 | CLOCK | 14 | -> | 9 | 29-61-93 | +VCC | |
| 49-56 | RX | 4 | -> | 9 | 32-64-96 | GROUND | |
| 57-58 | CLOCK | 4 | -> | 9 | | | |
| 59-66 | RX | 8 | -> | 9 | | | |
| 67-68 | CLOCK | 8 | -> | 9 | | | |
| 69-76 | TX | 10 | -> | 15 | | | |
| 77-82 | CLOCK | 10 | -> | 15 | | | |
| 69-76 | TX | 10 | -> | 5 | | | |
| 77-82 | CLOCK | 10 | -> | 5 | | | |
| 69-76 | TX | 10 | -> | 7 | | | |
| 77-82 | CLOCK | 10 | -> | 7 | | | |
| 83-90 | RX | 15 | -> | 10 | | | |
| 91-92 | CLOCK | 15 | -> | 10 | | | |

BOARD NO E - (PROCESSOR 11 12 13)

SIDE A

| PIN# | FUNCTION | PROCESSOR# | | |
|---|---|---|---|---|
| 1-8 | DATA IN | 11 | 12 | 13 |
| 9-14 | CLOCK | 11 | 12 | 13 |
| 15-22 | DATA OUT | 11 | 12 | 13 |
| 23-28 | OE | 11 | 12 | 13 |
| 29-36 | TX | 11 | -> | 6 |
| 37-38 | CLOCK | 11 | -> | 6 |
| 39-46 | RX | 6 | -> | 11 |
| 47-48 | CLOCK | 6 | -> | 11 |
| 49-56 | TX | 12 | -> | 7 |
| 57-60 | CLOCK | 12 | -> | 7 |
| 49-56 | TX | 12 | -> | 15 |
| 57-60 | CLOCK | 12 | -> | 15 |
| 61-68 | RX | 7 | -> | 12 |
| 69-70 | CLOCK | 7 | -> | 12 |
| 71-78 | RX | 15 | -> | 12 |
| 79-80 | CLOCK | 15 | -> | 12 |
| 81-88 | TX | 13 | -> | 8 |
| 89-92 | CLOCK | 13 | -> | 8 |
| 81-88 | TX | 13 | -> | 14 |
| 89-92 | CLOCK | 13 | -> | 14 |

SIDE B

| PIN# | FUNCTION | PROCESSOR# | | |
|---|---|---|---|---|
| 1-8 | RX | 9 | -> | 13 |
| 9-10 | CLOCK | 8 | -> | 13 |
| 11-18 | RX | 14 | -> | 13 |
| 19-20 | CLOCK | 14 | -> | 13 |
| 74 | STATUS OUT | | | |
| 75 | SYNC OUT | | | |
| 76 | SYNC IN | | | |
| 77 | SYSTEM CLOCK | | | |
| 78 | HALT | | | |
| 79 | RESET | | | |
| 29-61-93 | +VCC | | | |
| 32-64-96 | GROUND | | | |

BOARD NO F - (PROCESSOR 14 15 18)

SIDE A

| PIN# | FUNCTION | PROCESSOR# | |
|---|---|---|---|
| 1-8 | DATA IN | 14 | 15 |
| 9-12 | CLOCK | 14 | 15 |

SIDE B

| PIN# | FUNCTION |
|---|---|
| 74 | STATUS OUT |
| 75 | SYNC OUT |

| | | | | | |
|---|---|---|---|---|---|
| 13-20 | DATA OUT | 14 15 | 76 | SYNC IN | |
| 21-24 | OE | 14 15 | 77 | SYSTEM CLOCK | |
| 25-32 | TX | 14 -> 9 | 29-61-93 | +VCC | |
| 33-36 | CLOCK | 14 -> 9 | 32-64-96 | GROUND | |
| 25-32 | TX | 14 -> 13 | | | |
| 33-36 | CLOCK | 14 -> 13 | | | |
| 37-44 | RX | 9 -> 14 | | | |
| 45-46 | CLOCK | 9 -> 14 | | | |
| 47-54 | PX | 13 -> 14 | | | |
| 55-56 | CLOCK | 13 -> 14 | | | |
| 57-64 | TX | 15 -> 10 | | | |
| 65-68 | CLOCK | 15 -> 10 | | | |
| 57-64 | TX | 15 -> 12 | | | |
| 65-68 | CLOCK | 15 -> 12 | | | |
| 69-76 | RX | 10 -> 15 | | | |
| 77-78 | CLOCK | 10 -> 15 | | | |
| 79-86 | RX | 12 -> 15 | | | |
| 87-88 | CLOCK | 12 -> 15 | | | |

## CONTROL BOARD

### SIDE A

| PIN# | FUNCTION | PROCESSOR# | PIN# | FUNCTION | PROCESSOR# |
|---|---|---|---|---|---|
| 1-8 | DATA OUT | | 47-54 | DATA IN | |
| 17-18 | CLOCK | 1 | 63-64 | OE | 1 |
| 19-20 | CLOCK | 4 | 65-66 | OE | 7 |
| 21-22 | CLOCK | 7 | 67-68 | OE | 13 |
| 23-24 | CLOCK | 10 | 69-70 | OE | 4 |
| 25-26 | CLOCK | 13 | 71-72 | OE | 10 |
| 27-28 | CLOCK | 6 | 73-74 | OE | 11 |
| 29-30 | CLOCK | 9 | 75-76 | OE | 2 |
| 31-32 | CLOCK | 12 | 77-78 | OE | 8 |
| 33-34 | CLOCK | 15 | 79-80 | OE | 14 |
| 35-36 | CLOCK | 3 | 81-82 | OE | 5 |
| 37-38 | CLOCK | 11 | 83-84 | OE | 6 |
| 39-40 | CLOCK | 14 | 85-86 | OE | 12 |
| 41-42 | CLOCK | 2 | 87-88 | OE | 3 |
| 43-44 | CLOCK | 5 | 89-90 | OE | 9 |
| 45-46 | CLOCK | 8 | 91-92 | OE | 15 |

### SIDE B

| | |
|---|---|
| 1-6 | STATUS IN |
| 7-12 | SYNC IN |
| 13 | SYNC OUT |
| 14-19 | SYSTEM CLOCK OUTPUT TO PROCESSORS |
| 20 | RESET TO OTHER BOARDS |
| 21 | HALT TO OTHER BOARDS |
| 27 | -9V FOR RS-232 RX |
| 29-61-93 | +VCC 5V POWER FOR ALL BOARDS |
| 32-64-96 | GROUND |

## REFERENCES


(1) Bergland, G.D: 'Fast Fourier Transform Hardware Implementation - An Overview', IEEE Trans. Audio Electroacoustics, Jun. 1969, vol AU-17, pp. 104-108.

(2) Brigham, E.O.: The Fast Fourier Transform, Prentice Hall Inc., Englewood Cliffs, N.J., 1974.

(3) Winograd, S.: 'On Computing the Discrete Fourier Transform', Math. Comput. 1978, vol. 32, pp. 175-199.

(4) Winograd, S.: 'On Computing the Discrete Fourier Transform', Proc. Nat. Acad. Sci., 1976, vol. 73, pp. 1005-1006.

(5) Martin, S.C.P.: Number Theoretic Transform Implementation using Microprocessors, Ph.D. thesis, 1980, Univ. of Durham.

(6) Martin, S.C.P, and Stanier, B.J.: 'Microprocessor Implementation of Number Theoretic Transforms', Electron. Cir. and Syst., Jan. 1979, vol. 3, pp. 21-26.

(7) McClellan J.H., and Rader C.M.: Number Theory in Digital Signal Processing, Prentice Hall Inc., Englewood Cliffs, N.J. 1979.

(8) Cooley, J.W., and Tukey, J.W.: 'An Algorithm for the Machine Calculation of Complex Fourier Series', Math. Comput., 1965, vol. 19, pp. 297-301.

(9) Agarwal, R.C., and Burrus, C.S.: 'Number Theoretic Transforms to Implement Fast Digital Convolution', Proc. IEEE, 1975, vol. 63, pp. 550-560.

(10) Vanwormhoudt, M.C.: 'On Number Theoretic Fourier Transform in Residue Class Rings', Corresp., IEEE Trans., 1977, vol. ASSP-25, pp. 585-586.

(11) Leibowitz, L.M.: 'Fast Convolution by Number Theoretic Transforms', NRL Report 7924, Sept. 1975.

(12) Rader, C.M.: 'Discrete Convolutions via Mersenne Transforms', IEEE Trans. Comput., 1972, vol. C-21, pp. 1269-1273.

(13) Agarwal, R.C., and Burrus, C.S.: 'Fast Convolution Using Fermat Number Transform with Applications to Digital Filtering', IEEE Trans., 1974, vol. ASSP-22, pp. 87-97.

(14) Leibowitz, L.M.: 'A Simplified Arithmetic for the Fermat Number Transform', IEEE Trans., 1976, vol. ASSP-24, pp. 356-359.

(15) McClellan, J.H.: 'Hardware Realisation of a Fermat Number Transform', IEEE Trans., 1976, vol. ASSP-24, pp. 216-225.

(16) Bywater, R.E.H.: Hardware/Software Design of Digital Systems, Prentice Hall Inc. Englewood Cliffs, N.J., 1981.

(17) Lewin, D.: Theory and Design of Digital Computers, Thomas Nelson and Sons Ltd., 1972.

(18) Parasuraman, B.: 'Hardware Multiplication Techniques for Microprocessor Systems', Computer Design, 1977, pp. 75-82.

(19) Harman, M.G.: 'An Attempt to Design an Improved Multipication System', IEEE Trans. Comput., 1968, vol. C-17, pp. 1090.

(20) Rabiner, L.R., and Gold, B.: Theory and Application of Digital Signal Processing, Prentice Hall Inc. Englewood Cliffs, N.J., 1975.

(21) Chu, Y.: Digital Computer Design Fundamentals, McGraw Hill, 1962.

(22) Hayes, J.P.: Computer Architecture and Organisation, McGraw Hill, Kogakusha Ltd., 1978.

(23) Gosling, J.B.: Design of Arithmetic Units for Digital Computers, McMillan Press Ltd., London, 1980.

(24) Nussbaumer, H.J.: 'Fast Multipliers for Number Theoretic Transforms', IEEE Trans. C-27, Aug. 1978, pp. 764-765.

(25) Brubaker, T.A., and Becker, J.C.: 'Multiplication Using Logarithms Implemented with Read-Only Memory', IEEE Trans. Comput., vol. C-24, pp. 761-765.

(26) Chang, T.: 'Binary Read-Only-Memory Multiplier', Eletron. Lett., 13 Dec. 1973, vol. 9, pp. 580-581.

(27) Johnson, N.: 'Improved Binary Multiplication System', Electron. Lett., 11 Jan. 1973, vol. 9, pp. 6-7.

(28) Davies, A.C.: 'Trade-offs in Fixed-Point Multiplication Algorithms for Microprocessors', Comput. and Dig. Techniques, 1979, vol. 2, pp. 105-112.

(29) Weed, M.: 'Clockless Multiplication and Division Circuits', BYTE, Dec. 1978, pp. 128-136.

(30) Artwick, B.A.: Microcomputer Interfacing, Prentice Hall Inc., Englewood Cliffs, N.J., 1980.

(31) Davies, A.C., Fung, Y.T.: 'Interfacing a Hardware Multiplier to a General-Purpose Microprocessor', Microprocessors, 1977, vol. 1, pp. 425-432.

(32) Evanczuk, S.: 'Josephson Chip Multiplies Ultra Fast', Electronics, 14 July, 1982, pp. 48-50.

(33) Bate, J., and Burkowski, F.: 'A High Speed Extended Precision Multiplier for a Microprocessor', Proc. Int. Symp. on Mini and Micro Computers, Montreal, Canada, 11-18 Nov. 1977, pp. 10-13.

(34) Robinson, D.: 'Hardware Multiplier/Divider Unit for 8-bit

Microprocessor Systems', New Electronics (G.B), Mar. 1979, vol. 12, pp. 20.

(35) Mick, J., and Springer, J.: 'An Integrated Circuit, High-Speed Serial-Parallel Multiplier', Apr. 1976, pp. 42, 46.

(36) Rollenhagen, D.C., Kimball, R.M., and Shay, H.P.: 'LSI Multiplier-Divider for 8080', Proc. IEEE 1977 Nat. Aerospace and Electron. Conf., NAECON 1977, Dayton, Ohio, U.S.A., 17-19 May, pp. 887-892.

(37) Day, M.J.: 'Faster Multiply with Microprocessor Hardware Multiply Device', Electron (G.B.), 27 Feb. 1978, pp. 39.

(38) Waser, S., Newton, V.: 'Increasing Multiplication Speed', Electron (G.B.), 12 Dec. 1977, pp. 57-58.

(39) Rohr, P.: 'LSI Multipliers: The Second Generation', 1979 Int. Micro and Mini Computer Conf., Houston, Texas, U.S.A., Nov. 1979, pp. 140-143.

(40) Ambikairajah, E., and Carey, M.J.: 'Technique for Performing Multiplication on a 16-bit Microprocessor using extension of Booth's Algorithm', Electron. Lett., 17 Jan. 1980, vol. 16, pp. 53-54.

(41) Giest, D.J.: 'MOS Processor Picks up Speed with Bipolar Multipliers', Electronics (U.S.A.), July 1977, vol. 50, pp. 113-115.

(42) '16*16-bit Multipliers meet Military/Commercial High Speed Applications', Comput. Des. (U.S.A.), Aug. 1976, vol. 15, pp. 50.

(43) McCrea, P.G., and Matheson, W.S.: 'Design of High Speed Fully Serial Tree Multiplier', IEEE. Proc. Jan. 1981, vol. 128, pp. 13-20.

(44) Advanced Micro Devices: 'AM25S558, Eight-bit by Eight-bit Combinational Multiplier', Preliminary data sheet.

(45) Flores, I.: The Logic of Computer Arithmetic, Prentice Hall Inc., Englewood Cliffs, N.J., 1963.

(46) Booth, A.D, and Booth, K.H.V.: Automatic Digital Calculators, Butterworth and Co. Ltd., London, 1965.

(47) Abd-Alla, A.M, and Meltzer, A.C.: Principles of Digital Computer Design, Prentice Hall, Englewood Cliffs, N.J., 1976, vol. 1.

(48) Renold, A.: Comparison of some 8-bit Microprocessors by means of Benchmark Programs, Mitt. Agon. (Switzerland), Oct. 1981, pp. 71-75.

(49) Kolba, D.P., and Parks, T.W.: 'A Prime Factor FFT Algorithm Using High-Speed Convolution', IEEE Trans., vol. ASSP-25, Aug. 1977, pp. 281-294.

(50) Morris, L.R.: 'A Comparative Study of Time Efficient FFT and WFTA Programs for General Purpose Computers', IEEE Trans., vol. ASSP-26, Apr. 1978, pp.141-150.

(51) Silverman, H.F.: 'An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)', IEEE Trans., vol. ASSP-25, Apr. 1977, pp. 152-165.

'Correction and an Addendum to an Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)', IEEE Trans. ASSP-26, 1978, pp. 268.

(52) Nawab, H., and McClellan, J.H.: 'Bounds on the Minimum Number of Data Transfers in WFTA and FFT Programs', IEEE Trans., vol. ASSP-27, Aug. 1979, pp. 394-398.

(53) Bailey, D.: 'Winograd's Algorithm Applied to Number-Theoretic

Transforms', Electron. Lett., 1 Sept. 1977, vol. 13 pp. 548-549.

(54) Texas Instruments Ltd.: TMS9900 Microprocessor Data Manual, Aug. 1976.

(55) Texas Instruments Ltd.: TMS990/100M Microcomputer User's Guide, Mar. 1978.

(56) Moore, C.H.: 'FORTH: A New Way to Program a Minicomputer', Astron. Astrophy. Suppl., 1974, vol. 15, pp. 497-511.

(57) Brodie, L.: Starting Forth, Prentice Hall Inc., Englewood Cliffs N.J., 1981.

(58) Smith, M.F.: 'Comparative Software Analysis of the MC6809 Microprocessor', Microprocessors and Micro Systems, vol. 5, Nov. 1981, pp. 401-404.

(59) Mintzer, F.: 'Parallel and Cascade Microprocessor Implementation for Digital Signal Processing', IEEE Trans. ASSP-29, Oct. 1981, pp. 1018-1027.

(60) Zohar, S.: 'Outline of a Fast Hardware Implementation of Winograd's DFT Algorithm', IEEE ICASSP, Apr. 1980, vol. 3, pp.796-799.

(61) Mintzer, F.: 'Attributes of Parallel and Cascade Microprocessor Implementations of Digital Signal Processing', IEEE Int. Conf. ASSP, April 1980, ICASSP, vol. 3, pp. 912-915.

(62) Duff, M.J.B.: 'Array Processing', Electronics and Power Nov./Dec. 1980, pp. 888-893.

(63) Bain, W.L., and Jump, J.R.: 'Hardware Scheduling Strategies for Systems with many Processors', Proc. Int. Conf. Parallel Processing, Bellaire, MI, U.S.A., Aug. 1978, pp. 184-187.

(64) Bellm, H., and Sauer, A.: 'Methods of Data Exchange Between

Microcomputers', Proc. Microprocessing and Microprogramming, Amsterdam, N. Holland, 3-6 Oct. 1977, Microcomputer Archit., pp. 16-22.

(65) Arden, B.E., and Berenbaum, A.D.: 'A Multi-Microprocessor Computer System Architecture', Proc. 5th Symp. Operating System Principles, Operating System Rev., Nov. 1975, vol. 9, pp. 114-121.

(66) Enslow, P.H.: Multiprocessor and Parallel Processing, John Wiley and Sons, 1974.

(67) Pollard, L.H.: 'Multiprocessing with the TI9900', Eleventh Ann. Asilomer Conf. Circuits Systems and Computers, Pacific Grove, CA, U.S.A., 7-9 Nov. 1977, pp. 461-465.

(68) Hoffner, Y., and Smith, M.F.: 'Communication Between two Microprocessors Through Common Memory', Microprocessors and Microsystems, July/Aug. 1982, vol. 6, pp. 303-308.

(69) Witten, I.H., and Jenkins, R.L.: 'Processor-Processor Dialogue Through Existing Input-Output Channels', Computer and Digital Techniques, Oct. 1978, vol. 1, pp. 125-130.

(70) Parkinson, D.: 'An Introduction to Array Processors', Syst. Int. (G.B), Nov. 1977, vol. 5, pp. 21-23.

(71) Caprani, O., Jensen, K.H., and Ougaard, U.: 'Microprocessors Connected to a Common Memory', Microprocessor and Microprogramming Amsterdam, Netherland, 3-6 Oct. 1977, Euromicro Symp., Microcomputer Architec., pp. 175-181.

(72) Hughes, P., and Doone, T.: 'Multiprocessor Systems', Syst. Int. (G.B), Feb. 1978, vol. 6, pp. 20-21.

(73) Raphael, H.: 'Multiprocessor Techniques for uP Systems', Electron. Eng. (G.B), 1978, vol. 50, pp. 65-67.

(74) Tanabe, K., and Matsumoto, K.: '16-bit Microprocessor with Dual Bus Architecture', Proc. Spring COMPCON, 1979, San Francisco, 26 Feb. to 1 Mar. 1979, N.Y., U.S.A., pp. 98-101.

(75) Crushman, R.H.: 'uP/uC Chip Directory', EDN, Oct. 1979, pp. 133-240.

(76) Crushman, R.H., and Bucker, J.: 'EDN Seventh Annual uP/uC Chip Directory', EDN, Nov. 1979, pp. 94-211.

(77) Scales, H.: 'Multiprocessing with the Motorola's MC6809E', BYTE, Jul. 1981, pp. 136-156.

(78) Leventhal, L.A.: 6809 Assembly Language Programming, Osborne McGraw Hill, 1981.

(79) Motorola Semiconductors Ltd.: MC6809 Data sheet.

(80) Leibowitz, L.M.: 'A Binary Arithmetic for the Fermat Number Transform', NRL Report 7971, 18th Mar. 1976.

(81) Gallacher, J.: 'Processor-Processor Communication', Microprocessors and Microsystems, Sept. 1979, vol. 3, pp. 317-320.

(82) Fronheiser, K.: 'Device Operation and System Implementation of the Asynchronous Comminications Interface Adapter (MC6850) Motorola Semiconductors Ltd., Application Note AN-754.

(83) Motorola Semiconductors Ltd., MC6850 Data Sheet.

(84) Wakerly, J.: 'Serial Communications', Microprocessor and Microsystems, 1981, vol. 5, pp. 247-253.

(85) Motorola Semiconductors Ltd.: MC14411 Data Sheet.

(86) Texas Instruments Ltd.: The TTL Data Book for Design Engineers.

(87) Patel, J. H.: Processor-Memory Interconnection for Multiprocessors', Proc. 6th Ann. Symp. on Computers Architec.,

Philadelphia, PA, 23-25 Apr. 1979, N.Y., U.S.A, pp. 168-177.

(88) Davidson, K.A., Parsons R.L. etal: 'Processor-to-Processor Inter-Communication Employing a Common Storage Module', IBM Tech. Disc. Bull., Mar. 1979, vol. 21, pp. 3959-3960.

(89) Zaks, R.: Programming the Z80, 1982, SYBEX Inc.

(90) Signetics: 8X300 Data Sheet.

(91) National Semiconductors Ltd.: COP402 Data Sheet.

(92) Zaks, R.: Programming the 6502, 1978, SYBEX Inc.