Design of microprocessor-based hardware for number theoretic transform implementation

Shamim, Anwar Ahmed

How to cite:
Shamim, Anwar Ahmed (1983) Design of microprocessor-based hardware for number theoretic transform implementation, Durham theses, Durham University. Available at Durham E-Theses Online: http://etheses.dur.ac.uk/7213/

Use policy
The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full Durham E-Theses policy for further details.
DESIGN OF MICROPROCESSOR-BASED HARDWARE FOR
NUMBER THEORETIC TRANSFORM IMPLEMENTATION

by

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

The copyright of this thesis rests with the author.
No quotation from it should be published without
his prior written consent and information derived
from it should be acknowledged.

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

28. NOV 1983
Thesis
1985/SHA
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 \( \mathbb{Z}_M \), 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
ABSTRACT

ACKNOWLEDGEMENTS

CHAPTER 1

Introduction

CHAPTER 2

Elementary Number Theory and Number Theoretic Transforms

2.1 Introduction

2.2 Discrete Fourier Transform and the Convolution

2.3 Congruence

2.4 Chinese Remainder Theorem (CRT)

2.5 Groups, Rings, and Fields

2.6 Number Theoretic Transforms

2.6.1 Mersenne Number Transforms

2.6.2 Fermat Number Transforms

CHAPTER 3

Multiplication Techniques For Microprocessors

3.1 Introduction

3.2 Clocked Multiplication Algorithms

3.2.1 Multiplication on a Microprocessor

3.2.2 Burk-Goldstine - Von-Neumann Method

3.2.3 Robertson's First Method

3.2.4 Robertson's Second Method

3.2.5 Booth's Algorithm

3.2.6 A Short Cut Multiplication Method

3.2.7 Multiple Digit Multiplication Method
3.3 Clockless Multiplication
   3.3.1 Array or Parallel Multiplication

3.4 Read Only Memory (ROM) Multiplier
   3.4.1 Direct ROM Multiplier
   3.4.2 Quarter-Squares Lookup Table Multiplication
   3.4.3 Multiplication Using Logarithms

3.5 Parallel Multiplier Chips

3.6 Modular Arithmetic on Microprocessor
   3.6.1 Addition Modulo 65521
   3.6.2 Subtraction Modulo 65521
   3.6.3 Multiplication Modulo 65521

CHAPTER 4

Implementation of the Winograd Fourier Transform Algorithm

4.1 Introduction

4.2 Computation of NTT using WFTA
   4.2.1 Determination of the Constants for the WFTA

4.3 Architecture of the TMS9900 Microprocessor

4.4 Implementation on the Microprocessor

CHAPTER 5

External Hardware Modular Multiplier

5.1 Introduction

5.2 Design and Implementation of an External Hardware Modular Multiplier
   5.2.1 Interfacing Considerations
   5.2.2 Interfacing the Modular Multiplier with the TMS9900 Microprocessor
CHAPTER 6

Multi Processor and Parallel Processor Systems

6.1 Introduction

6.2 System Organisation
   6.2.1 Single Instruction Single Data (SISD) Machine
   6.2.2 Single Instruction Multiple Data (SIMD) Machine
   6.2.3 Multiple Instruction Multiple Data (MIMD) Machine
   6.2.4 Multiple Instruction Single Data (MISD) Machine

6.3 Multi Processor Systems
   6.3.1 Directly Coupled Multi Processor Systems
   6.3.2 Indirectly Coupled Multi Processor Systems

6.4 Inter Processor Communication
   6.4.1 Time-Shared Bus
   6.4.2 Dedicated Link

6.5 Parallel Processor Systems

6.6 Array Processors

6.7 Processor - Memory Interconnection

6.8 Computer Systems
   6.8.1 Ring Structure
   6.8.2 Star Link
   6.8.3 Fully Connected Link

CHAPTER 7

A Dedicated Parallel Microprocessor System

7.1 Introduction

7.2 Choice of a Microprocessor
7.3 Architecture of the MC6809 Microprocessor
   7.3.1 Hardware and Software Interrupts
   7.3.2 Microprocessor Synchronisation
7.4 Inter Microprocessor Communication
7.5 Dual Microprocessor System
   7.5.1 Merits and Demerits
7.6 Design and Implementation of the Dedicated Parallel Microprocessor System
   7.6.1 System Architecture
   7.6.2 Design of the Control Microprocessor
   7.6.3 Software of the Control Microprocessor
   7.6.4 Design of a Typical Slave Microprocessor
   7.6.5 Software of the Slave Microprocessors
   7.6.6 Synchronisation of the Hardware and the Software
7.7 Transforms of Real Time Signals
7.8 Results

CHAPTER 8

Conclusion

Appendix-A

Modular Arithmetic Routines for the following microprocessors.

TMS9900, MC6809, Z80, 6502

32/16-bit Divide Routine for the MC6809

Appendix-B

Assembler Source Listing for a 15-point WFTA (TMS9900)

FORTRAN Source Listing for a 15-point WFTA
Appendix-C

FORTH Source Listing for a 60-point WFTA

Appendix-D

Software of the Parallel Microprocessor System

Assembler Source listing for 15-point WFTA (MC6809)

Appendix-E

Backplane Wiring for the Parallel Microprocessor System

REFERENCES
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 $\mathbb{Z}_M$, 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.

\[ O(M) = g.c.d (p_{i} - 1) \]

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

\[ \alpha^{N} \equiv 1 \mod M, \alpha^{r} \not\equiv 1 \mod M, \forall r < N. \]

3- \(N^{-1}\) must exist in the ring \(\mathbb{Z}_{M}\). If M is not prime, then \(N^{-1}\) may or may not exist. \(N \cdot N^{-1} \equiv 1 \mod M.\)
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
The difficulties involved in the uniprocessor 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 microprocessors 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.
ELEMENTARY NUMBER THEORY AND NUMBER THEORETIC TRANSFORMS

CHAPTER 2

2.1 Introduction

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

\[
X(k) = \sum_{n=0}^{N-1} x(n) W^{nk}
\]  

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

\[
x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W^{-nk}
\]

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

\( W \) is the principal root of unity such that \( W^N \equiv 1 \mod N \), 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:

\[
y(n) = \sum_{i=0}^{N-1} x(i) h(n-i) \tag{2.3}
\]

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:

\[ T(y) = T(h) \cdot T(x) \quad (2.4) \]

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

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

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:

\[ y(n) = N^{-1} \sum_{k=0}^{N-1} H(k) X(k) W^{-nk} \quad (2.6) \]

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

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

2-3
\[ N-1 \quad N-1 \]
\[ = \sum_{i=0}^{N-1} x(i) \sum_{k=0}^{N-1} H(k) w^{-k(n-i)} \]

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

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

\[ y(n) = \sum_{i=0}^{N-1} x(i) h(n-i \mod N) \quad (2.7) \]

\[ = x(i) * h(i) \]

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 \([0 \text{ to } N-1]\). 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

$$a = qb + r \quad \text{for } 0 < r < q$$

(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:

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

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 $7 \equiv 2 \mod 5$, $12 \equiv 2 \mod 5$, and $17 \equiv 2 \mod 5$. Numbers 7, 12, 17 are congruent mod 5. In general we can write

$$a \equiv r \mod b$$

or

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

also if $a \equiv 0 \mod b$ 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

\[
\begin{align*}
\langle l + m \rangle_b & \equiv \langle l \rangle_b + \langle m \rangle_b \quad b \\
\langle l - m \rangle_b & \equiv \langle l \rangle_b - \langle m \rangle_b \quad b \\
\langle lm \rangle_b & \equiv \langle l \rangle_b \langle m \rangle_b \quad b
\end{align*}
\]

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:

\[
y = a_1 b_1 M/m_1 + a_2 b_2 M/m_2 + \ldots + a_j b_j M/m_j \tag{2.9}
\]

where \( y \equiv a_i \pmod{m_i} \), and composite modulus \( M \) is given by:

\[
M = \prod_{i} m_i \tag{2.10}
\]
provided that $m_i$ are relatively prime, $b_i$ are defined such that:

$$b_i \left( \frac{M}{m_i} \right) \equiv 1 \mod m_i$$

For example, let $x \equiv 2 \mod 3$, $x \equiv 2 \mod 5$, $x \equiv 4 \mod 7$. 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)

$$x = 2 \cdot b_1 \mod \frac{105}{3} + 2 \cdot b_2 \mod \frac{105}{5} + 4 \cdot b_3 \mod \frac{105}{7}$$

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

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

substitution of these values in (2.11) gives

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

### 2.5 Groups, Rings and Fields

Recall from the previous section that

$$a = b + Me \quad (2.12)$$

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

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

In a finite set $[a,b,c,...,M-1]$ 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 : \( a \ast b \cup \forall a, b \in \mathbb{Z}_M \)

2- Associative : \((a \ast b) \ast c = a \ast (b \ast c) \cup \forall a, b, c \in \mathbb{Z}_M \)

3- Identity element : \( a \ast I = I \ast a = a \cup \forall a, I \in \mathbb{Z}_M \)

4- Inverse element : \( a \ast a^{-1} = I \cup \forall a, a^{-1} \in \mathbb{Z}_M \)

5- Commutative : \( a \ast b = b \ast a \cup \forall a, b \in \mathbb{Z}_M \)

Where I represents an identity element and \( a^{-1} \) is the inverse of \( a \). If the operation \( \ast \) 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 \( \mathbb{Z}_M \) is called a group under the operation \( \ast \). 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 \( \mathbb{Z}_M \) under ordinary addition ' + ' and ordinary multiplication ' \cdot ' operations if the following distributive laws hold,

\[
\begin{align*}
a \cdot (b + c) &= a \cdot b + a \cdot c \\
(a + b) \cdot c &= a \cdot c + b \cdot c
\end{align*}
\]

\( \forall a, b, c \in \mathbb{Z}_M \), then the group is called a ring.

Consider some examples of arithmetic mod 11, the elements in the ring \( \mathbb{Z}_M \) are \( \{0, 1, 2, \ldots, 10\} \).

1- Addition : \( 5 + 8 = 13 \equiv 2 \mod 11 \)

2- Negation : \(-3 = 11 + (-3) \equiv 8 \mod 11 \)
3- Subtraction : 3 - 7 = 3 + (11 - 7) = 3 + 4 ≡ 7 mod 11

4- Multiplication : 5 \cdot 4 = 20 ≡ 9 mod 11

5- Multiplicative inverse : 6 \cdot 2 = 12 ≡ 1 mod 11

6 and 2 are multiplicative inverses of each other
or 6^{-1} ≡ 2 mod 11 or 2^{-1} ≡ 6 mod 11

6- Division : a/b is defined if and only if b^{-1} exists,
therefore, a/b ≡ a \cdot b^{-1} \ mod \ M
consider 9/2 = 9 \cdot 6 = 54 ≡ 10 mod 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 ≡ -1 mod 11
2^{10} ≡ 1 mod 11

2.6 Number Theoretic Transforms

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

\[ X(k) = T \cdot x(n), \text{ so } x(n) = T^{-1} \cdot X(k) \]

\[ X(k) = \sum_{n=0}^{N-1} x(n) \alpha^{nk} \] \hspace{1cm} \text{(2.13)}

where \( k = 0,1,2,...,N-1 \).

The inverse is given by:

\[ x(n) = N^{-1} \sum_{k=0}^{N-1} X(k) \alpha^{-nk} \] \hspace{1cm} \text{(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
\[ \alpha^N \equiv 1 \mod M, \quad \alpha, N \in \left[0, M-1\right]. \] 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 \( \mathbb{Z}_M \) which are relatively prime to a given modulus \( M \). This function is represented by \( \varphi(M) \). If \( M \) is composite then \( \varphi(M) < M \), but if \( M \) is prime then the Euler's function \( \varphi(M) = M-1 \), for example \( \varphi(6) = 2 \), and \( \varphi(7) = 6 \).

\[ \varphi(M) = M(1-1/p_1)(1-1/p_2)...(1-1/pr) \]
where \( p_1, p_2, ..., p_r \) are different primes dividing \( M \).

Euler's theorem states that for any non zero element \( a \) in the ring \( \mathbb{Z}_M \), which is relatively prime to \( M \), \( (a, M) = 1 \), the following congruence holds

\[ a^{\varphi(M)} \equiv 1 \mod M \]

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

\[ a^{M-1} \equiv 1 \mod M \]

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:

\[ O(M) = \text{g.c.d} \left( (p_1 - 1)(p_2 - 1)...(p_r - 1) \right) \quad \text{(2.14)} \]

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

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 \( \mod M \).
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 \( \mathbb{Z}_M \). If the modulus \( M \) is a composite number then the multiplicative inverses of all the elements do not exist. Hence \( \mathbb{Z}_M \) is a field if and only if \( M \) is prime. If \( \alpha \) is of the order of \( \varphi(M) \), (where \( \varphi(M) \) is the Euler's totient function), then \( \alpha \) is called the primitive root or the generating function, the non-zero elements of \( \mathbb{Z}_M \) 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 \( \mathbb{Z}_M \), 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.

\[
X(k) = \sum_{n=0}^{N-1} x(n) \cdot 2^{nk} \mod M_p \quad (2.15)
\]

where \( k = 0,1,2,\ldots,p-1 \)

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:

\[
x(n) = q \sum_{k=0}^{N-1} X(k) 2^{-nk} \mod M_p \tag{2.16}
\]

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 \( (k = pq \text{ a composite number}) \). 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:

\[ M = F_t = 2^b + 1 \]  \hspace{1cm} (2.17)

where \( b = 2^t \), \( t = 0,1,2,... \)

Fermat numbers \( F_0 - F_4 \) are prime and \( F_5 \) upwards are composite. Then for FNT to exist

\[ N \mid O(F_t) \]
\[ O(F_t) = 2^b = N_{\text{max}} \]

The largest possible transform length in this case is

\[ N = 2^m \quad m < b \]

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

\[ X(k) = \sum_{n=0}^{N-1} x(n) \alpha^{nk} \mod F_t \]  \hspace{1cm} (2.18)

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

\[ x(n) = N^{-1} \sum_{k=0}^{N-1} X(k) \alpha^{-nk} \mod F_t \]  \hspace{1cm} (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 \( \alpha^2 \equiv 2 \mod F_t \), \( \alpha \equiv \sqrt{2} \), the transform length \( N = 4 \times \text{wordlength} \). For example arithmetic mod \( F_2 \) provides us with \( 6^2 \equiv 2 \mod 17 \), \( 6 \equiv \sqrt{2} \mod 17 \).

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 $\mathbb{Z}_M$. 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.
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

\[
X = -x^o + X^* \\
Y = -y^o + Y^*
\]

(3.1)

- \( x^o \) and \( y^o \) 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

\[
X^* Y^* = (X + x^o)(Y + y^o)
= XY + x^oY + y^oX + x^o y^o
\]
To obtain the correct answer \(-(x^0Y + y^0X + x^0y^0)\) must be added to the final product, such that
\[X^*Y^* = XY\]

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.

\[ X = -x_n2^n + x_{n-1}2^{n-1} + \cdots + x_02^0 \]

\[ Y = -y_n2^n + y_{n-1}2^{n-1} + \cdots + y_02^0 \]

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.

\[
XY = \frac{(X + Y)^2 - (X - Y)^2}{4} \tag{3.1}
\]

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

\[
XY = \frac{(X + Y)^2 - (X - Y)^2}{4} \tag{3.3}
\]

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:

\[ 2m(2n+1) = \left(\frac{2m + 2n + 1}{4}\right)^2 - \left(\frac{2m - 2n - 1}{4}\right)^2 \]
\[ = (m+n)^2 + (m+n) + \frac{1}{4} - (m-n)^2 - (m-n) - \frac{1}{4} \]
\[ = (m+n)^2 + (m+n) - (m-n)^2 - (m-n) \]
\[ = 4mn + 2m \quad \text{(3.4)} \]

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

\[ \left(\frac{2m+2n+1}{2}\right)^2 - \left(\frac{2m-2n-1}{2}\right)^2 = (m+n)^2 - (m-n)^2 \]
\[ = 4mn \]
\[ \neq XY \quad \text{(3.5)} \]

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 $n \times n$-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.

<table>
<thead>
<tr>
<th>Clock MHz</th>
<th>Microprocessor (No. of bits)</th>
<th>Number of Program Bytes</th>
<th>Number of Instr Executed</th>
<th>Clock Cycles (Time µsec)</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>TMS9900 (16) Texas Instr Ltd</td>
<td>36</td>
<td>8</td>
<td>88 (29.3)</td>
<td>50.0</td>
</tr>
<tr>
<td>2</td>
<td>M6502 (8) MOS Technology</td>
<td>42</td>
<td>22</td>
<td>74 (37.0)</td>
<td>13.0</td>
</tr>
<tr>
<td>1</td>
<td>M6809 (8) Motorola</td>
<td>20</td>
<td>8</td>
<td>40 (40.0)</td>
<td>13.0</td>
</tr>
<tr>
<td>8</td>
<td>8X300 (8) Signetics</td>
<td>76</td>
<td>26</td>
<td>52 (6.50)</td>
<td>36.0</td>
</tr>
<tr>
<td>4</td>
<td>8Z80 (8) Zilog</td>
<td>36</td>
<td>14</td>
<td>75 (19.74)</td>
<td>11.0</td>
</tr>
<tr>
<td>.125</td>
<td>COP402 (4) National Semiconductors</td>
<td>205</td>
<td>108</td>
<td>216 (864.0)</td>
<td>4.80</td>
</tr>
</tbody>
</table>
Table 3.2: Results of benchmark programs for modular subtraction.

<table>
<thead>
<tr>
<th>Clock MHz</th>
<th>Microprocessor (No. of bits)</th>
<th>Number of Program Bytes</th>
<th>Number of Instr. Executed</th>
<th>Clock Cycles (Time μsec)</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>TMS9900 (16) Texas Instr. Ltd</td>
<td>24</td>
<td>8</td>
<td>88 (29.3)</td>
<td>50.0</td>
</tr>
<tr>
<td>2</td>
<td>M6502 (8) MOS Technology</td>
<td>75</td>
<td>16</td>
<td>59 (29.5)</td>
<td>13.0</td>
</tr>
<tr>
<td>1</td>
<td>M6809 (8) Motorola</td>
<td>14</td>
<td>6</td>
<td>32 (32.0)</td>
<td>13.0</td>
</tr>
<tr>
<td>8</td>
<td>8X300 (8) Signetics</td>
<td>109</td>
<td>50</td>
<td>100 (12.5)</td>
<td>36.0</td>
</tr>
<tr>
<td>4</td>
<td>Z80 (8) Zilog</td>
<td>49</td>
<td>22</td>
<td>117 (29.24)</td>
<td>11.0</td>
</tr>
<tr>
<td>0.125</td>
<td>COP402 (4) National Semiconductors</td>
<td>211</td>
<td>134</td>
<td>268 (1072.0)</td>
<td>4.80</td>
</tr>
</tbody>
</table>
### Table 3.3: Results of benchmark programs for modular multiplication.

<table>
<thead>
<tr>
<th>Clock MHz</th>
<th>Microprocessor (No. of bits)</th>
<th>Number of Program Bytes</th>
<th>Number of Instr. Executed</th>
<th>Clock Cycles (Time Usec)</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>TMS9900 (16) Texas Instr. Ltd</td>
<td>18</td>
<td>5</td>
<td>242 (80.0)</td>
<td>50.0</td>
</tr>
<tr>
<td>2</td>
<td>M6502 (8) MOS Technology</td>
<td>333</td>
<td>1246</td>
<td>4866 (2433.0)</td>
<td>13.0</td>
</tr>
<tr>
<td>1</td>
<td>M6809 (8) Motorola</td>
<td>128</td>
<td>60</td>
<td>336 (336.0)</td>
<td>13.0</td>
</tr>
<tr>
<td>8</td>
<td>8X300 (8) Signetics</td>
<td>160</td>
<td>325</td>
<td>550 (81.25)</td>
<td>36.0</td>
</tr>
<tr>
<td>4</td>
<td>Z80 (8) Zilog</td>
<td>252</td>
<td>109</td>
<td>2462 (615.5)</td>
<td>11.0</td>
</tr>
<tr>
<td>.125</td>
<td>COP402 (4) National Semiconductors</td>
<td>859</td>
<td>2269</td>
<td>4553 (18212.0)</td>
<td>4.80</td>
</tr>
</tbody>
</table>
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 $103 \equiv 3 \mod 10$. 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) $7 \times 8 = 56 \equiv 11 \mod 15$

in binary form it is given as

$$0111 \times 1000 = 0011 1000$$

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

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

$$1110 \times 1110 = 1100 0100$$

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

If the modulus is chosen as $2^k + 1$ then $2^k = -1$ and $2^{nk} = (-1)^n$. 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 ($\text{FFF}1$) 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) \[
\begin{array}{c}
0279 \\
+ 041C \\
\hline \\
0695 \mod \text{FFF}1 \\
\end{array}
\]

carry = 0

ii) \[
\begin{array}{c}
\text{FFE}F \\
+ \text{00}14 \\
\hline \\
\text{00}03 \mod \text{FFF}1 \\
\text{00}0F \\
\hline \\
\text{00}12 \mod \text{FFF}1 \\
\end{array}
\]

carry = 0
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) 

\[
\begin{array}{c}
0352 \\
- 0140 \\
\hline
0212 \\
\end{array}
\quad \text{mod FFF1}
\]

ii) 

\[
\begin{array}{c}
0140 \\
- 0352 \\
\hline
FDEE \\
+ FFF1 \\
\hline
FDDF \\
\end{array}
\quad \text{mod FFF1}
\]

3.6.3 Multiplication Modulo 65521

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

\[
\begin{align*}
0003 \times 0003 & \equiv 0009 \quad \text{mod FFF1} \\
FFFF \times FFF0 & \equiv 0001 \quad \text{mod FFF1} \\
\quad & \quad (FFFF \equiv -1 \mod \text{FFF1})
\end{align*}
\]
Implementation of the Winograd Fourier Transform Algorithm

4.1 Introduction

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

\[
X(k) = \sum_{n=0}^{N-1} x(n) W^{nk} \tag{4.1}
\]

and the inverse is given by:

\[
x(n) = N^{-1} \sum_{k=0}^{N-1} X(k) W^{-nk} \tag{4.2}
\]

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.
<table>
<thead>
<tr>
<th>Short-length WFTA</th>
<th>No. of Adds</th>
<th>No. of Multiples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>36</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>44</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>74</td>
<td>18</td>
</tr>
</tbody>
</table>

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 \( x_0, x_1, \ldots \) denote the input sequence and \( X_0, X_1, \ldots \) denote the transformed sequence.
3-Point WFTA

\[ N = 3, \ U = 2 \pi/3 \]
\[ t_1 = x_1 + x_2 \]
\[ m_0 = 1 \cdot (x_0 + t_1) \]
\[ m_1 = (\cos U - 1) \cdot t_1 \]
\[ m_2 = j \sin U (x_1 - x_2) \]
\[ s_1 = m_0 + m_1 \]
\[ X_0 = m_0 \]
\[ X_1 = s_1 + m_2 \]
\[ X_2 = s_1 - m_2 \]

5-Point WFTA

\[ N = 5, \ U = 2 \pi/5 \]
\[ t_1 = x_1 + x_4, \ t_2 = x_2 + x_3, \ t_3 = x_1 - x_4 \]
\[ t_4 = x_3 - x_2, \ t_5 = t_1 + t_2 \]
\[ m_0 = 1 \cdot (x_0 + t_5) \]
\[ m_1 = \frac{1}{2} (\cos U + \cos 2U) \cdot t_5 \]
\[ m_2 = \frac{1}{2} (\cos U - \cos 2U) \cdot (t_1 - t_2) \]
\[ m_3 = j \sin U \cdot (t_3 + t_4) \]
\[ m_4 = j (\sin U - \sin 2U) \cdot t_4 \]
\[ m_5 = j (\sin 2U + \sin U) \cdot t_3 \]
\[ s_1 = m_0 + m_1, \ s_2 = s_1 + m_2, \ s_3 = m_5 - m_3 \]
\[ s_4 = s_1 - m_2, \ s_5 = m_3 + m_4 \]
\[ X_0 = m_0 \]
\[ X_1 = s_2 + s_3 \]
\[ X_2 = s_4 + s_5 \]
\[ X_3 = s_4 - s_5 \]
\[ X_4 = s_2 - s_3 \]
Fig. 41: 3-Point WFTA

Fig. 42: 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:

\[
X(k) = \sum_{n=0}^{N-1} x(n) \alpha^{nk} \tag{4.3}
\]

and the inverse is given by:

\[
x(n) = N^{-1} \sum_{k=0}^{N-1} X(k) \alpha^{-nk} \tag{4.4}
\]

where \(\alpha = e^{-j2\pi/N}\), and is an integer root of unity, such that
\(\alpha^N \equiv 1 \mod M\), where \(M\) is the modulus, and \(\alpha\) is defined in a
finite ring of integers \(\mathbb{Z}_M^*\). The choice of modulus is made such
that \(N \parallel M\), if \(M\) is prime then \(N \parallel M-1\). The inverse
\(N^{-1}\) is defined such that \(NN^{-1} \equiv 1 \mod M\). 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 $N \mid \varphi(M)$, where $\varphi(M)$ is the g.c.d of $(p_i-1)$. $\varphi(65521) = 13 \times 5040$ and so this modulus will support any Winograd transform algorithm (5), (9).

2- Choice of transform length $N = 15$.

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

4- Determination of element of order $N$,
   
   $\alpha^{15} \equiv 1 \mod 65521$, $(7791)^{15} \equiv 1 \mod 65521$.

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

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

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

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.

\[(r_2 q_1 + r_1 q_2) \mod N\]

Let
\[
\begin{align*}
  r_1 &= 3, & r_2 &= 5, \\
  q_1 &= 0, 1, 2, & q_2 &= 0, 1, 2, 3, 4
\end{align*}
\]

We get

\[(5q_1 + 3q_2) \mod 15 \quad (4.5)\]

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

\[
\begin{array}{cccccc}
  0 & 3 & 6 & 9 & 12 \\
  5 & 8 & 11 & 14 & 2 \\
  10 & 13 & 1 & 4 & 7
\end{array}
\]

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

\[
\begin{align*}
  5x &\equiv 1 \mod 3 \implies x = 2 \\
  3y &\equiv 1 \mod 5 \implies y = 2 \quad (4.6)
\end{align*}
\]

Equation (4.5) is rewritten as

\[(5xq_1 + 3yq_2) \mod 15 \]

substituting values of \( x \) and \( y \), we get

\[(10q_1 + 6q_2) \mod 15 \quad (4.7)\]
where \( q_1 = 0,1,2 \) and \( q_2 = 0,1,2,3,4. \)

This relationship gives us the output reordering vectors as

\[
\begin{align*}
0 & \quad 6 & \quad 12 & \quad 3 & \quad 9 \\
10 & \quad 1 & \quad 7 & \quad 13 & \quad 4 \\
5 & \quad 11 & \quad 2 & \quad 8 & \quad 14
\end{align*}
\]

9- Determination of the transform coefficients.

By definition

\[
\begin{align*}
\cos U &= \frac{1}{2}(e^{jU} + e^{-jU}) \\
\sin U &= \frac{1}{j}(e^{jU} - e^{-jU})
\end{align*}
\]

where \( U = \frac{2\pi}{N} \)

(4.8) \hspace{3cm} (4.9)

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

\[
\begin{align*}
\cos U &= 2^{-1}(U + U^{-1}) \\
\sin U &= 2^{-1}(-j)(U - U^{-1})
\end{align*}
\]

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 \( U = \alpha^5 \mod 65521 \)

\[
\begin{align*}
(7791)^5 & \equiv 48847 \mod 65521 \\
(48847)^{-1} & \equiv 16673 \mod 65521 \\
m_0 &= 1 \\
m_1 &= \cos U - 1 = 2^{-1}(U + U^{-1}) - 1 \\
&= 32761(48847 + 16673) - 1
\end{align*}
\]
\[ 32760 \mod 65521 \]

\[ m_2 = \text{SINU} - 1 = 32761.41224.24297(48847 - 16673) \]
\[ = 16087 \mod 65521 \]

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

(b) Coefficients for the 5-point transform

Let \( U = \alpha^3 \mod 65521 \)

\[ (7791)^3 \equiv 30887 \mod 65521 \]

\[ (30887)^{-1} \equiv 20625 \mod 65521 \]

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

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

\[ \text{COS2U} = \text{COS}^2 U - \text{SIN}^2 U \equiv 3004 \mod 65521 \]

\[ \text{SIN2U} = 2 \cdot \text{SINU}, \text{COSU} \equiv 49289 \mod 65521 \]

\[ m_0 = 1 \]

\[ m_1 = 2^{-1} \cdot (\text{COSU} + \text{COS2U}) - 1 \equiv 16379 \mod 65521 \]

\[ m_2 = 2^{-1} \cdot (\text{COSU} - \text{COS2U}) \equiv 13376 \mod 65521 \]

\[ m_3 = j(\text{SINU} + \text{SIN2U}) \equiv 19136 \mod 65521 \]

\[ m_4 = j(\text{SIN2U}) \equiv 18005 \mod 65521 \]

\[ m_5 = j(\text{SINU} - \text{SIN2U}) \equiv 48647 \mod 65521 \]

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 \(15^{-1} \equiv 61153 \mod 65521\).

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 anywhere 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.
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 \(2^{16} - 65521 \approx 15\) 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 \(2^{16} - 65521 \approx 15\) 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 (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.
<table>
<thead>
<tr>
<th>Clock cycles</th>
<th>Labels</th>
<th>Mnemonics</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td></td>
<td>MOV</td>
<td>@MPD,R2</td>
</tr>
<tr>
<td>88</td>
<td></td>
<td>MPY</td>
<td>@MPR,R2</td>
</tr>
<tr>
<td>132</td>
<td></td>
<td>DIV</td>
<td>@MOD,R2</td>
</tr>
<tr>
<td>----</td>
<td>MOD</td>
<td>DATA</td>
<td>65521</td>
</tr>
<tr>
<td>234</td>
<td>MPR</td>
<td>DATA</td>
<td>.....</td>
</tr>
<tr>
<td></td>
<td>MPD</td>
<td>DATA</td>
<td>.....</td>
</tr>
</tbody>
</table>

R3 contains the modular product.

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

<table>
<thead>
<tr>
<th>Clock cycles</th>
<th>Labels</th>
<th>Mnemonics</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INPUT1</td>
<td>EQU</td>
<td>&gt;3FF2</td>
</tr>
<tr>
<td></td>
<td>INPUT2</td>
<td>EQU</td>
<td>&gt;3FF4</td>
</tr>
<tr>
<td></td>
<td>OUTPUT</td>
<td>EQU</td>
<td>&gt;3FF6</td>
</tr>
<tr>
<td>20</td>
<td>MOV</td>
<td>@MPR,@INPUT1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>MOV</td>
<td>@MPD,@INPUT2</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>MOV</td>
<td>@OUTPUT,R3</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>MPR</td>
<td>DATA</td>
<td>.....</td>
</tr>
<tr>
<td>54</td>
<td>MPD</td>
<td>DATA</td>
<td>.....</td>
</tr>
</tbody>
</table>

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 $FFFFFF, 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 ROM1

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>390</td>
<td>780</td>
<td>1170</td>
<td>1560</td>
<td>1950</td>
<td>2340</td>
<td>2730</td>
<td>3120</td>
</tr>
<tr>
<td>15</td>
<td>405</td>
<td>795</td>
<td>1185</td>
<td>1575</td>
<td>1965</td>
<td>2355</td>
<td>2745</td>
<td>3135</td>
</tr>
<tr>
<td>30</td>
<td>420</td>
<td>810</td>
<td>1200</td>
<td>1590</td>
<td>1980</td>
<td>2370</td>
<td>2760</td>
<td>3150</td>
</tr>
<tr>
<td>45</td>
<td>435</td>
<td>825</td>
<td>1215</td>
<td>1605</td>
<td>1995</td>
<td>2385</td>
<td>2775</td>
<td>3165</td>
</tr>
<tr>
<td>60</td>
<td>450</td>
<td>840</td>
<td>1230</td>
<td>1620</td>
<td>2010</td>
<td>2400</td>
<td>2790</td>
<td>3180</td>
</tr>
<tr>
<td>75</td>
<td>465</td>
<td>855</td>
<td>1245</td>
<td>1635</td>
<td>2025</td>
<td>2415</td>
<td>2805</td>
<td>3195</td>
</tr>
<tr>
<td>90</td>
<td>480</td>
<td>870</td>
<td>1260</td>
<td>1650</td>
<td>2040</td>
<td>2430</td>
<td>2820</td>
<td>3210</td>
</tr>
<tr>
<td>105</td>
<td>495</td>
<td>885</td>
<td>1275</td>
<td>1665</td>
<td>2055</td>
<td>2445</td>
<td>2835</td>
<td>3225</td>
</tr>
<tr>
<td>120</td>
<td>510</td>
<td>900</td>
<td>1290</td>
<td>1680</td>
<td>2070</td>
<td>2460</td>
<td>2850</td>
<td>3240</td>
</tr>
<tr>
<td>135</td>
<td>525</td>
<td>915</td>
<td>1305</td>
<td>1695</td>
<td>2085</td>
<td>2475</td>
<td>2865</td>
<td>3255</td>
</tr>
<tr>
<td>150</td>
<td>540</td>
<td>930</td>
<td>1320</td>
<td>1710</td>
<td>2100</td>
<td>2490</td>
<td>2880</td>
<td>3270</td>
</tr>
<tr>
<td>165</td>
<td>555</td>
<td>945</td>
<td>1335</td>
<td>1725</td>
<td>2115</td>
<td>2505</td>
<td>2895</td>
<td>3285</td>
</tr>
<tr>
<td>180</td>
<td>570</td>
<td>960</td>
<td>1350</td>
<td>1740</td>
<td>2130</td>
<td>2520</td>
<td>2910</td>
<td>3300</td>
</tr>
<tr>
<td>195</td>
<td>585</td>
<td>975</td>
<td>1365</td>
<td>1755</td>
<td>2145</td>
<td>2535</td>
<td>2925</td>
<td>3315</td>
</tr>
<tr>
<td>210</td>
<td>600</td>
<td>990</td>
<td>1380</td>
<td>1770</td>
<td>2160</td>
<td>2550</td>
<td>2940</td>
<td>3330</td>
</tr>
<tr>
<td>225</td>
<td>615</td>
<td>1005</td>
<td>1395</td>
<td>1785</td>
<td>2175</td>
<td>2565</td>
<td>2957</td>
<td>3345</td>
</tr>
<tr>
<td>240</td>
<td>630</td>
<td>1020</td>
<td>1410</td>
<td>1800</td>
<td>2190</td>
<td>2580</td>
<td>2970</td>
<td>3360</td>
</tr>
<tr>
<td>255</td>
<td>645</td>
<td>1035</td>
<td>1425</td>
<td>1815</td>
<td>2205</td>
<td>2595</td>
<td>2985</td>
<td>3375</td>
</tr>
<tr>
<td>270</td>
<td>660</td>
<td>1050</td>
<td>1440</td>
<td>1830</td>
<td>2220</td>
<td>2610</td>
<td>3000</td>
<td>3390</td>
</tr>
<tr>
<td>285</td>
<td>675</td>
<td>1065</td>
<td>1455</td>
<td>1845</td>
<td>2235</td>
<td>2625</td>
<td>3015</td>
<td>3405</td>
</tr>
<tr>
<td>300</td>
<td>690</td>
<td>1080</td>
<td>1470</td>
<td>1860</td>
<td>2250</td>
<td>2640</td>
<td>3030</td>
<td>3420</td>
</tr>
<tr>
<td>315</td>
<td>705</td>
<td>1095</td>
<td>1485</td>
<td>1875</td>
<td>2265</td>
<td>2655</td>
<td>3045</td>
<td>3435</td>
</tr>
<tr>
<td>330</td>
<td>720</td>
<td>1110</td>
<td>1500</td>
<td>1890</td>
<td>2280</td>
<td>2670</td>
<td>3060</td>
<td>3450</td>
</tr>
<tr>
<td>345</td>
<td>735</td>
<td>1125</td>
<td>1515</td>
<td>1905</td>
<td>2295</td>
<td>2685</td>
<td>3075</td>
<td>3465</td>
</tr>
<tr>
<td>360</td>
<td>750</td>
<td>1140</td>
<td>1530</td>
<td>1920</td>
<td>2310</td>
<td>2700</td>
<td>3090</td>
<td>3480</td>
</tr>
<tr>
<td>375</td>
<td>765</td>
<td>1155</td>
<td>1545</td>
<td>1935</td>
<td>2325</td>
<td>2715</td>
<td>3105</td>
<td>3495</td>
</tr>
</tbody>
</table>
Table 5.2: Values in ROM2

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>34319</td>
<td>3117</td>
<td>37743</td>
<td>6234</td>
<td>40553</td>
<td>9351</td>
<td>43670</td>
<td>12468</td>
<td>46787</td>
<td>3840</td>
<td>6957</td>
<td>41276</td>
<td>10074</td>
<td>44393</td>
<td>13191</td>
<td>47510</td>
</tr>
<tr>
<td>1</td>
<td>41999</td>
<td>10797</td>
<td>45116</td>
<td>13914</td>
<td>48233</td>
<td>17031</td>
<td>51350</td>
<td>20148</td>
<td>54467</td>
<td>7680</td>
<td>11520</td>
<td>48956</td>
<td>17754</td>
<td>52073</td>
<td>20871</td>
<td>55190</td>
</tr>
<tr>
<td>2</td>
<td>96376</td>
<td>18477</td>
<td>52796</td>
<td>21594</td>
<td>55913</td>
<td>24711</td>
<td>59030</td>
<td>27828</td>
<td>62147</td>
<td>15360</td>
<td>49679</td>
<td>26157</td>
<td>60476</td>
<td>29274</td>
<td>63593</td>
<td>32391</td>
</tr>
<tr>
<td>3</td>
<td>53519</td>
<td>22317</td>
<td>56636</td>
<td>25434</td>
<td>59753</td>
<td>28551</td>
<td>62870</td>
<td>31668</td>
<td>65467</td>
<td>19200</td>
<td>53519</td>
<td>29997</td>
<td>64316</td>
<td>33114</td>
<td>1912</td>
<td>36231</td>
</tr>
<tr>
<td>4</td>
<td>38491</td>
<td>41517</td>
<td>10315</td>
<td>44634</td>
<td>13432</td>
<td>47751</td>
<td>16549</td>
<td>50866</td>
<td>19666</td>
<td>42240</td>
<td>11038</td>
<td>45357</td>
<td>14155</td>
<td>48474</td>
<td>17272</td>
<td>51591</td>
</tr>
<tr>
<td>5</td>
<td>14878</td>
<td>49197</td>
<td>17995</td>
<td>52314</td>
<td>21112</td>
<td>55431</td>
<td>24229</td>
<td>58548</td>
<td>27346</td>
<td>46080</td>
<td>14878</td>
<td>49197</td>
<td>17995</td>
<td>52314</td>
<td>21112</td>
<td>55431</td>
</tr>
<tr>
<td>6</td>
<td>18718</td>
<td>53037</td>
<td>21835</td>
<td>56154</td>
<td>24952</td>
<td>59271</td>
<td>28069</td>
<td>62388</td>
<td>31186</td>
<td>49920</td>
<td>18718</td>
<td>53037</td>
<td>21835</td>
<td>56154</td>
<td>24952</td>
<td>59271</td>
</tr>
<tr>
<td>7</td>
<td>22558</td>
<td>56877</td>
<td>25675</td>
<td>59994</td>
<td>28792</td>
<td>63111</td>
<td>31909</td>
<td>707</td>
<td>35026</td>
<td>57600</td>
<td>22558</td>
<td>56877</td>
<td>25675</td>
<td>59994</td>
<td>28792</td>
<td>63111</td>
</tr>
<tr>
<td>8</td>
<td>26398</td>
<td>60717</td>
<td>29515</td>
<td>63834</td>
<td>32632</td>
<td>1430</td>
<td>35749</td>
<td>4547</td>
<td>38866</td>
<td>61440</td>
<td>30238</td>
<td>64557</td>
<td>33355</td>
<td>2153</td>
<td>36472</td>
<td>5270</td>
</tr>
<tr>
<td>9</td>
<td>34078</td>
<td>2876</td>
<td>37195</td>
<td>5993</td>
<td>40312</td>
<td>9110</td>
<td>43429</td>
<td>12227</td>
<td>46546</td>
<td>65280</td>
<td>34078</td>
<td>2876</td>
<td>37195</td>
<td>5993</td>
<td>40312</td>
<td>9110</td>
</tr>
<tr>
<td>A</td>
<td>37918</td>
<td>6716</td>
<td>41035</td>
<td>9833</td>
<td>44152</td>
<td>12950</td>
<td>47269</td>
<td>16067</td>
<td>50386</td>
<td>3599</td>
<td>37918</td>
<td>6716</td>
<td>41035</td>
<td>9833</td>
<td>44152</td>
<td>12950</td>
</tr>
<tr>
<td>B</td>
<td>41758</td>
<td>10556</td>
<td>44875</td>
<td>13673</td>
<td>47992</td>
<td>16790</td>
<td>51109</td>
<td>19907</td>
<td>54226</td>
<td>7439</td>
<td>41758</td>
<td>10556</td>
<td>44875</td>
<td>13673</td>
<td>47992</td>
<td>16790</td>
</tr>
<tr>
<td>C</td>
<td>45598</td>
<td>14396</td>
<td>48715</td>
<td>17513</td>
<td>51832</td>
<td>20630</td>
<td>54949</td>
<td>23747</td>
<td>58066</td>
<td>11279</td>
<td>45598</td>
<td>14396</td>
<td>48715</td>
<td>17513</td>
<td>51832</td>
<td>20630</td>
</tr>
<tr>
<td>D</td>
<td>49438</td>
<td>18236</td>
<td>52555</td>
<td>21353</td>
<td>55672</td>
<td>24470</td>
<td>58789</td>
<td>27587</td>
<td>61906</td>
<td>15119</td>
<td>49438</td>
<td>18236</td>
<td>52555</td>
<td>21353</td>
<td>55672</td>
<td>24470</td>
</tr>
<tr>
<td>E</td>
<td>53278</td>
<td>22076</td>
<td>56395</td>
<td>25193</td>
<td>59512</td>
<td>28310</td>
<td>62629</td>
<td>31427</td>
<td>65959</td>
<td>18959</td>
<td>53278</td>
<td>22076</td>
<td>56395</td>
<td>25193</td>
<td>59512</td>
<td>28310</td>
</tr>
<tr>
<td>F</td>
<td>57118</td>
<td>25916</td>
<td>60235</td>
<td>29033</td>
<td>63352</td>
<td>32150</td>
<td>948</td>
<td>35267</td>
<td>6999</td>
<td>22799</td>
<td>57118</td>
<td>25916</td>
<td>60235</td>
<td>29033</td>
<td>63352</td>
<td>32150</td>
</tr>
<tr>
<td></td>
<td>60958</td>
<td>29756</td>
<td>64075</td>
<td>32873</td>
<td>1671</td>
<td>35990</td>
<td>4788</td>
<td>39107</td>
<td>42947</td>
<td>26639</td>
<td>60958</td>
<td>29756</td>
<td>64075</td>
<td>32873</td>
<td>1671</td>
<td>35990</td>
</tr>
<tr>
<td></td>
<td>64798</td>
<td>33596</td>
<td>2394</td>
<td>36713</td>
<td>5511</td>
<td>39830</td>
<td>8628</td>
<td>42947</td>
<td>74279</td>
<td>30479</td>
<td>64798</td>
<td>33596</td>
<td>2394</td>
<td>36713</td>
<td>5511</td>
<td>39830</td>
</tr>
</tbody>
</table>
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 $s_1, s_2, ..., s_m$ (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 $m$ separate input data streams $D_1, D_2, ..., D_m$. To generate the output data streams $D'_1, D'_2, ..., D'_m$. 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 $D_1, D_2, \ldots, D_m$ to generate the output data streams $D'_1, D'_2, \ldots, D'_m$. 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 beginning 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 cut off 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.

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

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.
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 \( N = r_1 r_2 x ... x r_n \) 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: 16-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.

<table>
<thead>
<tr>
<th>MPU STATE</th>
<th>BA</th>
<th>BS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL RUNNING</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INTERRUPT ACK.</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SYNC ACK.</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>HALT OR BUS GRANT</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 occurred.

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 \( BA \cdot BS = 1 \) 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 occurring 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.
Figure 7.10: ACIA interface.
Figure (7.11) shows circuitry associated with the control microprocessor. The control microprocessor has its 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 initialise 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.

<table>
<thead>
<tr>
<th>Proc. No.</th>
<th>No. of data exchange</th>
<th>No. of additions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Receive</td>
<td>Transmit</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>P5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>P6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>P7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>P8</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>P9</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>P10</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>P11</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>P12</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>P13</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>P14</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>P15</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>P16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P17</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P18</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>System</th>
<th>Assembler</th>
<th>FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC6809</td>
<td>10 msec</td>
<td>--</td>
</tr>
<tr>
<td>Parallel Structure</td>
<td>675 usec</td>
<td>--</td>
</tr>
<tr>
<td>TMS9900</td>
<td>4 msec</td>
<td>--</td>
</tr>
<tr>
<td>IBM 370/168</td>
<td>365 usec</td>
<td>2 msec</td>
</tr>
<tr>
<td>IBM 370/4341</td>
<td>1 msec</td>
<td>5 msec</td>
</tr>
</tbody>
</table>

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 overheads 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 MULTIPLICATION *

* * MODULAR ADDITION *

* * MODULAR SUBTRACTION *

START

MODULAR MULTIPLICATION

MOV 2MPR,R1
MOV 2MPR,R1
MOV 2MPD,R2

MPY R1,R2

JOC OVER

CI R2,65521

JL OVR

AI R2,15

OVR MOV R2,@SUM

OVER MOV R1,@RES

MODULAR ADDITION

MOV @AD1,R1
MOV @AD2,R2

A R1,R2

JOC OVER

CI R2,65521

JL OVR

AI R2,15

OVR MOV R2,@SUM

OVER MOV R1,@RES

MODULAR SUBTRACTION

MOV @SUBT1,R1
MOV @SUBT2,R2

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
AORG $30

START

MODULAR MULTIPLICATION

#ADS

OVER

LDX #ADS

ADD ,X++

BBSKIP

CMPD #65521

BLD SKIP1

SKIP ADD #15

SKIP1 STO ,X

** MODULAR ARITHMETIC PROGRAMS FOR MC6809 **

* * MODULAR MULTIPLICATION *

* * MODULAR ADDITION *

* * MODULAR SUBTRACTION *
START:
LD HL, (ADD1)
LD BC, (ADD2)
ADD HL, BC
Appendix-A

OVER1:
LD BC,15
ADD HL,BC
LD (SUM),HL
OVER:
JP SKIP
;
AUD1:
DEFW 0
AUD2:
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 RC,65521
ADD HL,BC
JP OVR
ZLRO:
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:
DEFS 0
SUBT2:
DEFS 0
SUBT3:
DEFS 0
SUBT4:
DEFS 0
RES:
DEFS 0
;
**********************************************************************
;
MODULAR MULTIPLICATION
;
**********************************************************************
SKIP2:
LD A,(MPR1)
LD H,A
LD A,(MPD1)
LD E,A
CALL MULT
LD (PPD3),HL
LD A,(MPR2)
LD H,A
LD A,(MPD2)
LD E,A
CALL MULT
LD (PPD1),HL
;
**********************************************************************

;
Appendix-A

LD A,241
CP L
JP Z,BAK6
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
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

****************************

MULT:
JUMP:
NOAOD:
MPO1:
MPOZ:
MPR1:
MPRZ:
PROO1:
PROO2:
PROO3:
PRODS:
PROD6:

L,D0
LD B,8
JUMP: ADD HL,HL
JR NC,NOADD
ADD HL,DE

****************************

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

****************************

***************MODULAR ARITHMETIC PROGRAMS FOR 6502***************

* MODULAR ARITHMETIC PROGRAMS FOR 6502

NAM M6502
ORG $1024

LDA 5,X
CMP #$F1
BEQ SKIP1
BMI SUBT1
LDA 5,X
SKIP1
ADC #15
STA 5,X
LDA #0
ADC 4,X
STA 4,X
SUBT1
JMP SUBT
ORG $D023
FDB 0
FDB 0
FCB 0
FCB 0
MODULAR SUBTRACTION

ORG $1024
LDX #$SUB
STA CHECK
LDA 0,X
CMP 2,X
BEQ OMIT
BCS JMP
INC CHECK
JMP LDA 1,X
SEC
SBC 3,X
STA 5,X
LDA 0,X
SBC 2,X
STA 4,X
LDA CHECK
REQ MULT1
CLC
LDA 5,X
ADC #$F1
STA 5,X
LDA 4,X
ADC #$FF
STA 4,X
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 $0023
SUB FDB 0
SUB1 FDB 0
SUB2 FCB 0
SUB3 FDB 0
CHECK FCB 0
MULT LDX #$MPL2

MULTIPLICATION ROUTINE
<table>
<thead>
<tr>
<th>ADC</th>
<th>STA</th>
<th>LDA</th>
<th>ADC</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,x</td>
<td>14,x</td>
<td>#0</td>
<td>9,x</td>
<td>13,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STA</th>
<th>LDA</th>
<th>ADC</th>
<th>STA</th>
<th>LDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,y</td>
<td>#15</td>
<td>0,y</td>
<td>#15</td>
<td>4,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>JSR</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>12,x</td>
<td>SUBRT</td>
<td>2,x</td>
<td>2,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

\* \* ROUTINE FOR MODULARISING \* \*

<table>
<thead>
<tr>
<th>LDA</th>
<th>CMP</th>
<th>BNE</th>
<th>LDA</th>
<th>CMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,x</td>
<td>#FF</td>
<td>JMPA</td>
<td>16,x</td>
<td>#FF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>LDA</th>
<th>ADC</th>
<th>STA</th>
<th>LDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,x</td>
<td>#0</td>
<td>19,x</td>
<td>16,x</td>
<td>#0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>ADC</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,x</td>
<td>15,x</td>
<td>15,x</td>
<td>15,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,x</td>
<td>2,x</td>
<td>15,x</td>
<td>15,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>JSR</th>
<th>BNE</th>
<th>JSR</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>#FF</td>
<td>OVER1</td>
<td>SUBRT</td>
<td>STAB</td>
<td>#0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JSR</th>
<th>STA</th>
<th>CMP</th>
<th>JSR</th>
<th>CMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBRT</td>
<td>15,x</td>
<td>#FF</td>
<td>SUBRT</td>
<td>0,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JMPA</th>
<th>CLC</th>
<th>LDA</th>
<th>ADC</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,x</td>
<td>4,x</td>
<td>15,x</td>
<td>15,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>LDA</th>
<th>ADC</th>
<th>LDA</th>
<th>STM</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,x</td>
<td>16,x</td>
<td>15,x</td>
<td>16,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STA</th>
<th>LDA</th>
<th>BEL</th>
<th>STA</th>
<th>LDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,x</td>
<td>10,x</td>
<td>1,5</td>
<td>15,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STA</th>
<th>ADC</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,x</td>
<td>#15</td>
<td>16,x</td>
<td>15,x</td>
<td>16,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>LDA</th>
<th>ADC</th>
<th>CMP</th>
<th>BCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0</td>
<td>#0</td>
<td>#5F</td>
<td>#F1</td>
<td>OVER1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>LDA</th>
<th>ADC</th>
<th>CMP</th>
<th>BCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,x</td>
<td>15,x</td>
<td>#0</td>
<td>#0</td>
<td>OVER1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STA</th>
<th>ADC</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,x</td>
<td>#15</td>
<td>16,x</td>
<td>15,x</td>
<td>16,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STA</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,x</td>
<td>15,x</td>
<td>14,x</td>
<td>14,x</td>
<td>13,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>CMP</th>
<th>BNE</th>
<th>LDA</th>
<th>CMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>#FF</td>
<td>#0</td>
<td>OVRA</td>
<td>#FF</td>
<td>OVRA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>CMP</th>
<th>BNE</th>
<th>LDA</th>
<th>CMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0</td>
<td>#FF</td>
<td>OVRA</td>
<td>#0</td>
<td>OVRA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>JSR</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>14,x</td>
<td>SUBRT</td>
<td>2,x</td>
<td>2,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

\* \* MULTIPLICATION ROUTINE \* \*

<table>
<thead>
<tr>
<th>LDA</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,x</td>
<td>15,x</td>
<td>15,x</td>
<td>15,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>STA</th>
<th>LDA</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>17,x</td>
<td>13,x</td>
<td>19,x</td>
<td>13,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDA</th>
<th>ASL</th>
<th>LDA</th>
<th>ASL</th>
</tr>
</thead>
<tbody>
<tr>
<td>13,x</td>
<td>1,x</td>
<td>13,x</td>
<td>2,x</td>
</tr>
</tbody>
</table>
**Appendix-A**

```
ORG $0023

LDA #1
DPA 1,X
STA 1,Y
CLC
RDP 0,Y
BCC BAK1
MPR FO3 0

LDA 2,X
ADC 4,X
STA 4,X
LDA 1,X
ADC 3,X
STA 3,X
BCC BAK1

DEY
REQ OUT
JMP OVER

OUT
RTS

... 32/16 BIT DIVISION FOR MC6809 MICROPROCESSOR ...

... 32 BIT / 16 BIT UNSIGNED ...

```

```
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 |
DVSP2 FCB 00 |END
Appendix-B

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

FORTRAN program source listing for a 15-point WFTA
Appendix-B

* 15-POINT WINograd ALGORITHM (WFTA) TMS9900 *

** IDT "WINO15" **
** OPTION XREF,SYMT **
** AORG >6000 **

START
LI R4,YREG
LI R5,XREG

* INPUT SHUFFLE *
* 3 POINT PREWEAVE *

** 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,*P5

* 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(P5)

* MOV @14(R4),@128(R5)
* MOV @2(R5),@10(R5)
* MOV @20(R5),@18(R5)
* MOV @20(R4),@20(R5)
* MOV @26(R4),@22(R5)
* MOV @26(R4),@24(R5)
* MOV @26(R4),@32(R5)
* MOV @26(R4),@36(R5)
* MOV @30(R4),@314(R5)
* MOV @30(R4),@316(R5)
* MOV @32(R4),@316(R5)
* MOV @32(R4),@314(R5)
* MOV @32(R4),@312(R5)
* MOV @32(R4),@312(R5)
* MOV @32(R4),@310(R5)
* MOV @32(R4),@310(R5)
* MOV @32(R4),@312(R5)
* MOV @32(R4),@314(R5)
* MOV @32(R4),@316(R5)

** 5 POINT PREWEAVE **

* MOV @30(R5),R0
* MOV @30(R5),R1
* BL ADDSUB
* MOV R2,@30(R5)
* MOV R3,@30(R5)
* MOV @32(R5),R3
* BL ADD
* MOV R3,@4(P5)

* MOV @34(R5),R0
* MOV @34(R5),R1
<table>
<thead>
<tr>
<th>MOV</th>
<th>R2, A4(R5)</th>
<th>MOV</th>
<th>R3, A28(R5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>R3, A26(R5)</td>
<td>MOV</td>
<td>R3, R0(R5), R0</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A10(R5)</td>
<td>MOV</td>
<td>R3, A24(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A16(R6), R3</td>
<td>MOV</td>
<td>R3, A16(R5)</td>
</tr>
<tr>
<td>BL</td>
<td>ADD</td>
<td>MOV</td>
<td>R3, A16(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A214(R6)</td>
<td>MOV</td>
<td>R3, A12(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A20(R6), R0</td>
<td>MOV</td>
<td>A214(R5), R2</td>
</tr>
<tr>
<td>MOV</td>
<td>A22(R6), R3</td>
<td>BL</td>
<td>ADD</td>
</tr>
<tr>
<td>MOV</td>
<td>A14(R6), P0</td>
<td>MOV</td>
<td>A14(R5), R2</td>
</tr>
<tr>
<td>MOV</td>
<td>A16(R6), R1</td>
<td>BL</td>
<td>ADD</td>
</tr>
<tr>
<td>MOV</td>
<td>A22(R6)</td>
<td>MOV</td>
<td>R3, A16(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A316(R6), R1</td>
<td>MOV</td>
<td>R3, A12(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A320(R6), R1</td>
<td>MOV</td>
<td>R3, A14(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A24(R6), P3</td>
<td>MOV</td>
<td>A34(R5), R3</td>
</tr>
<tr>
<td>MOV</td>
<td>A326(R6), R2</td>
<td>BL</td>
<td>ADD</td>
</tr>
<tr>
<td>MOV</td>
<td>A322(R6), R3</td>
<td>MOV</td>
<td>R2, A10(R5), R2</td>
</tr>
<tr>
<td>MOV</td>
<td>A34(R6), R2</td>
<td>MOV</td>
<td>R3, A18(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A32(R6), R3</td>
<td>BL</td>
<td>ADD</td>
</tr>
<tr>
<td>MOV</td>
<td>A324(R6)</td>
<td>MOV</td>
<td>R2, A10(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A32(R6), R0</td>
<td>MOV</td>
<td>R2, A14(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A32(R6), R1</td>
<td>MOV</td>
<td>R3, A20(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>A320(R6), R0</td>
<td>MOV</td>
<td>R3, A22(R5)</td>
</tr>
<tr>
<td>BL</td>
<td>ADD</td>
<td>MOV</td>
<td>R2, A26(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A20(R6)</td>
<td>MOV</td>
<td>R2, A24(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A26(R6), R0</td>
<td>MOV</td>
<td>R3, A24(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A26(R6), R0</td>
<td>MOV</td>
<td>R3, A24(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A26(R6), R1</td>
<td>MOV</td>
<td>R3, A24(R5)</td>
</tr>
<tr>
<td>BL</td>
<td>ADD</td>
<td>MOV</td>
<td>R2, A26(R5)</td>
</tr>
<tr>
<td>MOV</td>
<td>R3, A22(R5)</td>
<td>MOV</td>
<td>R2, A24(R5)</td>
</tr>
</tbody>
</table>
APPENDIX - B

MOV $16(R5), R0
MOV $26(R5), R1
BL $ADD
MOV R2, $16(R5)
MOV R3, $26(R5)

MOV $18(R5), R0
MOV $26(R5), R1
BL $ADD
MOV R2, $18(R5)
MOV R3, $26(R5)

*    *    *    *    *
* * OUTPUT SHUFFLE   *
*    *    *    *    *

MOV *R5, *R6
MOV @12(R5), @2(R6)
MOV @24(R5), @4(R6)
MOV @36(R5), @6(R6)
MOV @18(R5), @8(R6)
MOV @30(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  *
*    *    *    *    *    *

AUDSUB MOV R1, R2
A R0, R2
JOC PLUS
CI R2, 65521
JL SUB
PLUS AI R2, 15
Appendix-B

***************
* 15-POINT WINOGRAD ALGORITHM (WFTA) *
***************

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

INPUT SHUFFLE VECTORS

DATA IRF /0, 3, 6, 9, 12, 5, P, 11, 14,
1 2, 10, 13, 1, 4, 7/

OUTPUT SHUFFLE VECTORS

DATA IRFI /0, 6, 12, 3, 9, 10, 1, 7, 13,
1 4, 5, 11, 2, 8, 14/

FORWARD TRANSFORM COEFFICIENTS

DATA COEF /1.00, 16379.00, 13376.00, 19136.00,
1 18005.00, 49647.00, 32759.00, 8192.00,
2 45457.00, 36817.00, 5753.00, 25311.00,
3 16087.00, 29032.00, 8748.00, 23174.00,
4 43615.00, 1465.00/
DATA COEFR /61153.00, 5460.00, 18364.00, 46773.00,
1 20640.00, 5493.00, 6552.00, 57331.00,
2 37975.00, 29122.00, 34561.00, 24521.00,
3 29504.00, 28641.00, 12521.00, 5913.00,
4 24748.00, 21938.00/

READ INPUT DATA ARRAY

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 = MODG(S1 - S3)
S7 = MODG(S2 + S4)
S8 = MODG(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. 1.00) GO TO 50
DO 40 I = 1, 18
40 Z(I) = MODG(Z(I)*COEF(I))
GO TO 70
50 DO 60 I = 1, 18
60 ZCI) = MODG(Z(I)*COEF(I))
70 J = 1
DO 80 I = 1, 3
IND = 5 * (I - 1)
S9 = MODG(Z(J) + Z(J + 1))
S10 = MODG(S9 + Z(J + 2))
S11 = MODG(S9 - Z(J + 2))
S12 = MODG(Z(J + 3) - Z(J + 4))
S13 = MODG(Z(J + 4) + Z(J + 5))
S14 = MODG(S10 + S12)
S15 = MODG(S10 - S12)
S16 = MODG(S11 + S13)
S17 = MODG(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 = MODG(X(I) + X(5 + I))
T2 = MODG(T + X(10 + I))
X(10 + I) = MODG(T - X(10 + I))
X(5 + I) = T2
90 CONTINUE
DO 100 I = 1, 15
OUT(IPFI(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
DOUBLE PRECISION FUNCTION MODO(F)
REAL*8 F, MOD
MOD = 65521.00
IF (F .LT. 0.000) GO TO 10
MODO = DMOD(F,MOD)
GO TO 20
10 MODO = MOD - DMOD(-F,MOD)
20 RETURN
END
Appendix-C

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

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 TEMPOARY 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 FCODEF 144 ARRAY RCODEF 120 ARRAY X 144 ARRAY Y
120 ARRAY RF 120 ARRAY RFI

INPUT SHUFFLE VECTORS 
RF FILL
0 72 24 96 48 90 42 114 66 13 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 23 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 54 34 102 80 94 3 32 56
110 14 38 62 86 20 44 68 92 116 50 74 93 2 26

S
COEFFICIENTS FOR FORWARD TRANSFORM
FCODEF FILL
1 16379 13376 64390 46385 48647
1 16379 13376 64390 46385 48647
1 16379 13376 64390 46385 48647
41224 13991 53009 26668 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 23796 17202

S
COEFFICIENTS FOR REVERSE TRANSFORM
Appendix-C

RCOEF FILL
64429 1365 4591 9847 4687 50514
64429 1365 4591 9847 4687 50514
64429 1365 4591 9847 4687 50514
3681 11779 30785 35383 3541 64907
1638 30713 25874 17990 25730 55271
1638 30713 25874 17990 25730 55271
1638 30713 25874 17990 25730 55271
27239 15092 52104 12439 13250 44432
27239 15092 52104 12439 13250 44432
27239 15092 52104 12439 13250 44432
27239 15092 52104 12439 13250 44432
50784 2041 30577 40308 40673 45578

:M ( 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 12 MOV 0 9 1 3
MOV 1 4 3 0 MOV 7 PUSH RETURN ; AMULT ALOAD 144 0 DO I FCOEF + @ 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 :

:M ( MODULAR ADDITION ) HEX
CODE MDD 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
:C MODULAR MULTIPLICATION )
CODE 0/ 7 POP 5 POP FFF1 4 LI 5 0 7 MPY
5 0 4 DIV 6 PUSH RETURN
( REG4 CONTAINS DIVISOR )

:C MODULAR SUBTRACTION )
CODE SBT 2 POP 1 POP 0 3 0 1 MOV 0 1 0 2 5 0 2 0 3 0
FLT IF FFF1 1 AI 1 PUSH ELSE 1 PUSH THEN RETURN
( MODULAR HARDWARE MULTIPLIER )
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 DC I 40 + X + 0 I 80 + X + 0 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 + 2 I 20 + Y + @ MOD I X + ! 2 +=LOOP ;
: 42AD 10 0 DO I 10 + Y + 2 I 30 + Y + @ OVER OVER MOD
I 10 + X + ! SBT I 30 + X + ! 2 +=LOOP ;
Appendix-C

C-7

: 425B 10 0 DO I Y + 3 I 20 + Y + 3 SBT I 20 + X + ! 2 +LOOP ;
: 43AD 10 0 DO I X + 3 I 10 + X + 3 MOD TM ! I Y + 3 I 10 + X + 3 SBT I 10 + X + ! 2 +LOOP ;

: 44AD 10 0 DO I 40 + Y + 3 I 60 + Y + 3 OVER OVER MOD I 40 + X + ! SBT I 60 + X + ! 2 +LOOP ;
: 45AD 10 0 DO I 50 + Y + 3 I 70 + Y + 3 OVER OVER MOD I 50 + X + ! SBT I 70 + X + ! 2 +LOOP ;
: 48AD 10 0 DO I 40 + X + 3 I 50 + Y + 3 MOD TM ! I 40 + X + 3 SBT I 50 + X + ! 2 +LOOP ;
: 49AD 10 0 DO I 80 + X + ! SBT I 100 + X + ! 2 +LOOP ;
: 4AAD 10 0 DO I 90 + Y + 3 I 110 + Y + 3 OVER OVER MOD I 90 + X + ! SBT I 110 + X + ! 2 +LOOP ;
: 4B0D 10 0 DO I 80 + X + 3 I 90 + X + 3 MOD TM ! I 80 + X + 3 SBT I 90 + X + ! TM ! I 80 + X + ! 2 +LOOP ;
: I4PT 41AD 42AD 425B 43AD 44AD 45AD 48AD 49AD 4AAD 40AD ;

: S

(MULTIPLICATION WITH COEFFICIENTS)
0 INTEGER FLAG

: FMULT 144 0 DO I FCDEF + 3 I Y + 3 0 N/ Y + ! 2 +LOOP ;
: FMULT 144 0 DO I FCDEF + 3 I Y + 3 0 N/ Y + ! 2 +LOOP ;
: MULT FLAG 3 0 = IF FMULT ELSE RMULT THEN

(5 POINT PRE-WEAVE)

I15PT TM 3 X + 3 TM3 3 X + 3 OVER OVER MOD S1 3 Y + !
SBT 55 3 Y + ! ;
I25PT TM1 3 X + 3 TM2 3 X + 3 MOD S2 3 Y + ! TM2 3 X + 3 TM1 3 X + 3 SBT 54 3 Y + ! ;
I35PT S1 3 Y + 3 S2 3 Y + 3 OVER OVER MOD S1 3 Y + !
SBT 52 3 Y + ! TM0 3 X + 3 S1 3 Y + 3 MOD S0 3 Y + ! ;
I45PT S5 3 Y + 3 S4 3 Y + 3 MOD S3 3 Y + ! ;
I5PT INT2 SINT 24 0 DO I15PT I25PT I35PT I45PT 2CHG
1CHG 2 +LOOP ;

(5 POINT POST-WEAVE)

: FVPT S0 3 Y + 3 DUP S1 3 Y + 3 MOD T1 ! TM0 3 X + ! ;
: 1FVPT S3 3 Y + 3 S5 3 Y + 3 MOD T5 ! ;
: 2FVPT S3 3 Y + 3 S4 3 Y + 3 SBT T3 ! ;
: 3FVPT T1 3 S2 3 Y + 3 OVER OVER MOD T2 ! SBT T4 ! ;
: 4FVPT T2 3 T3 3 OVER OVER MOD TM 3 X + ! SBT
TM3 3 X + ! ;
: 5FVPT T4 3 T5 3 OVER OVER MOD TM1 3 X + ! SBT
TM2 3 X + ! ;
: 05PT INT2 SINT 24 0 DO FVPT 1FVPT 2FVPT 3FVPT
4FVPT 5FVPT 2CHG 1CHG 2 +LOOP ;

(4 POINT POST-WEAVE)

: 401 10 0 DO I X + 3 I Y + ! 2 +LOOP ;
: 140 10 0 DO I 20 + X + 3 I 30 + X + 3 OVER OVER
MOD I 10 + Y + ! SBT I 30 + Y + ! I 10 + X + ! I 20 + Y + ! 2 +LOOP ;
Appendix-C

: 402 10 0 DC I 40 + X + 3 I 40 + Y + ! 2 + LOOP ;
: 240 10 0 DC I 60 + X + 3 I 70 + X + 3 OVER OVER MOD I
   50 + Y + ! SBT I 70 + Y + ! I 50 + X + 3 I 60 + Y + !
   2 + LOOP ;
: 403 10 0 DO I 90 + X + 2 I 80 + Y + ! 2 + LOOP ;
: 340 10 0 DO I 100 + X + 3 I 110 + X + 3 OVER OVER MOD I
   90 + Y + ! SBT I 110 + Y + ! I 90 + X + 3
   I 100 + Y + ! 2 + LOOP ;
: 04 PT 401 140 402 240 403 340 ;

: 5

( 3 POINT POST-WEAVE )
: 03 PT 40 0 DO I Y + 3 I 40 + Y + 3 MOD
   I 80 + Y + 3 OVER OVER MOD I 40 + X + ! SBT I 90
   + X + ! I Y + 3 I X + ! 2 + LOOP ;
( INPUT RE-ORDERING VECTOR RF )
: IORD 120 0 DO I RF + 3 Y + 3 I X + ! 2 + LOOP ;
( OUTPUT RE-ORDERING VECTOR RF1 )
: GORD 120 0 DO I X + 3 I RF1 + 3 Y + ! 2 + LOOP ;

: S

: TRANSFORM IORD I3PT I4 PT I5PT MULT O5PT
   O4 PT O3 PT O2 D O1 D O0 D ;
: 1TRANSFORM IORD I3PT I4PT I5PT CMULT O5PT
   O4 PT O3 PT O2 D O1 D O0 D ;

( FRO FOR FORWARD AND INV FOR INVERSE TRANSFORM
 USING MULTIPLY AND DIVIDE INSTRUCTION )
: FRO O FLAG ! TRANSFORM ; : INV I FLAG ! TRANSFORM ;
( 1FRD FOR FORWARD 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)
Appendix D

* *
************************************************************************************
* *
* PROCESSOR NUMBER 1
* *
************************************************************************************

NAME 68091 | SKP15  STD  MCND

OUTPUT
EQU $0400  |

STATUS
EQU $0402  |

T0
EQU $0403  |

T2
EQU $0405  |

INPUT
EQU $0410  |

R6
EQU $0412  |

R2
EQU $0414  |

SEM
EQU $0416  |

* *

CRG  $F800  |

MDP
CRCC  #$01010000  |

LDO  #PRED1  |

BEGIN
CLRA  |

STA  FLAG  |

LDA  SEM  |

BEQ  FRO  |

START
LDA  #1  |

STA  FLAG  |

FRD
LGY  #MCND  |

LDX  #MLTFR  |

LDA  #1  |

STA  STATUS  |

SYNC  |

SYNC
CLRA  |

STA  STATUS  |

LDD  INPUT  |

NEXT
LDY  #MCND  |

LDX  #MLTFR  |

SYNC  |

LDD  SAVE  |

OVER
SYNC  |

SYNC
ADDD  R6  |

BLO  SKP13  |

SKP12
ADDD  #15  |

SYNC  |

SYNC
ADDD  R2  |

BLO  SKP15  |

SKP14
ADDD  #15  |

*
<table>
<thead>
<tr>
<th>BCS</th>
<th>SKP22</th>
<th></th>
<th>STD</th>
<th>,U</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td>LDA</td>
<td>1,U</td>
</tr>
<tr>
<td>SLD</td>
<td>SKP23</td>
<td></td>
<td>LDB</td>
<td>#15</td>
</tr>
<tr>
<td><strong>SKP22</strong></td>
<td></td>
<td></td>
<td><strong>MUL</strong></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>#15</td>
<td></td>
<td><strong>SKP23</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td></td>
<td></td>
<td><strong>RCS</strong></td>
<td>LDP20</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td></td>
<td></td>
<td><strong>BLO</strong></td>
<td>LDP21</td>
</tr>
<tr>
<td>STD</td>
<td>T2</td>
<td></td>
<td><strong>ADD</strong></td>
<td>2,U</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td></td>
<td></td>
<td><strong>LDP20</strong></td>
<td>ADDD #15</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td></td>
<td></td>
<td><strong>LDP21</strong></td>
<td>STG 2,U</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td></td>
<td></td>
<td><strong>LDA</strong></td>
<td>,U</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td></td>
<td></td>
<td><strong>CLD</strong></td>
<td>,X</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>SAVE</td>
<td></td>
<td><strong>CLR</strong></td>
<td>1,X</td>
</tr>
<tr>
<td><strong>LDX</strong></td>
<td>flag</td>
<td></td>
<td><strong>CLR</strong></td>
<td>2,X</td>
</tr>
<tr>
<td><strong>CMPA</strong></td>
<td>#1</td>
<td></td>
<td><strong>LDB</strong></td>
<td>#15</td>
</tr>
<tr>
<td><strong>SEQ</strong></td>
<td>MULT</td>
<td></td>
<td><strong>MUL</strong></td>
<td>,Y</td>
</tr>
<tr>
<td><strong>CMPA</strong></td>
<td>#2</td>
<td></td>
<td><strong>STD</strong></td>
<td>,X</td>
</tr>
<tr>
<td><strong>REQ</strong></td>
<td>CONV</td>
<td></td>
<td><strong>LDA</strong></td>
<td>,X</td>
</tr>
<tr>
<td><strong>LDQ</strong></td>
<td>SAVE</td>
<td></td>
<td><strong>LDB</strong></td>
<td>#15</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>RES</td>
<td></td>
<td><strong>MUL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LBRA</strong></td>
<td>BEGIN</td>
<td></td>
<td><strong>ADD</strong></td>
<td>1,X</td>
</tr>
<tr>
<td><strong>LDP</strong></td>
<td>SAVE</td>
<td></td>
<td><strong>ADD</strong></td>
<td>2,U</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>OUTPUT</td>
<td></td>
<td><strong>RCS</strong></td>
<td>LDP22</td>
</tr>
<tr>
<td><strong>LBRA</strong></td>
<td>BEGIN</td>
<td></td>
<td><strong>LDP22</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MULT</strong></td>
<td></td>
<td></td>
<td><strong>LDP23</strong></td>
<td>STD SAVE</td>
</tr>
<tr>
<td>INCR</td>
<td>FLAG</td>
<td></td>
<td><strong>LOP22</strong></td>
<td>ADDD #15</td>
</tr>
<tr>
<td><strong>LDY</strong></td>
<td>#SAVE</td>
<td></td>
<td><strong>LOP23</strong></td>
<td>STG ALADD</td>
</tr>
<tr>
<td><strong>CLR</strong></td>
<td>#RES</td>
<td></td>
<td><strong>SYNC</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CLR</strong></td>
<td>,U</td>
<td></td>
<td><strong>SYNC</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CLR</strong></td>
<td>1,X</td>
<td></td>
<td><strong>SYNC</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LDA</strong></td>
<td>1,Y</td>
<td></td>
<td><strong>SYNC</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LDB</strong></td>
<td>1,Y</td>
<td></td>
<td><strong>LBRA</strong></td>
<td>NEXT</td>
</tr>
<tr>
<td><strong>MUL</strong></td>
<td></td>
<td></td>
<td><strong>MLTR</strong></td>
<td>FDB 1</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>2,U</td>
<td></td>
<td><strong>MLTRR</strong></td>
<td>FDB 61153</td>
</tr>
<tr>
<td><strong>LDA</strong></td>
<td>,Y</td>
<td></td>
<td><strong>ORG</strong></td>
<td>#0000</td>
</tr>
<tr>
<td><strong>LDB</strong></td>
<td>1,Y</td>
<td></td>
<td><strong>PROC0</strong></td>
<td>FCB 0</td>
</tr>
<tr>
<td><strong>MUL</strong></td>
<td></td>
<td></td>
<td><strong>PROC1</strong></td>
<td>FCB 0</td>
</tr>
<tr>
<td><strong>ADD</strong></td>
<td>1,U</td>
<td></td>
<td><strong>PROC2</strong></td>
<td>FCB 0</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>1,U</td>
<td></td>
<td><strong>PROC3</strong></td>
<td>FCB 0</td>
</tr>
<tr>
<td><strong>BCC</strong></td>
<td>LOP16</td>
<td></td>
<td><strong>LDA</strong></td>
<td>1,X</td>
</tr>
<tr>
<td><strong>INC</strong></td>
<td>,U</td>
<td></td>
<td><strong>LDB</strong></td>
<td>,Y</td>
</tr>
<tr>
<td><strong>MUL</strong></td>
<td></td>
<td></td>
<td><strong>TEMP</strong></td>
<td>FCB 0</td>
</tr>
<tr>
<td><strong>ADD</strong></td>
<td>1,U</td>
<td></td>
<td><strong>TEMP1</strong></td>
<td>FCB 0</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>1,U</td>
<td></td>
<td><strong>LDA</strong></td>
<td>1,U</td>
</tr>
<tr>
<td><strong>BCC</strong></td>
<td>LOP19</td>
<td></td>
<td><strong>LDB</strong></td>
<td>1,Y</td>
</tr>
<tr>
<td><strong>INC</strong></td>
<td>,U</td>
<td></td>
<td><strong>LDA</strong></td>
<td>,Y</td>
</tr>
<tr>
<td><strong>MUL</strong></td>
<td></td>
<td></td>
<td><strong>LDB</strong></td>
<td>1,Y</td>
</tr>
<tr>
<td><strong>ADD</strong></td>
<td>,U</td>
<td></td>
<td><strong>LDA</strong></td>
<td>,Y</td>
</tr>
</tbody>
</table>

**Appendix-D**

LDP16

| LDA  | 1,X | | **LDA** | 1,Y |
| LDB  | ,Y | | **LDB** | ,Y |
| MUL  | | | **MLTR** | FDB 61153 |
| **ADD** | 1,U | | **PROC** | FCB 0 |
| **STD** | 1,U | | **PROC1** | FCB 0 |
| **BCC** | LOP19 | | **LDA** | 1,Y |
| **INC** | ,U | | **LDA** | ,Y |
| **MUL** | | | **LDA** | ,Y |
| **ADD** | ,U | | **LDA** | ,Y |
Appendix-J

CLR 2, X | LDR 15 | LOP 15 | CLR 1, U
MUL | LOP 15 | CLR 1, U
STD , X | STD 2, U
LDA , X | LDA 1, Y
LDB 15 | LDB 1, Y
MUL | MUL
ADD 1, X | ADD 1, U
ADD 2, U | ADD 1, U
BCLS SKP24 | MUL
CMPD #65521 | ADD 1, U
BLO SKP25 | STD 1, U
SKP24 | INC 1, U
ADD I R1 | LOX P
BCLS SKP26 | ADD 1, U
CMPD #65521 | STD 1, U
BLO SKP27 | ECD LOP19
SKP27 | ADD 1, U
STD T3 | LOP 19 | LDA 1, U
SYNC | STD 1, U
ADD 1 R3 | STD 1, U
BCLS SKP28 | STD 1, U
CMPD #65521 | STD 1, U
BLO SKP29 | STD 1, U
SKP28 | STD 1, U
ADD 15 | STD 1, U
SKP29 | STD 1, U
STD T5 | STD 1, U
SYNC | STD 1, U
ADD 1 R5 | STD 1, U
BCLS SKP30 | STD 1, U
CMPD #65521 | STD 1, U
BLO SKP31 | STD 1, U
SKP30 | STD 1, U
ADD 15 | STD 1, U
SKP31 | STD 1, U
STD T7 | STD 1, U
SYNC | STD 1, U
ADD 1 R7 | STD 1, U
BCLS SKP32 | STD 1, U
CMPD #65521 | STD 1, U
BLO LOP 20 | STD 1, U
SKP32 | STD 1, U
ADD 15 | STD 1, U
STD T9 | STD 1, U
SYNC | STD 1, U
STD SAVE | STD 1, U
LOAD FLAG | STD 1, U
CMPA #1 | STD 1, U
REQ MULT | STD 1, U
CMPA #2 | STD 1, U
REQ CONV | STD 1, U
LOAD SAVE | STD 1, U
STD RES | STD 1, U
LDBA BEG | STD 1, U
SDA #15 | STD 1, U
CURR | STD 1, U
LOAD #15 | STD 1, U
LOAD #15 | STD 1, U
STD OUTPUT | STD 1, U
LDBA #15 | STD 1, U
MULT | STD 1, U
INCM 1 | STD 1, U
LDX #15 | STD 1, U
SAVE | STD 1, U
LOD #15 | STD 1, U
STD 1, U
LDBA #15 | STD 1, U
CONV | STD 1, U
LOAD #15 | STD 1, U
STD 1, U
LDBA #15 | STD 1, U
MULT | STD 1, U
INCM 1 | STD 1, U
LDX #15 | STD 1, U
SAVE | STD 1, U
LOD #15 | STD 1, U
Appendix D

* LUP23  STD T3  | ORG $0000
   SYNC  | MCND  FDB 0
   LDD R3  | PPOD1 FCB 0
   STD T5  | PPOD2 FCB 0
   SYNC  | PPOD3 FCB 0
   SYNC  | PPOD4 FCB 0
   LDD R3  | TEMP FCB 0
   SYNC  | TEMP1 FCB 0
   STD SAVE  | TEMP 2 FCB 0
   LRRA NEXT  | SAVE FDB 0
   * | RES FDB 0
   MLTFR FDB 16379 | TGR $FFFF
   MLTRR FDB 5460  | STRT FQU $F000
   * | END BEGIN
   * |
   "**********************************************************************
   * PROCESOR NUMBER 3
   "**********************************************************************

   NAME  68093  | OVER  SYNC
   OUTPUT EQU $0400  | SYNC
   STATUS EQU $0402  | ADD D R8
   T6 EQU $0403  | EGS SKP12
   T4 EQU $0405  | CMPD #65521
   T2 EQU $0407  | BLO SKP12
   INPUT EQU $0410  | SKP12 ADD D #15
   R8 EQU $0412  | SKP13 STD T4
   R4 EQU $0414  | SYNC
   R2 EQU $0416  | ADD D R4
   SEM EQU $0418  | MCS SKP14
   * ORG $F800  | CMPD #65521
   NOP  | BLO SKP15
   QPCC #00100000  | SKP16 STD T2
   LDU #PPOD1  | SYNC
   BEG I NS START | STD SAVE
   CLRA  | LDD #2
   STA FLAG  | STD SAVE
   LDA SEM  | ADD D #55521
   BEQ FRO  | BCC SKP16
   START LDA #1  | SKP16 SYNC
   STA FLAG  | STC 1
   * FRO LDD #MCND  | STD MCND
   LDY #MLTFR  | CLR ,U
   LDA #1  | CLR 1,U
   STA STATUS  | LDA 1,X
   SYNC  | LDB 1,Y
   CLRA  | MUL
   STA STATUS  | MUL
   LDD INPUT  | STD 2,U
   BRA OVER  | LDA 1,X
   NEXT LDY #MCND  | LDB 1,Y
   LDX #MLTRP  | MUL
   SYNC  | ADD D 1,U
   LDD SAVE  | STD 1,U
   RCR SKP18
<table>
<thead>
<tr>
<th>Appendix-D</th>
<th>0-5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SKP18</strong></td>
<td></td>
</tr>
<tr>
<td>INC $u$</td>
<td></td>
</tr>
<tr>
<td>LDA 1,$x$</td>
<td></td>
</tr>
<tr>
<td>LDR $y$</td>
<td>BCS SKP2P</td>
</tr>
<tr>
<td>MUL</td>
<td>CMPD $65521$</td>
</tr>
<tr>
<td>ADDD 1,$u$</td>
<td>BLO SKP2o</td>
</tr>
<tr>
<td>STD 1,$u$</td>
<td>SKP29 ADDD #15</td>
</tr>
<tr>
<td>BCC SKP21</td>
<td>SKP29 STD T2</td>
</tr>
<tr>
<td>INC $u$</td>
<td>BCC SKP21</td>
</tr>
<tr>
<td>LDA $x$</td>
<td>STD SAVE</td>
</tr>
<tr>
<td>LDR $y$</td>
<td>LDA FLAG</td>
</tr>
<tr>
<td>MUL</td>
<td>CHPA #1</td>
</tr>
<tr>
<td>ADDD $u$</td>
<td>BCC SKP22</td>
</tr>
<tr>
<td>STD $u$</td>
<td>BCC #2</td>
</tr>
<tr>
<td>ADDD #15</td>
<td>LDA SAVE</td>
</tr>
<tr>
<td>STD 2,$u$</td>
<td>STD RES</td>
</tr>
<tr>
<td>LDA 1,$u$</td>
<td>LDA BARR BEGIN</td>
</tr>
<tr>
<td>MUL</td>
<td>LDA 2,$u$</td>
</tr>
<tr>
<td>ADDD 2,$u$</td>
<td>LOD SAVE</td>
</tr>
<tr>
<td>STD #15</td>
<td>CODV LOD OUTPUT</td>
</tr>
<tr>
<td>ADDD SKP22</td>
<td>LDA BARR BEGIN</td>
</tr>
<tr>
<td>CMPD $65521$</td>
<td>BCC CODV</td>
</tr>
<tr>
<td>SDC SKP23</td>
<td>BCC SKP22</td>
</tr>
<tr>
<td>ADDD #15</td>
<td>INC FLAG</td>
</tr>
<tr>
<td>STD 2,$u$</td>
<td>LDX #SAVE</td>
</tr>
<tr>
<td>STD $u$</td>
<td>LOY #RES</td>
</tr>
<tr>
<td>LDA $u$</td>
<td>LDP15 CLR ,u</td>
</tr>
<tr>
<td>LDX #TEMP</td>
<td>CLR 1,$u$</td>
</tr>
<tr>
<td>CLR $x$</td>
<td>LDA 1,$x$</td>
</tr>
<tr>
<td>CLR 1,$x$</td>
<td>LDR 2,$x$</td>
</tr>
<tr>
<td>CLR 2,$x$</td>
<td>MUL 2,$u$</td>
</tr>
<tr>
<td>LDB #15</td>
<td>STD 2,$u$</td>
</tr>
<tr>
<td>MUL</td>
<td>LDA $x$</td>
</tr>
<tr>
<td>STD $x$</td>
<td>LDB 1,$y$</td>
</tr>
<tr>
<td>LDA $x$</td>
<td>MUL</td>
</tr>
<tr>
<td>LDB #15</td>
<td>ADDD 1,$u$</td>
</tr>
<tr>
<td>MUL</td>
<td>STD 1,$u$</td>
</tr>
<tr>
<td>ADDD 1,$x$</td>
<td>INC #1</td>
</tr>
<tr>
<td>ADDD 2,$u$</td>
<td>LDP16</td>
</tr>
<tr>
<td>BCS SKP24</td>
<td>BCC LDP16</td>
</tr>
<tr>
<td>CMPD $65521$</td>
<td>BCC SKP24</td>
</tr>
<tr>
<td>BLC SKP25</td>
<td>BCC LDP19</td>
</tr>
<tr>
<td>ADDD #15</td>
<td>BCC SKP25</td>
</tr>
<tr>
<td>BCC SKP24</td>
<td><strong>SKP25</strong></td>
</tr>
<tr>
<td>BCC SKP25</td>
<td><strong>SKP25</strong></td>
</tr>
<tr>
<td>INC</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>STD T2</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>SYNC</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>STD SAVE</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>LDR #2</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>SUBD SAVE</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>ADDD SKP26</td>
<td>BCC SKP26</td>
</tr>
<tr>
<td>ADDD #65521</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>STD T4</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>SUBD SAVE</td>
<td>LDP19 LOA ,x</td>
</tr>
<tr>
<td>ADDD #65521</td>
<td>BCC SKP26</td>
</tr>
<tr>
<td>STD #4</td>
<td>ADD #15</td>
</tr>
<tr>
<td>ADDD #4</td>
<td>ADD #15</td>
</tr>
<tr>
<td>ADDD #4</td>
<td>ADD #15</td>
</tr>
<tr>
<td>SKP26</td>
<td>ADD #15</td>
</tr>
</tbody>
</table>
BCS   LDP20     |   SYNC
CMPD #65521       |   SYNC
LDP20

LUP20

A RDD  #15        |   LDD  #22
LUP21

STD  2, U         |   Sync

LDP23

A RDD  #15

LUP22

STD  T2

LUP23

SYNC

STD  R2

STD  SAVE

LDA  #1

EQU  $0400

STA   STATUS

EQU  $0402

SYNC

EQU  $0403

CLRA

EQU  $0405

STA   STATUS

T9

EQU  $0407

LDD  INPUT

T3

EQU  $0410

BRA   OVER

T16

EQU  $0412

NEXT  L DY #MCND

INPUT

EQU  $0414

LOG  #MLTFR

R5

EQU  $0416

SYNC

SEM

EQU  $0418

LDD  SAVE

ORG  $F800

OVER  SYNC

NOP

ORCC #01010000

ADD U  #9

BEGIN

CLRA

CMPD #65521

ADD U  #15

STA  FLAG

ADD U  #9

LDA  SEM

BEQ  FRD

SUB  P3

BEQ  F RD

SKP12

BEQ  T3

T1

STA  FLAG

ADDU  #15

LDY  #HMCND

BCC  SKP14

LDX  #MLTFR

ADDU  #65521
SKP14

STD   T16        | BCS            | SKP22
SYNC                          | CMPD           | #65521
SYNC                          | BLD            | SKP23

**

STD  MCHD        | SKP22          | ADOO #15
CLR   ,U          | SKP23          | SYNC
CLR   1, U        |                | SYN
LDA   1, X        | ADOO           | P16
LDR   1, Y        | RCS            | SKP24
MUL                           | CMPD           | #65521
STD   2, U        | BLD            | SKP25
LDA   1, X        | SKP24          | ADOO #15
LDR   1, Y        | SKP25          | SYNC
MUL                           | STD            | T3
ADOO  1, U        |                | SYNC
STD   1, U        | STD            | SAVE
BCC   SKP16       | LDD            | R3
INC   ,U          | SUBD           | SAVE
LDA   1, X        | BCC            | SKP26
LDB   , Y        | ADOO           | #65521
MUL                           | SKP26          | STD T9
ADOO  1, U        |                | SYNC
STD   1, U        |                | SYNC
BCC   SKP19       |                | **
INC   ,U          | STD            | SAVE
LDA   , Y        | LDA            | FLAG
LDB   , Y        | CMPA           | #1
MUL                           | BEOQ           | MULT
ADOO  1, U        | CMPA           | #2
STD   , U        | BEOQ           | CONV
ADOO  1, U        | LDD            | SAVE
STD   , U        | STD            | RES
LDA   , Y        | LDBRA          | BEGIN
LDB   #15        | ICONDV         | LDD SAVE
MUL                           | STD            | OUTPUT
ADOO  2, U        | LBPA           | BEGIN
BCC   SKP20       |                | **
CMPD           | MULT           | INC FLAG
RLO   SKP21       |                | **
ADOO  #15        | LDX            | #SAVE
SKP20

SKP21

STD  2, U        | LOP15          | CLR ,U
LDA   , U        |                | CLR 1, U
LOX   #TEMP      | LDA            | 1, X
CLR   , X        | LDR            | 1, Y
CLR   1, X        | MUL            | 
CLR   2, X        | STD            | 2, U
LDB   #15        | LDA            | , Y
MUL                           | LDR            | 1, Y
STD   , X        | MUL            | 
LDA   , Y        | ADOO           | 1, U
LDR   #15        | STD            | 1, U
MUL                           | BCC            | LOP16
ADOO  1, X        | INC            | , U
ADOO  2, U        | LOP16          | LDA 1, X
Appendix-D

LUP19

| LDR ,Y | | ADDD 2, U |
| MUL | | BCS LOP22 |
| ADDD 1, U | | CMPD #65521 |
| STD 1, U | | BLO LOP23 |
| BCS LOP19 | | LOP22 ADDD #15 |
| INC ,U | | LOP23 SYNC |
| LOA ,X | | STD T3 |
| LDR ,Y | | LDA P16 |
| ADDD ,U | | STD SAVE |
| STD ,U | | SYNC |
| | | | LSRA NEXT |
| LDR 1, U | | MLTRR EOR 5492 |
| MUL | | MLTRR EOR 5492 |
| ADDD ,U | | CMPO #65521 |
| BCS LOP20 | | SLO LOP21 |
| CMPD #65521 | | ORG #0000 |
| SLO LOP21 | | ORG #0000 |

LUP20

| LDA 2, U | | PPO01 FCB 0 |
| STD T3 | | PPO02 FCB 0 |
| LDX #TEMP | | PPO3 FCB 0 |
| CLR ,X | | PPO4 FCB 0 |
| CLR 1, X | | TEMP FCB 0 |
| CLR 2, X | | TEMP1 FCB 0 |
| LDB #15 | | TEMP3 FCB 0 |
| MUL | | SAVE FDB 0 |
| STD ,X | | FLAG FCB 0 |
| LDA ,X | | RES FDB 0 |
| LDR #15 | | ORG #FFFF |
| MUL | | START EQU #900 |
| ADDD 1, X | | ENP PEGIN |

Processor Number 5

*******************************

OUTPUT EQU $0400
STATUS EQU $0402
T10 EQU $0403
T2 EQU $0405
T16 EQU $0407
INPUT EQU $0410
R10 EQU $0412
R2 EQU $0414
R16 EQU $0416
SEM EQU $0418

ORG $F800
NOP
ORCC #Y01010000 NEXT LDY #MCMD
LDU #PROD1 | | LOX #MLTRR
BEGIN CLR A
<table>
<thead>
<tr>
<th>OVER</th>
<th>LDD</th>
<th>SAVE</th>
<th>SKP20</th>
<th>ADDD</th>
<th>#15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKP12</td>
<td>ADDD</td>
<td>#15</td>
<td></td>
<td>LDA</td>
<td>,U</td>
</tr>
<tr>
<td>SKP13</td>
<td>STD</td>
<td>T2</td>
<td></td>
<td>LDX</td>
<td>*TEMP</td>
</tr>
<tr>
<td></td>
<td>SYNC</td>
<td></td>
<td></td>
<td>CLR</td>
<td>X</td>
</tr>
<tr>
<td>ADDD</td>
<td>#10</td>
<td></td>
<td></td>
<td>CLR</td>
<td>1,X</td>
</tr>
<tr>
<td>PCS</td>
<td>SKP12</td>
<td></td>
<td></td>
<td>CLR</td>
<td>2,Y</td>
</tr>
<tr>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLO</td>
<td>SKP13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>#15</td>
<td></td>
<td></td>
<td>ADDD</td>
<td>1,X</td>
</tr>
<tr>
<td>SYNC</td>
<td></td>
<td></td>
<td></td>
<td>ADDD</td>
<td>2,U</td>
</tr>
<tr>
<td>STD</td>
<td>SAVE</td>
<td></td>
<td></td>
<td>RCS</td>
<td>SKP22</td>
</tr>
<tr>
<td>STD</td>
<td>#2</td>
<td></td>
<td></td>
<td>STD</td>
<td>SAVE</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td>ADDD</td>
<td>#65521</td>
</tr>
<tr>
<td>STD</td>
<td>2,U</td>
<td></td>
<td></td>
<td>STD</td>
<td>T2</td>
</tr>
<tr>
<td>STD</td>
<td>MCHD</td>
<td></td>
<td></td>
<td>STD</td>
<td>SAVE</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td>STD</td>
<td>T2</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td>STD</td>
<td>SAVE</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td>STD</td>
<td>SAVE</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td>STD</td>
<td>T10</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td>STD</td>
<td>MCHD</td>
</tr>
<tr>
<td>ADDD</td>
<td>#1,U</td>
<td></td>
<td></td>
<td>STD</td>
<td>T10</td>
</tr>
<tr>
<td>STD</td>
<td>#1,U</td>
<td></td>
<td></td>
<td>STD</td>
<td>SAVE</td>
</tr>
<tr>
<td>STD</td>
<td>#1,U</td>
<td></td>
<td></td>
<td>STD</td>
<td>OUTPUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKP19</td>
<td>LDD</td>
<td>#15</td>
<td></td>
<td>LBPA</td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>#2,U</td>
<td></td>
<td></td>
<td>BCS</td>
<td>SKP20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKP16</td>
<td>LDA</td>
<td>1,X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>#1,U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>,U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>,X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td>BCS</td>
<td>SKP20</td>
</tr>
<tr>
<td>ADDD</td>
<td>#2,U</td>
<td></td>
<td></td>
<td>ALLO</td>
<td>STD</td>
</tr>
<tr>
<td>STD</td>
<td>T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>MCHD</td>
<td></td>
<td></td>
<td>STD</td>
<td>SAVE</td>
</tr>
<tr>
<td>STD</td>
<td>MCHD</td>
<td></td>
<td></td>
<td>STD</td>
<td>RES</td>
</tr>
<tr>
<td>STD</td>
<td>MCHD</td>
<td></td>
<td></td>
<td>STD</td>
<td>RES</td>
</tr>
<tr>
<td>STD</td>
<td>MCHD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LUP15
CLR ,U      | LDB  #15
CLR 1,U     | MUL
LOA 1,X     | STD ,X
LOB 1,Y     | LOA ,X
MUL          | LOB  #15
STD 2,U     | MUL
LOA ,X      | ADD 1,X
LOB 1,Y     | ADD  2,U
MUL          | BCS LOP22
ADDD 1,U    | CMPD #65521
STD 1,U     | BLO LOP23
BCC LOP16   | LOP22 ADD  #15
INC ,U      | LOP23 STD T16
LUP16
LDA 1,X     | SYNC
LDB ,Y      | SYNC
MUL          | LOD R2
ADDD 1,U    | STD SAVE
STD 1,U     | SYNC
BCC LOP19   | SYNC
INC ,U      | LBA NEXT
LUP19
LDA ,X      | ****************************
LDB ,Y      | MLTFR FDB 19136
MUL          | MLTRR FDB 46773
ADDD ,U     | *
STD ,U      | ORG $0000

LDA 1,U     | INC40 FDB 0
LDR  #15     | PRC01 FCB 0
MUL          | PRC02 FCB 0
ADDD 2,U    | PRC03 FCB 0
BCS LOP20   | PRC04 FCB 0
CMPD #65521 | TEMP FCB 0
BLO LOP21   | TEMP1 FCB 0
ADDD  #15   | SAVE FDB 0
LUP20
LUP21
STD 2,U     | FLAG FCB 0

LDA ,U      | *
LDX #TEMP    | ORG $FFFF
CLR ,X      | STRT EQU $F800
CLP 1,X     | END BEGIN
CLP 2,X     | *

*******************************************************************************
*            PROCESSOR NUMBER 6            *
*******************************************************************************
NAM 68096    | SEM EQU $0418
OUTPUT      | EQU $0400
STATUS      | EQU $0402
T11         | EQU $0403
T1          | EQU $0405
T7          | EQU $0407
INPUT       | EQU $0410
R11         | EQU $0412
R1          | EQU $0414
R7          | EQU $0416

START
LDA #1
STA FLAG
LDY #MCND
LDX #MLTFP
LDA #1
STA STATUS
SYNC
CLRA
STA STATUS
LDD INPUT
BRA OVER
LDY #MCND
LDX #MLTRK
SYNC
LDD SAVE
SKI#20 ADDD #15

NEXT
STO T1
SYNC
ADDD #11
PCS SKP12
CMPD #65521
BLO SKP13
ADDD #15
LDB #15
SKP12
STO T1
SYNC
SYNC
SYNC
ADDD R7
PCS SKP14
CMPD #65521
BLO SKP15
ADDD #15
SKP13
STO MCND
CLR ,U
CLR 1,X
LDA 1,Y
MUL
STD 2,U
LDA 1,X
LDB 1,Y
MUL
ADDD 1,U
STD 1,U
BCC SKP16
INC ,U
LDA 1,X
LDB 1,Y
MUL
ADDD 1,U
STD 1,U
BCC SKP19

OVER
STO T1
SYNC
ADDD R11
PCS SKP12
CMPD #65521
BLO SKP13
ADDD #15
LDB #15
SKP12
STO T1
SYNC
SYNC
SYNC
ADDD R7
PCS SKP14
CMPD #65521
BLO SKP15
ADDD #15
SKP13
STO MCND
CLR ,U
CLR 1,X
LDA 1,Y
MUL
STD 2,U
LDA 1,X
LDB 1,Y
MUL
ADDD 1,U
STD 1,U
BCC SKP16
INC ,U
LDA 1,X
LDB 1,Y
MUL
ADDD 1,U
STD 1,U
BCC SKP19

APPENDIX-D
<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Address 1</th>
<th>Address 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKP27</td>
<td>STD</td>
<td>SAVE</td>
<td>ILDP20</td>
<td>ADDD #15</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>FLAG</td>
<td>ILDP21</td>
<td>STD 2,U</td>
</tr>
<tr>
<td></td>
<td>CMPA #1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEQ</td>
<td>MULT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMPA #2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEQ</td>
<td>CONV</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDD</td>
<td>SAVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>RES</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBRA</td>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUNV</td>
<td>LDD</td>
<td>SAVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>OUTPUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LBRA</td>
<td>BEGIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT</td>
<td>INC</td>
<td>FLAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDX</td>
<td>#SAVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDY</td>
<td>#RES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUP15</td>
<td>CLR</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLR</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>1,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>1,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>2,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>1,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCC</td>
<td>LDP16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INC</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUP16</td>
<td>LDA</td>
<td>1,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCC</td>
<td>LDP19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INC</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDD</td>
<td>3,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCS</td>
<td>LDP20</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMPD #65521</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLO</td>
<td>LDP21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDB</td>
<td>#15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDD</td>
<td>3,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCS</td>
<td>LDP20</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMPD #65521</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLO</td>
<td>LDP21</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PROCESSOR NUMBER 7

* 

NAME 68097 | ADDO P10
OUTPUT EQU $0400 | RCS SKP14
STATUS EQU $0402 | CMPD #65521
T12 EQU $0403 | SLQ SKP15
T2 EQU $0405 | SKP14 ADDO #15
T10 EQU $0407 | SKP15 STD T8
T6 EQU $0409 | SYNC
T6 EQU $0405 | ADDO R8
INPUT EQU $0410 | RCS SKP16
R12 EQU $0412 | CMPD #65521
R2 EQU $0414 | BLO SKP17
R10 EQU $0416 | SKP16 ADDO #15
R8 EQU $0418 | SKP17 STD T6
R6 EQU $041A | SYNC
S&I EQU $041C | STD MCS

* 

ORG $F800 | CLRP ,U
NOP | CLR 1, U
ORCC #X01010000 | LOA 1,X
LDU #PRD1 | LDA 1,Y

BEGIN 

CLRA 

STA FLAG | STD 2,U
LDA SEM | LOA ,X
BEQ FRD | LDB 1,Y

START 

LDA #1 | MUL
STA FLAG | ADDO 1,U
LDA #1 | MUL

FND 

LDY #MCHD | STD 1,Y
LDX #MLTRR | BCC SKP19
LOA #1 | INC ,U
STA STATUS | SKP13 LDA 1,X
SYNC | LDP ,Y
CLRA | MUL
STA STATUS | ADDO 1,U
LDD INPUT | STD 1,U
BRA OVEP | ACR SKP21

NEXT 

LDX #MCHD | INC ,U
LDX #MLTRR | SKP21 LDA ,X
SYNC | LDR ,Y
LDD SAVE | MUL

OVER 

STD T12 | ADDO ,U
SYNC | STD ,Y
ADD0 P12 | LOA 1,U
RCS SKP12 | LDB #15
CMPD #65521 | MUL
BLO SKP13 | ADDO 2,U

SKP12 ADD0 #15 | RCS SKP22
SKP13 STD T2 | CMPD #65521
SYNC | BLO SKP23
STD T10 | SKP22 ADDO #15
SYNC | SKP23 STD 2,U
<table>
<thead>
<tr>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Instruction 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA U</td>
<td>ISKP35 STD</td>
<td>SAVE</td>
</tr>
<tr>
<td>LDX #TEMP</td>
<td>LDA FLAG</td>
<td></td>
</tr>
<tr>
<td>CLR X</td>
<td>CMPA #1</td>
<td></td>
</tr>
<tr>
<td>CLR 1*X</td>
<td>REQ MULT</td>
<td></td>
</tr>
<tr>
<td>CLR 2*X</td>
<td>CMPA #2</td>
<td></td>
</tr>
<tr>
<td>LDD #15</td>
<td>REQ CONV</td>
<td></td>
</tr>
<tr>
<td>MUL X</td>
<td>LDD SAVE</td>
<td></td>
</tr>
<tr>
<td>STD X</td>
<td>STD RES</td>
<td></td>
</tr>
<tr>
<td>LDA X</td>
<td>LBRA BEGIN</td>
<td></td>
</tr>
<tr>
<td>LDR #15</td>
<td>LDD SAVE</td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td>LBRA BEGIN</td>
<td></td>
</tr>
<tr>
<td>ADDD 1*X</td>
<td>INC FLAG</td>
<td></td>
</tr>
<tr>
<td>ADDD 2*X</td>
<td>MULT</td>
<td></td>
</tr>
<tr>
<td>BCS #65521</td>
<td>LDX #SAVE</td>
<td></td>
</tr>
<tr>
<td>CMPD SKP25</td>
<td>LDY #RES</td>
<td></td>
</tr>
<tr>
<td>BLO SKP25</td>
<td>LOP15 CLR</td>
<td></td>
</tr>
<tr>
<td>SKP24</td>
<td>CLR 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP25</td>
<td>LDA 1,Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDR 1,Y</td>
<td></td>
</tr>
<tr>
<td>SKP26</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP27</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>STD T8</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SYNC</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>ADDD R6</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BCS SKP26</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BLO SKP27</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP28</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP29</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>STD T10</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SYNC</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>ADDD R8</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BCS SKP28</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BLO SKP29</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP30</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP31</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>ADDD R2</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BCS SKP32</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BLO SKP33</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP32</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP33</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>STD T12</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SYNC</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>ADDD R12</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BCS SKP34</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>BLO SKP35</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td>SKP34</td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDD 1,U</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDA #TEMP</td>
<td></td>
</tr>
<tr>
<td>SKP15</td>
<td>STD</td>
<td>T7</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>SYNC</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>LDD</td>
<td>R7</td>
<td></td>
</tr>
<tr>
<td>SUBD</td>
<td>SAVE</td>
<td></td>
</tr>
<tr>
<td>BCC</td>
<td>SKP16</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>#65521</td>
<td></td>
</tr>
</tbody>
</table>

| SKP16 | STD  | MCND  | | LOA  | T7   |
|-------|------|-------| | LOB   | T7   |
|       | SYNC |       | | STD   | SAVE |
| CLP   | 1;U  |       | | LDD   | R7   |
| LDA   | 1;X  |       | | STD   | SAVE |
| LDD   | 1;Y  |       | | LOB   | SP    |
| MUL   |       |       | | LDA   | SP    |
| STD   | 2;U  |       | | LDD   | SP    |
| LOA   | 1;X  |       | | BCC   | SKP18 |
| MUL   |       |       | | INC   | SP    |
| ADDD  | 1;U  |       | | MUL   | SP    |
| STD   | 1;U  |       | | ADDD  | P9   |
| BCC   | SKP18|       | | STD   | SAVE |
| INC   | 1;U  |       | | INC   | SP    |
| LDA   | 1;X  |       | | LDA   | SP    |
| LDD   | 1;Y  |       | | MUL   | SP    |
| STD   | 1;U  |       | | ADDD  | #15  |
| LDA   | 1;U  |       | | LDA   | SP    |
| LDR   | #15  |       | | LDR   | SP    |
| MUL   |       |       | | MUL   | SP    |
| ADDD  | 2;U  |       | | BCS   | SKP22 |
| BCC   | SKP22|       | | CMPD  | #65521 |
| CMPD  | #65521|      | | BLO   | SKP32 |
| ADDD  | #15  |       | | STD   | SP    |
| BCS   | SKP22|       | | STD   | #15  |
| CMPD  | #65521|      | | LDA   | TEMP |
| BLO   | SKP23|       | | BCS   | #15  |
| ADDD  | #15  |       | | LDA   | U    |
| STD   | 2;U  |       | | LDA   | X    |
| BCS   | SKP23|       | | LDX   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| ADDD  | #15  |       | | STD   | X    |
| BCS   | SKP23|       | | LDA   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| LDA   | 1;U  |       | | STD   | X    |
| BCS   | SKP23|       | | LDA   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| LDA   | 1;U  |       | | STD   | X    |
| BCS   | SKP23|       | | LDA   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| LDA   | 1;U  |       | | STD   | X    |
| BCS   | SKP23|       | | LDA   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| LDA   | 1;U  |       | | STD   | X    |
| BCS   | SKP23|       | | LDA   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| LDA   | 1;U  |       | | STD   | X    |
| BCS   | SKP23|       | | LDA   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| LDA   | 1;U  |       | | STD   | X    |
| BCS   | SKP23|       | | LDA   | TEMP |
| CMPD  | #65521|      | | CLR   | X    |
| BLO   | SKP23|       | | CLR   | 1;X  |
| LDA   | 2;X  |       | | CLR   | 2;X  |
| MUL   | #15  |       | | LDA   | #15  |
| STD   | X    |       | | MUL   | U    |
| LDA   | 1;U  |       | | STD   | X    |
LDX #SAVE | MUL
LDY #RES | STD ,X
LUP15
CLP ,X | LDA ,X
CLR 1,Y | LDB 1,X
MUL 1,Y | ADDD 1,X
STD 2,Y | RFS LOP22
LDA ,Y | CMPD $#5551
LDB 1,Y | PLO LOP23
MUL | LOP22 ADDD $#15
ADDD 1,Y | LOP23 STD T12
STD 1,Y | SYNC
BCC LOP16 | LDD 29
INC ,Y | STD T7
LUP16
LDA 1,X | LDD $#13
LDB ,Y | STD T9
MUL | SYNC
ADDD 1,U | LDO $#7
STD 1,U | STD SAVE
BCC LOP19 | SYNC
INC ,U | LOP19
LUP19
LDA ,Y | LBRA NEXT
LDA ,X | MUL ,X
ADDD ,U | MLTR# FDB $#7975
STD ,U | ORG $#0000
LDA 1,U | MCOND FDB 0
LDB $#15 | PRED1 FCB 0
MUL | PRED2 FCB 0
ADDD 2,Y | PRED3 FCB 0
RCS LOP20 | PRED4 FCB 0
CMPD $#65521 | TEMP FCB 0
BLO LOP21 | TEMP1 FCB 0
LUP20
ADDD $#15 | TEMP3 FCB 0
LUP21
STD 2,U | SAVE FDB 0
* | FLAG FCB 0
LDA ,Y | RES FDB 0
LDX #TEMP | ORG $#FFFF
CLR ,X | STRT EQU $#800
CLR 2,X | END "BEGIN
LDR $#15 | /*

* ***************************************************************************
* PROCESOR NUMBER 9
* ***************************************************************************
* ***************************************************************************

NAM $80999 | INPUT EQU $0410
OUTPUT EQU $0400 | IP14 EQU $0412
STATUS EQU $0402 | IR4 EQU $0414
T14 EQU $0403 | IR8 EQU $0416
T4 EQU $0405 | HR17 EQU $0418
T8 EQU $0407 | SEM EQU $041A
T17 EQU $0409 | /*
<table>
<thead>
<tr>
<th>INPUT</th>
<th>EQU</th>
<th>$0410</th>
<th>MUL</th>
</tr>
</thead>
<tbody>
<tr>
<td>R14</td>
<td>EQU</td>
<td>$0412</td>
<td>STD</td>
</tr>
<tr>
<td>R8</td>
<td>EQU</td>
<td>$0414</td>
<td>LOA</td>
</tr>
<tr>
<td>R17</td>
<td>EQU</td>
<td>$0416</td>
<td>LDR</td>
</tr>
<tr>
<td>SEM</td>
<td>EQU</td>
<td>$0418</td>
<td>MUL</td>
</tr>
<tr>
<td></td>
<td>ORG</td>
<td>$F800</td>
<td>INC</td>
</tr>
<tr>
<td></td>
<td>ORCC</td>
<td>#0010000</td>
<td>SKP16</td>
</tr>
<tr>
<td></td>
<td>LOU</td>
<td>#PROD</td>
<td>LDR</td>
</tr>
<tr>
<td>BEGIN</td>
<td>CLPA</td>
<td></td>
<td>MUL</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>FLAG</td>
<td>ADDD</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>SEM</td>
<td>STD</td>
</tr>
<tr>
<td></td>
<td>BEQ</td>
<td>FRO</td>
<td>BCC</td>
</tr>
<tr>
<td>START</td>
<td>LDA</td>
<td>#1</td>
<td>INC</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>FLAG</td>
<td>SKP19</td>
</tr>
<tr>
<td></td>
<td>LDY</td>
<td>#MCND</td>
<td>LDB</td>
</tr>
<tr>
<td></td>
<td>LDX</td>
<td>#MLTRR</td>
<td>MUL</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>#1</td>
<td>ADDD</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>STATUS</td>
<td>STD</td>
</tr>
<tr>
<td>SYN</td>
<td>CLRA</td>
<td></td>
<td>LDA</td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>STATUS</td>
<td>LDB</td>
</tr>
<tr>
<td></td>
<td>LDD</td>
<td>INPUT</td>
<td>MUL</td>
</tr>
<tr>
<td></td>
<td>BRA</td>
<td>OVER</td>
<td>ADDD</td>
</tr>
<tr>
<td>NEXT</td>
<td>LDY</td>
<td>#MCND</td>
<td>RCS</td>
</tr>
<tr>
<td></td>
<td>LDX</td>
<td>#MLTRR</td>
<td>CMPD</td>
</tr>
<tr>
<td></td>
<td>SYNC</td>
<td></td>
<td>LDO</td>
</tr>
<tr>
<td></td>
<td>LDD</td>
<td>SAVE</td>
<td>SKP20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SKP21</td>
</tr>
<tr>
<td>OVER</td>
<td>STD</td>
<td>T14</td>
<td>LDA</td>
</tr>
<tr>
<td></td>
<td>SYNC</td>
<td></td>
<td>LOX</td>
</tr>
<tr>
<td></td>
<td>ADDD</td>
<td>R14</td>
<td>RCS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CLR</td>
</tr>
<tr>
<td>SKP12</td>
<td>ADDD</td>
<td>#15</td>
<td>LDB</td>
</tr>
<tr>
<td>SKP13</td>
<td>STD</td>
<td>T4</td>
<td>MUL</td>
</tr>
<tr>
<td></td>
<td>SYNC</td>
<td></td>
<td>STD</td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>T8</td>
<td>LOA</td>
</tr>
<tr>
<td></td>
<td>SYNC</td>
<td></td>
<td>LDB</td>
</tr>
<tr>
<td></td>
<td>SUBD</td>
<td>R8</td>
<td>MUL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ADDD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ADDD</td>
</tr>
<tr>
<td>SKP14</td>
<td>ADDD</td>
<td>#65521</td>
<td>RCS</td>
</tr>
<tr>
<td></td>
<td>STD</td>
<td>T17</td>
<td>CMPD</td>
</tr>
<tr>
<td></td>
<td>SYNC</td>
<td></td>
<td>LDO</td>
</tr>
<tr>
<td></td>
<td>SKP22</td>
<td>ADDD</td>
<td>SKP23</td>
</tr>
<tr>
<td></td>
<td>CLP</td>
<td>#U</td>
<td>SKP23</td>
</tr>
<tr>
<td></td>
<td>CLR</td>
<td>1,U</td>
<td>SKP17</td>
</tr>
<tr>
<td></td>
<td>LDA</td>
<td>1,X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDR</td>
<td>1,Y</td>
<td></td>
</tr>
</tbody>
</table>
**Appendix-D**

```
  CMPD #65521  | LOP16  | LOA  1,X
  BLO SKP25    |        | LDR  1,Y

  SKP24
    ADDU #15    |        | MUL  

  SKP25
    SYNC       |        | ADDD 1,U
    STD T8     |        | STD 1,U
    SYNC       |        | BCC LOP19
    STD SAVE   |        | INC  1,U
    LDD R8     | LOP19  | LOA  1,Y
    SORD SAVE  |        | LOB  1,Y
    BCC SKP26  |        | MUL  
    ADDD #65521|        | ADDD 1,U

  SKP26
    SYNC       |        | STD  1,U
    ADDD R4    |        | 
    BCS SKP28  |        | LOA  1,U
    CMPD #65521|        | LDB  #15
    BLO SKP29  |        | MUL  

  SKP28
    ADDD #15    |        | ADDD 2,U

  SKP29
    STD T14    |        | RCS LOP20
    SYNC       |        | CMPD #65521
    ADDD R14    |        | BLO LOP21
    HCS SKP30   | LOP20  | ADDD #15
    CMPD #65521| LOP21  | STD  2,U
    BLO SKP31  |        | 

  SKP30
    ADDD #15    |        | LDA  1,U
    **        |        | 
    **        |        | 

  SKP31
    STD SAVF   |        | CLP  1,X
    LDA FLAG   |        | CLP  1,X
    CMPA #2     |        | CLR  2,Y
    BEQ MULT   |        | LDB  #15
    CMPA #2     |        | 
    BEQ CONV   |        | STD  1,Y
    LDD SAVE   |        | LOA  1,Y
    STD RES    |        | LOA  #15
    LRBA BEGIN |        | MUL  
    LDD SAVE   |        | 
    STD OUTPUT |        | 
    LRBA BEGIN |        | RCS LOP22

  **        |        | RCS LOP22
  **        |        | CMPD #65521

  CUNV
    LDD SAVE   |        | ADDD 1,X
    STD OUTPUT |        | ADDD 2,U
    LRBA BEGIN |        | RCS LOP22

  **        |        | RCS LOP22
  **        |        | CMPD #65521

  **        |        | 

  MUL
    INC FLAG   |        | 
    LDX #SAVE  | LOP22  | ADDD #15
    LDY #RES   | LOP23  | STD  T8

  LOP15
    CLR 1,U    |        | SYNC
    CLR 1,U    |        | 
    LDA 1,X    |        | LDD  R17
    LDB 1,Y    |        | STD  T14
    MUL 2,U    |        | LDD  R8 
    STD  1,Y   |        | STD  T17
    LDB 1,Y    |        | SYNC
    MUL 1,Y    |        | 
    ADDD 1,U   |        | STD  SAVE
    STD  1,U   |        | LBPA NEXT
    BCC LOP16  |        | 
    INC 1,U    |        | MUL FP  F08  25311
```
MLTRR  FDB 24521 | TEMP1  FCB 0
**
| ORG  $0000 | TEMP3  FCB 0
MCND  FDB 0 | IFLAG  FCB 0
PRD1  FCB 0 | RES  FDB 0
PRD2  FCB 0 | **
PRD3  FCB 0 | ORG  $FFFE
PRD4  FCB 0 | START  EQU  $E800
TEMP  FCB 0 | **

*PROCESOR NUMBER 10*

**

OUTPUT  EQU  $0400 | CMPD  $65521
STATUS  EQU  $0402 | SKP12  ADD  $15
T15  EQU  $0403 | SKP13  STD  T5
T>  EQU  $0405 | SYNC
T7  EQU  $0407 | STD  T7
T17  EQU  $0409 | SYNC
INPUT  EQU  $0410 | STD  SAVE
R15  EQU  $0412 | LOD  R7
R5  EQU  $0414 | SUBD  SAVE
R7  EQU  $0416 | RCD  SKP14
R17  EQU  $0418 | ADD  $65521
SLM  EQU  $041A | SKP14  STD  T17
**

ORG  $F800 | SYNC
NOP  **
ORCC  $01010000 | STD  MCND
LDU  #PROC1  | CLR  ,U
BEGIN  CLR  | CLR  1,U
STA  FLAG  | LDA  1,X
LDA  SLM  | LDB  1,Y
BEQ  FROD  | MUL
START  LDA  #1  | STD  2,U
STA  FLAG  | LDA  ,X
FKD  LDA  #MCND  | LDB  1,Y
LDA  #MLTRR  | MUL
LDA  #1  | ADD  1,U
STA  STATUS  | STD  1,U
SYNC  | BCD  SKP16
CLA  | INC  ,U
STA  STATUS  | SKP16  LDA  1,X
LDD  INPUT  | LDB  ,Y
BRA  OVER  | MUL
NEXT  LDY  #MCND  | ADD  1,U
LDX  #MLTRR  | STD  1,U
SYNC  | BCD  SKP19
LDD  SAVE  | INC  ,U
**

OVER  STD  T15  | LDA  ,Y
SYNC  | MUL
ADD  P15  | ADD  ,U
BCS  SKP12  | STD  ,U
LDA 1, U
LDR #15
MUL
ADD 2, U
BIC SKP20
CMPD #65521
BLO SKP21
ADD 15
SKP20
SKP21
LDA #15
LDX #65521
CLR 1, X
CLR 2, X
LDR #15
MUL
STO, X
LDA, X
LDR #15
MUL
ADD 1, X
ADD 2, U
BIC SKP22
CMPD #65521
BLO SKP23
ADD 15
SKP22
SKP23
SYNC
LDR 17
BIC SKP24
ADD #65521
SYNC
STD T7
SYNC
STD SAVE
LDR R7
SUBD SAVE
BIC SKP26
ADD #65521
SYNC
ADD R5
BIC SKP28
CMPD #65521
BLO SKP29
ADD 15
SKP28
SKP29
SYNC
ADD 15
R15
BIC SKP20
CMPD #65521
BLO SKP31
BLU
Appendix-D

```
LDA ,U          | STD  SAVE
LDX #TEMP      | LBRA  NEXT
CLR ,X          | *
CLR 1,X         | MLTR  FDS  36917
CLR 2,X         | MLTRR FDB  28122
LDN #15         | *
MUL ,X          | ORG  $0000
STD ,X          | MCND  FDB  0
LDA ,X          | P3002  FCB  0
LDN #15         | P3003  FCB  0
MUL ,X          | P3004  FCB  0
ADD 1,X         | P3005  FCB  0
ADD 2,U         | TEMP  FCB  0
RCS LOP22       | TEMP1 FCB  0
CMPD #65521     | TEMP2 FCB  0
BLD LOP23       | SAVE  FDB  0
LUP22 ADD #15   | FLAG  FCB  0
LUP23 STD T17   | REG  FDB  0
SYNC            | *
SYNC            | ORG  $FFFF
SYNC            | START EQU $F000
SYNC            | END  BEGIN
LDN R17         | *
```

**PROCESSOR NUMBER 11**

```
NAM 630911       | BRA  OVER
OUTPUT EQU $0400 | NEXT  LUY  #MCND
STATUS EQU $0402 | LDX  #MLTRR
T6  EQU $0403    | Sync
T12 EQU $0405   | LDN  SAVE
INPUT EQU $0410  | *
R6  EQU $0412    | OVER  STD  T6
R12 EQU $0414   | Sync
SEM  EQU $0416   | STD  SAVE

ORG  $F800       | SUBD  SAVE
NOP               | BCC  SKP12
ORCC #01010000   | ADDD #65521
LDU #P001        | SKP12  SYNC
BEGIN             | SYN
CLA              | SYN
STA               | SYN
LDA SEM           | SYN
SEQ FRD           | ADDD #12
START             | RCS  SKP14
LDA #1           | MCND  SKP14
STA FLAG          | MCND  SKP15
FRD               | LDY  #MCND  SKP14 ADDD #15
LDA #1           | *
STA STATUS        | SKP15  STD  MCND
SYNC              | CLR  ,U
CLA               | CLR  1,U
STA STATUS        | LDA  1,X
LDD INPUT        | LDB  1,Y
```

MUL 2, U  |  SYNC
LDA  , X  |  SYNC
LDB  , Y  |  STD  T6
MUL  |  |  STD  SAVE
ADD 1, U  |  LDD  R6
STD 1, U  |  SUBD  SAVE
BCC  SKP16  |  BCC  SKP24
INC  , U  |  ADDD #65521
SKP16
LDA 1, X  |  SKP24  STD  SAVE
LDB  , Y  |  LDA  FLAG
MUL  |  CMPA #1
ADD 1, U  |  BEQ  MULT
STD 1, U  |  BEQ  CONV
BCC  SKP19  |  INC  , U
INC  , U  |  STD  RES
SKP19
LDA  , X  |  LDD  SAVE
LDB  , Y  |  LOBRA  BEGIN
MUL  |  LDB  SAVE
ADD 1, U  |  STD  OUTPUT
STD  , U  |  LOBRA  BEGIN
SKP20
LDA 1, U  |  MUL  ADD 15
LDB #15  |  INC  FLAG
MUL  |  LDX #SAVE
ADD 2, U  |  LDY #RES
BCC  SKP20  |  LDP15  CLR  , U
CMPD #65521  |  CLR 1, U
BLO  SKP21  |  LDA 1, X
SKP21
ADD 15  |  LDB 1, Y
STD 2, U  |  MUL  LDA 1, U
SKP22
LDA  , X  |  LDB 1, Y
LDX #TEMP  |  LDB 1, Y
CLR  , X  |  MUL  LDB 1, Y
CLR 1, X  |  ADD 1, U
CLP 2, X  |  STD 1, U
LDB 15  |  BCC  LDP16
MUL  |  INC  , U
STD  , X  |  LDP15  LDA 1, X
LDA  , X  |  LDB  , Y
LDB 15  |  MUL  LDA 1, U
MUL  |  ADD 1, U
ADD 1, X  |  STD 1, U
ADD 2, U  |  BCC  LDP19
BCC  SKP22  |  INC  , U
CMPD #65521  |  LDP19  LDA 1, X
BLO  SKP23  |  LDB , Y
SKP23
ADD 15  |  MUL  LDA 1, U
SYNC  |  ADDD  , U
SYNC  |  STD  , U
SKP22
STD  T12  |  LDA 1, U
SYNC  |  LDB #15
FRD

LDY #MCND  | LDB ,Y
LDX #MLTRR  | MUL
LDA #1     | ADDD 1,U
STA STATUS | STD 1,Y
SYNC       | BCC SKP21
CLPA       | INC 1,U
STA STATUS | SKP21 LDA ,X
LDD INPUT  | LDB ,Y
BRA OVER   | MUL
LDY #MCND  | ADDD 1,U
LDX #MLTRR | STD 1,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 1,U
SYNC     | LDY #TEMP
ADDD R15 | CLR 1,X
BCS SKP14 | CLP 1,X
CMPD #65521 | CLR 2,Y
BLO SKP15 | LDR #15

SKP14

ADDD #15 | MUL
SKP15

STD T13  | STD 1,X
SYNC     | LDA 1,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

SKP18

LDY 1,X  | SKP24 ADDD #15

SYNC     | SKP25 SYN
CLR 1,U  | *
CLR 1,U  | SYNC
LDA 1,X  | ADDD R11
LDB 1,Y  | BCS SKP26
MUL      | CMPD #65521
STD 2,U  | BLO SKP27
LDA 1,X  | SKP26 ADDD #15
LDR 1,Y  | SKP27 STD #13
MUL      | SYN
ADDD 1,U | ADDD R13
STD 1,U  | BCS SKP28
BCC SKP18 | CMPD #65521
INC 1,U  | BLO SKP29
SKP18

LDA 1,X  | SKP28 ADDD #15
Appendix-D

SKP29
STD T15 | MUL
SYNC | ADDD 1,U
ADDD R15 | STD 1,U
BCS SKP30 | *
CMPO #65521 | LDA 1,U
SLO SKP31 | LDB #15
SKP30
ADDD #15 | MUL
SKP31
SYNC | ADDD 2,U
STD T7 | BCS LOP20
SYNC | CMPO #65521
STD SAVE | BLO LOP21
LDD R7 | LOP20 ADDD #15
SUBD SAVE | LOP21 STD 2,U
BCC SKP32 | *
ADDD #65521 | LDA 1,U
| LDX #TEMP
SKP32
STD SAVE | CLR ,X
LDA FLAG | CLP 1,X
CMPA #1 | CLR 2,X
BEQ MULT | LOB #15
CMPA #2 | MUL
BEQ CONV | STD ,X
LDD SAVE | LDA ,X
STD RES | LDB #15
LBRA BEGIN | MUL
CONV
LDD SAVE | ADDD 1,X
STD OUTPUT | ADDD 2,U
LBRA BEGIN | BCS LOP22
| CMPO #65521
* MULT
INC FLAG | BLO LOP23
LDX #SAVE | LOP22 ADDD #15
LDY #RES | LOP23 STD T7
LOP15
CLR 1,U | SYNC
CLR 1,Y | LDD R7
LDA 1,X | STD T13
LDB 1,Y | SYNC
MUL 2,U | LDD R13
STD 1,X | STD T7
LDA 1,Y | SYNC
LDB 2,U | LDD R7
MUL 1,U | STD SAVE
ADD 1,U | SYNC
STD 1,U | LBRA NEXT
BCC LOP16 | *
INC 1,U | MLTFR FDB 29032
INC LOP16
LDA 1,X | MLTFR FDB 28641
LDB 1,Y | *
MUL 1,U | ORG $0000
ADD 1,U | MCND FDB 0
STD 1,U | PROD1 FCB 0
BCC LOP19 | PROD2 FCB 0
INC 1,U | PROD3 FCB 0
LUP19
LDA 1,X | PROD4 FCB 0
LDB 1,Y | TEMP FCB 0
**Appendix-D**

| TEMP1  | FCB  | 0    | | TEMP3  | FCB  | 0    | | SAVE   | FDB  | 0    | | FLAG   | FCB  | 0    | | RES    | FDB  | 0    | | ORG    | $FFFE |
|--------|------|------| | STRT   | EQU  | $F800 |
| END    | BEGIN |
| IMS     | 80   | 913  | | AOODD  | #14  |
| ACS    | SKP14 |
| CMPD   | #65521 |
| SLO    | SKP15 |
| SKP14  | ADDD  | #15  |
| SKP15  | STD   | T12  |
| SYNC   |       |
| STD    | SAVE  |
| LDD    | R12  |
| SUBD   | SAVE  |
| BCC    | SKP16 |
| AOODD  | #65521 |
| ORG    | $F800 |
| SYNC   |       |
| STD    | MCND  |
| CLR    |       |
| CLR    |       |
| STD    | 1, U |
| LDA    |       |
| STD    | 1, U |
| LDA    |       |
| BCC    | SKP18 |
| INC    |       |
| INC    |       |
| LDA    |       |
| AOODD  | #1_U |
| AOODD  | #1_U |
| LDD    |       |
| MUL    |       |
| MUL    |       |
| STD    |       |
| BCC    | SKP21 |
| INC    |       |
| AOODD  | #1_U |
| AOODD  | #1_U |
| STD    |       |
| INC    |       |
| LDA    |       |
| LDD    | #Y   |
| MUL    |       |
| STD    | T8    |
| LDD    |       |
| STD    | SAVE  |
| AOODD  | #U   |
| STD    | R8    |
| STD    |       |
| LDD    | #U   |
| SUBD   | SAVE  |
| BCC    | SKP12 |
| LDA    |       |
| LDD    |       |
| AOODD  | #65521 |
| LDD    | #15  |
| MUL    |       |
| TEMP1  | FCB  | 0    | | TEMP3  | FCB  | 0    | | SAVE   | FDB  | 0    | | FLAG   | FCB  | 0    | | RES    | FDB  | 0    | | ORG    | $FFFE |
| STRT   | EQU  | $F800 |
| END    | BEGIN |
| IMS     | 80   | 913  | | AOODD  | #14  |
| ACS    | SKP14 |
| CMPD   | #65521 |
| SLO    | SKP15 |
| SKP14  | ADDD  | #15  |
| SKP15  | STD   | T12  |
| SYNC   |       |
| STD    | SAVE  |
| LDD    | R12  |
| SUBD   | SAVE  |
| BCC    | SKP16 |
| AOODD  | #65521 |
| ORG    | $F800 |
| SYNC   |       |
| STD    | MCND  |
| CLR    |       |
| CLR    |       |
| STD    | 1, U |
| LDA    |       |
| STD    | 1, U |
| LDA    |       |
| BCC    | SKP18 |
| INC    |       |
| INC    |       |
| LDA    |       |
| AOODD  | #1_U |
| AOODD  | #1_U |
| LDD    |       |
| MUL    |       |
| MUL    |       |
| STD    |       |
| BCC    | SKP21 |
| INC    |       |
| AOODD  | #1_U |
| AOODD  | #1_U |
| STD    |       |
| INC    |       |
| LDA    |       |
| LDD    | #Y   |
| MUL    |       |
| STD    | T8    |
| LDD    |       |
| STD    | SAVE  |
| AOODD  | #U   |
| STD    | R8    |
| STD    |       |
| LDD    | #U   |
| SUBD   | SAVE  |
| BCC    | SKP12 |
| LDA    |       |
| LDD    |       |
| AOODD  | #65521 |
| LDD    | #15  |
| MUL    |       |
Appendix-D

| CMPD #65521 | LDD SAVE |
| BLO SKP23 | STD RES |
| ADD #15 | LBPA BEGIN |
| STD 2,U | CONV LDD SAVE |
| I | STD OUTPUT |
| LDA ,U | LBRA BEGIN |
| LDX #TEMP | |
| CLR ,X | MULT INC FLAG |
| CLR 1,X | LDX #SAVE |
| CLR 2,X | LDY #RES |
| LDB #15 | LOP15 CLR ,U |
| MUL | CLR 1,U |
| STD ,X | LDA 1,X |
| LDA ,X | LDB 1,Y |
| LDB #15 | MUL |
| MUL | STD 2,U |
| ADD 1,X | LDA ,X |
| ADD 2,U | LDB 1,Y |
| BCS SKP24 | MUL |
| CMPD #65521 | ADDD 1,U |
| BLO SKP25 | STD 1,U |
| ADDD #15 | BCC LOP16 |
| SKP24 | SYNC |
| | LOP16 LDA 1,X |
| STD T12 | LDB ,Y |
| SYNC | MUL |
| STD SAVE | STD 1,U |
| LDD R12 | BCC LOP19 |
| SUBD SAVE | INC ,U |
| BCC SKP26 | LOP19 LDA ,X |
| ADDD #65521 | LDB ,Y |
| SKP26 | SYNC |
| STD T14 | MUL |
| ADDD R14 | ADDD 1,U |
| SYNC | STD ,U |
| BCS SKP28 | |
| CMPD #65521 | LDA 1,U |
| BLO SKP29 | LDB #15 |
| ADDD #15 | MUL |
| SKP28 | SYNC |
| ADDD 2,U | ADDD 2,U |
| SKP29 | BCS LOP20 |
| STD T8 | MUL |
| SYNC | CMPD #65521 |
| STD SAVE | BLO LOP21 |
| LDD R8 | LOP20 ADDD #15 |
| SUBD SAVE | LOP21 STD 2,U |
| BCC SKP30 | |
| ADDD #65521 | LDA ,U |
| SKP30 | STD #TEMP |
| STD SAVE | CLR ,X |
| LDA FLAG | CLR 1,X |
| CMPA #1 | CLR 2,X |
| REQ MULT | LDB #15 |
| CMPA #2 | MUL |
| REQ CONV | STD ,X |
LDA ,X
LDB #15
MUL
ADD 1,X
ADD 2,Y
BCS LOP22
CMPD #65521
BLD LOP23
ADD #15
BLD LOP22
STP T8
SYNC
LDD R8
STD T14
LDD R14
STD T12
SYNC
LDD R12
STD SAVE
SYNC
SYNC
STD SAVE ORG $FFF0
SYNC
LDD R12
STD SAVE ORG $FFF0
SYNC
SYNC
LBRA NEXT

* "___________________________________________________________________________

* "______________________________________________________________  ____________

"___________________________________________________________________________

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| SYNC
INPUT EQU $0410| LDY OVER STD T9
R9 EQU $0412| SYNC
R13 EQU $0414| STD SAVE
R18 EQU $0416| LDD P9
SEM EQU $0418| SUBD SAVE
* ORG $F800| ADD #65521
NOP| SKP12 SYNC
ORCC #01010000| STD T13
LDA #PRD1| SYNC
BEGIN CLR| SUBD R13
STA FLAG| BCC SKP14
LDA SEM| ADD #65521
BEQ FRO| SKP14 STD T1R
START LDA #1| SYNC
STA FLAG| SYNC
FRD LDA #MCND| SYNC
LDX #MLTRR| STD MCND
LDA #1| CLR ,U
STA STATUS| CLR 1,Y
SYNC LDB INPUT| STD 2,Y
<table>
<thead>
<tr>
<th>LOA</th>
<th>,X</th>
<th>SKP24</th>
<th>ADDD #15</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDB</td>
<td>,Y</td>
<td>SKP25</td>
<td>SYNC</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>1,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC</td>
<td>SKP16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOA</td>
<td>,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC</td>
<td>SKP19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOA</td>
<td>,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC</td>
<td>SKP20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td>STD</td>
</tr>
<tr>
<td>BLD</td>
<td>SKP21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>#15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>2,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>#TFMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>1,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>2,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOB</td>
<td>#15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOA</td>
<td>,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>#15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>1,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>2,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCS</td>
<td>SKP22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLD</td>
<td>SKP23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>#15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOB</td>
<td>#15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>1,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCS</td>
<td>SKP24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLD</td>
<td>SKP25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix-D

**LUP19**

```
LDA ,X  | LDP23  | STD  | T13
LDB ,Y  |       | SYNC |
MUL     |       | SYNC |
ADDU    |       | LDO  | R13
STD     |       | STD  | T18
```

**LUP20**

```
ADDU=15  |       | LDO  | R9
MUL      |       | STD  | SAVE
BCS LOP20|       | SYNC |
CMPD #65521|       | LBRA | NEXT
```

**LUP21**

```
STD 2,U  |       | MLTRR FDB 21938
```

**LUP22**

```
ADDU=15  |       |
```

---

**BEGIN**

```
ORG $F800  | STA STATUS
```

**OUTPUT**

```
NAM 680915  | STA FLAG
```

**STATUS**

```
EQU $0400  | LDA SEM
EQU $0402  | BEQ FRO
```

**T10**

```
EQU $0403  | START LDA #1
```

**T12**

```
EQU $0405  | STA Flag
```

**T18**

```
EQU $0407  | FRD LDX #MCND
```

**INPUT**

```
EQU $0410  | LDX #MLTRR
```

**R10**

```
EQU $0412  | STA STATUS
```

**R12**

```
EQU $0414  | STA STATUS
```

**R18**

```
EQU $0416  | SYNC
```

**SEM**

```
EQU $0418  | CLRA
```

**ORG**

```
ORG $F800  | LDO INPUT
```

**NEXT**

```
ORG $F800  | LDO INPUT
```

**PROCEDURE**

```
PROCEDURE 15
```

---

**BEGIN**

```
CLRA
```

---

**END**
Appendix-D

* OVER
  STD T10
  SYNC
  STD SAVE
  LDD R10
  SUBD SAVE
  BCC SKP12
  ADDD #65521
  SYNC
  STD T12
  SYNC
  STD SAVE
  LDD R12
  SUBD SAVE
  BCC SKP14
  ADDD #65521
  SKP12
  STD T12
  STD SAVE
  LDD R12
  ADDD 2,U
  SUBD SAVE
  BCC SKP14
  ADDD #65521
  SKP14
  STD T18
  SYNC
  STD SAVE
  LDD R12
  ADDD 2,U
  SUBD SAVE
  BCC SKP14
  ADDD #65521
  SKP16
  STD T10
  SYNC
  STD MCND
  CLR i,U
  CLR 1,U
  LDA 1,X
  LDB 1,Y
  MUL
  STD 2,U
  LDA ,X
  LOB ,Y
  MUL
  ADDD 1,U
  STD 1,U
  BCC SKP16
  INC ,U
  LDA 1,X
  LOB ,Y
  MUL
  ADDD 1,U
  STD 1,U
  BCC SKP19
  INC ,U
  LDA ,X
  LOB ,Y
  MUL
  ADDD ,U
  STD ,U
  CMPA #1
  ADD #15
  MUL
  ADDD 2,U
  BCS SKP20
  CMPD #65521
  BLO SKP21
  CMP
  STD SAVE
  STD FLAG
  CMPA #2
  BEQ MULT
  CMPA #1
  BEQ CONV
  LDD SAVE
  STD RES
  LDBA BEGIN
  CONV
  LDD SAVE
  STD OUTPUT
<table>
<thead>
<tr>
<th>LBRA</th>
<th>BEGIN</th>
<th>CLR</th>
<th>,X</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC</td>
<td>FLAG</td>
<td>CLR</td>
<td>1,X</td>
</tr>
<tr>
<td>LDX</td>
<td>#SAVE</td>
<td>LDA</td>
<td>#15</td>
</tr>
<tr>
<td>LDY</td>
<td>#RES</td>
<td>MUL</td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>,U</td>
<td>STD</td>
<td>,X</td>
</tr>
<tr>
<td>CLR</td>
<td>1,U</td>
<td>LDA</td>
<td>,X</td>
</tr>
<tr>
<td>LDA</td>
<td>1,X</td>
<td>LDA</td>
<td>#15</td>
</tr>
<tr>
<td>LDB</td>
<td>1,Y</td>
<td>MUL</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>2,U</td>
<td>ADDD</td>
<td>1,X</td>
</tr>
<tr>
<td>LDA</td>
<td>,X</td>
<td>ADDD</td>
<td>2,U</td>
</tr>
<tr>
<td>LDB</td>
<td>1,Y</td>
<td>EGS</td>
<td>LOP22</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td>CMPD</td>
<td>#65521</td>
</tr>
<tr>
<td>ADDD</td>
<td>1,U</td>
<td>LOP22</td>
<td>ADD</td>
</tr>
<tr>
<td>STD</td>
<td>1,U</td>
<td>LOP23</td>
<td>STD</td>
</tr>
<tr>
<td>INC</td>
<td>,U</td>
<td>LOP16</td>
<td>SYNC</td>
</tr>
<tr>
<td>LDA</td>
<td>1,X</td>
<td>LOP16</td>
<td>SYNC</td>
</tr>
<tr>
<td>LDB</td>
<td>,Y</td>
<td>LOP16</td>
<td>SYNC</td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td>LOP19</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>1,U</td>
<td>LOP19</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>1,U</td>
<td>LOP19</td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>1,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>1,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDB</td>
<td>,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>1,U</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PROCESSOR NUMBER 16**

<table>
<thead>
<tr>
<th>NAM</th>
<th>680916</th>
</tr>
</thead>
<tbody>
<tr>
<td>T4</td>
<td>EQU</td>
</tr>
<tr>
<td>T5</td>
<td>EQU</td>
</tr>
<tr>
<td>R4</td>
<td>EQU</td>
</tr>
<tr>
<td>R5</td>
<td>EQU</td>
</tr>
<tr>
<td>SEM</td>
<td>EQU</td>
</tr>
<tr>
<td>ORG</td>
<td>$F800</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>ORCC</td>
<td>#01010000</td>
</tr>
<tr>
<td>LDU</td>
<td>#PROD1</td>
</tr>
<tr>
<td>BEGIN</td>
<td>CLRA</td>
</tr>
<tr>
<td>STA</td>
<td>FLAG</td>
</tr>
<tr>
<td>LDA</td>
<td>SEM</td>
</tr>
<tr>
<td>BEQ</td>
<td>FRO</td>
</tr>
<tr>
<td>START</td>
<td>LDA</td>
</tr>
</tbody>
</table>
FRD
LDY #MCND
LDX #MLTRR
BRA OVER

NEXT
LDY #MCND
LDX #MLTRR

OVER
SYNC
SYNC
SYNC
SYNC
LDD R4
ADDD R5

BLO SKP12
CMPO #65521

BLO SKP13
ADDD #15

SYNC

BLO SKP12

ADDD #15

STD MCND

LDA 1,X

LDA 1,Y
MUL
STD 2,Y
LDA 1,Y
LDR 1,Y
MUL

ADDD 1,Y

STD 1,Y
BCC SKP17
INC ",,U

LDA 1,X

LDA 1,Y
MUL
ADDD 1,Y

STD 1,Y
BCC SKP17
INC ",,U

LDA 1,X

LDA 1,Y
MUL
ADDD 1,Y

STD 1,Y
BCC SKP17
INC ",,U

LDA 1,X

LDA 1,Y
MUL

ADD 2,Y

BCS SKP18
CMPO #65521

BLO SKP19

ADDD #15

`
Appendix-0

FLAG FCB 0 | ORG $FFFF
* | STRT EQU $F800
* | END BEGIN
* | *****************************************************
* | PROCESSOR NUMBER 17
* | *****************************************************

NAM 680917 | INC ,U
T9 EQU $0410 | LDA 1,X
T10 EQU $0412 | LDR ,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
| LDL #PRD1 | MUL
BEgin CLRA | ADDD ,U
| STA FLAG | STD ,U
| LDA SEM | *
| BEQ FRD | LDA 1,U
STAPT LDA #1 | LDB #15
| STA FLAG | MUL
| FKR LDY #MCND | ADDD 2,U
| LDX #MLTFR | BCS SKP18
| BRA OVER | CMPD #65521
NEXT LDY #MCND | BLD 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
| BLD SKP13 | STD ,X
| SKP12 | LDA ,X
| ADDD #15 | LDB #15
SKP13 | MUL
SYNC | A
| | STD MCND | ADDD 1,X
| | CLR ,U | ADDD 2,U
| | CLR 1,X | BCS SKP20
| | LDA 1,X | CMPD #65521
| | LDB 1,Y | BLO SKP21
| | MUL | SKP20 ADDD #15
| | STD 2,U | SKP21 SYNC
| | STD ,X | *
| | LDB 1,Y | STD #10
| | MUL | STD #9
| | ADDD 1,U | SYNC
| | STD 1,U | SYNC
| | BCC SKP14 | SYNC
<table>
<thead>
<tr>
<th>SYNC</th>
<th>MLTFR FDB 34561</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>FLAG</td>
<td>CRG $0000</td>
</tr>
<tr>
<td>CMPA</td>
<td>#1</td>
<td>MCND FDB 0</td>
</tr>
<tr>
<td>BEQ</td>
<td>SKP</td>
<td>IPPD1 FCB 0</td>
</tr>
<tr>
<td>LBRA</td>
<td>BEGIN</td>
<td>IPPD2 FCB 0</td>
</tr>
<tr>
<td>INC</td>
<td>FLAG</td>
<td>IPPD3 FCB 0</td>
</tr>
<tr>
<td>SYNC</td>
<td>#1</td>
<td>IPPD4 FCB 0</td>
</tr>
<tr>
<td>LDD</td>
<td>R10</td>
<td>TEMP FCB 0</td>
</tr>
<tr>
<td>STD</td>
<td>T9</td>
<td>TEMP1 FCB 0</td>
</tr>
<tr>
<td>SYNC</td>
<td>#1</td>
<td>TFMP3 FCB 0</td>
</tr>
<tr>
<td>SYNC</td>
<td>#1</td>
<td>SAVE FDB 0</td>
</tr>
<tr>
<td>LDD</td>
<td>R9</td>
<td>FLAG FCB 0</td>
</tr>
<tr>
<td>STD</td>
<td>T10</td>
<td>LOA 0</td>
</tr>
<tr>
<td>LBRA</td>
<td>NEXT</td>
<td>ORG $FFE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>END BEGIN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>* MLTFR FDB 5753 *</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
</tr>
<tr>
<td>**</td>
</tr>
<tr>
<td>** PROCESSOR NUMBER 18 **</td>
</tr>
<tr>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAM 680918</th>
<th>SKP13 SYNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>T14 EQU $0410</td>
<td>*</td>
</tr>
<tr>
<td>T15 EQU $0412</td>
<td>STD MCND</td>
</tr>
<tr>
<td>R14 EQU $0414</td>
<td>CLR R14</td>
</tr>
<tr>
<td>R15 EQU $0416</td>
<td>CLR 1, U</td>
</tr>
<tr>
<td>SEM EQU $0418</td>
<td>LDA 1, X</td>
</tr>
<tr>
<td>*</td>
<td>LDB 1, Y</td>
</tr>
<tr>
<td>ORG $F900</td>
<td>MUL</td>
</tr>
<tr>
<td>NOP</td>
<td>STD 2, U</td>
</tr>
<tr>
<td>ORCC %01010000</td>
<td>LDA 1, X</td>
</tr>
<tr>
<td>LOU #PROD1</td>
<td>LDB 1, Y</td>
</tr>
<tr>
<td>BEGIN CLRA</td>
<td></td>
</tr>
<tr>
<td>STA FLAG</td>
<td>ADDD 1, U</td>
</tr>
<tr>
<td>LDA SEM</td>
<td>STD 1, U</td>
</tr>
<tr>
<td>BEQ FRD</td>
<td>BCC SKP14</td>
</tr>
<tr>
<td>START LDA #1</td>
<td>INC</td>
</tr>
<tr>
<td>STA FLAG</td>
<td>SkP14 LDA 1, X</td>
</tr>
<tr>
<td>FKD LDY #MCND</td>
<td>LDB</td>
</tr>
<tr>
<td>LDX #MLTFR</td>
<td>MUL</td>
</tr>
<tr>
<td>BRA OVER</td>
<td>ADDD 1, U</td>
</tr>
<tr>
<td>NEXT LDY #MCND</td>
<td>STD 1, U</td>
</tr>
<tr>
<td>LDX #MLTFR</td>
<td>BCC SKP17</td>
</tr>
<tr>
<td>OVER SYNC</td>
<td>INC</td>
</tr>
<tr>
<td>SYNC</td>
<td>SKP17 LDA 1, X</td>
</tr>
<tr>
<td>SYNC</td>
<td>LDB 1, Y</td>
</tr>
<tr>
<td>SYNC</td>
<td>MUL</td>
</tr>
<tr>
<td>SYNC</td>
<td>ADDD 1, U</td>
</tr>
<tr>
<td>LDD R14</td>
<td>STD 1, U</td>
</tr>
<tr>
<td>ADDO P15</td>
<td>*</td>
</tr>
<tr>
<td>BCS SKP12</td>
<td>LDA 1, U</td>
</tr>
<tr>
<td>CMPD #65521</td>
<td>LDB #15</td>
</tr>
<tr>
<td>BLO SKP13</td>
<td>MUL</td>
</tr>
<tr>
<td>SKP12 ADDO #15</td>
<td>ADDD 2, U</td>
</tr>
</tbody>
</table>
Appendix-D

**CONTROL MICROPROCESSOR**

<table>
<thead>
<tr>
<th>NAME</th>
<th>CONTROL</th>
<th>ORG</th>
<th>START</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACIACR</td>
<td>EQU $1040</td>
<td>ORCC</td>
<td>NOP</td>
</tr>
<tr>
<td>ACIASR</td>
<td>EQU $1040</td>
<td>LDS</td>
<td>$20</td>
</tr>
<tr>
<td>ACIARX</td>
<td>EQU $1041</td>
<td>CLRA</td>
<td></td>
</tr>
<tr>
<td>ACIATX</td>
<td>EQU $1041</td>
<td>STA</td>
<td>CONTRL</td>
</tr>
<tr>
<td>ARYOUT</td>
<td>EQU $1020</td>
<td>STA</td>
<td>STATUS</td>
</tr>
<tr>
<td>INPUT</td>
<td>EQU $1000</td>
<td>LDA</td>
<td>#00001011</td>
</tr>
<tr>
<td>OUT1M</td>
<td>EQU $1050</td>
<td>STA</td>
<td>CONTRL</td>
</tr>
<tr>
<td>OUT1L</td>
<td>EQU $1051</td>
<td>LDA</td>
<td>#13</td>
</tr>
<tr>
<td>OUT2M</td>
<td>EQU $1052</td>
<td>STA</td>
<td>ACIACR</td>
</tr>
<tr>
<td>OUT2L</td>
<td>EQU $1053</td>
<td>LDA</td>
<td>#11</td>
</tr>
<tr>
<td>DATA</td>
<td>EQU $1054</td>
<td>STA</td>
<td>ACIACR</td>
</tr>
<tr>
<td>CONTRL</td>
<td>EQU $101E</td>
<td>LDA</td>
<td>#00000111</td>
</tr>
<tr>
<td>LDA</td>
<td>$000000110</td>
<td>STA</td>
<td>TMP</td>
</tr>
</tbody>
</table>
Appendix-D

STA CONTRL | JSR RCX
LOA #000000111 | JSR TXR
STA CONTRL | CMPA #$0D
STA STATUS | BEQ SK5
BRA BEGIN | JSR VALID

INIT
LOD #$80 | ADDA TMP
LOA #$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 EMSG1

* SKP1
LOA #1 | LOOP5 JSR CRLF
STA FLAG | WRITE LDA CNT
LDX #MSG3 | LSPA
LOA #15 | LSPA
STA CNT | LSPA
JSR DSPLY | LSRA
LDY #INI | JSR CONVA
LDX #ARYIN | JSR TXR
JSR EXG | LDA CNT
BRA MODFY | ANDA #$0F

SKP2
LOA #2 | JSR CONVA
STA FLAG | JSR TXR
LDY #IN2 | LDA #7=
LDX #ARYIN | JSR TXR
JSR EXG | LDA #$20
LDX #MSG8 | JSR TXR
LOA #13 | LDB CNT
STA CNT | LDA $x
JSR DSPLY | LSRA
JSR CPLF | LSRA
JSR PFX | LSRA

* MODFY
JSR CPLF | JSR CONVA
LDX #ARYIN | JSR TXR
LOA #$20 | LDB CNT
JSR TXR | LDA $x
JSR RCX | ANDA #$0F
JSR TXP | 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  #"-"    | ANDA  #%111111110
BEQ   DECR    | STA    CONTRL
CMPA  #20     | DRA    #00001001
BEQ   INCR    | STA    CONTRL
JSR   VALID   | SYNC
LSLA  | ANDA  #%11110111
LSLA  | STA    CONTRL
LSLA  | STA    STATUS
LSLA  | STA    STATUS
LSLA  | STA    STATUS
LSLA  | STA    STATUS
STA   TMP     | CONV    LDX   #OUT
JSR   RCX     | JSR    EXCHG.
JSR   TXR     | JSR    CRLF
JSR   VALID   | LDX    #MSG4
ADDA  TMP     | LDA    #11
LDA   CNT     | STA    CNT
STA   B,X     | JSP    DSPLY
LDA   CNT     | JSP    CRLF
LDA   CNT     | LDX    #OUT
DECRA  #30    | JSP    ARAY
BHS   MOVE    | JSP    CRLF
STA   CNT     | SKP4   LDA    #15
BRA   LOOP5   | STA    CNT
DECR  CNT     | LDX    #OUT
DECA  | SKP5   LDD    ,X++
BLT   MOVE    | STD    OUT2M
STA   CNT     | DEC    CNT
LBRA  LOOP5   | BNE    SKP5
MOV   LSRA    | LDA    ACIASR
JSR   CPLF    | LDA    ACIASR
JSR   PFX     | BCC    SKP4
JSR   CRLF    | LDA    ACIASR
LDY   #ARYIN  | ANDA  #$7F
LDA   FLAG    | CMPA  #$7
CMPA  #1      | LBEQ   GET
BNE   SKP3    | LBRA   BEGIN
LDA   STATUS  | SKP3   LDA    ,X++
ANDA  #%10111111 | DSPLY   LDA    ,X++
STA   CONTRL  | JSR    TXR
STA   STATUS  | DEC    CNT
LDA   #INIT   | BNE    DSPLY
JSR   EXG     | RTS
LDA   #INPUT  | TXR    LDB    #$02
JSR   EXG     | RTS
LBRA  BEGIN   | WAIT   LDB    #$02
LDA   STATUS  | BEQ    WAIT
ORA   #01000000 | STA    ACIATX
STA   CONTRL  | RTS    RETURN
STA   STATUS  | ST
LDA   #IN2    | RCX    LDA    ACIASR
JSR   EXG     | RTS
LDA   #INPUT  | BCC    RCX
JSR   EXG     | RTS
STA   STATUS  | ST
RTS

Note: The code appears to be a collection of assembly language instructions, possibly related to computer programming or control systems. The instructions are mixed and not in a conventional order, which makes it difficult to interpret without context. The code includes statements like `BEQ`, `JSR`, `LSLA`, `STA`, `RTS`, and other assembly language commands.
\* CONVA CMPA #9 | RTS
BLS OMIT | *
ADD A #A-#9-1 | GET JSR CRLF
OMIT ADDA #0 | JSR PFX
ANDA #76F | LDX #MSG4
RTS | LDA #12

\* CRLF LDA #30D | STA CNT
JSR TXR | JSR PFX
LDA #30A | LOOP W LDA STATUS
JSR TXR | ORA #00010000
RTS | STA STATUS

\* EXCHG LDA STATUS | STA CONTRL
ORA #%00001000 | LOOP X LDA #15
STA CONTRL | STA CNT
STA STATUS | LDD #ARYIN
LDY #ARYOUT | LDX #INPUT
SYNC | LDU #IN2

\* EXG LDD ,Y | LDD DATA
STD ,X | STD ,X++
LDD 2,Y | STD ,Y++
STD 3,X | DEC CNT
LDD 0,Y | BNE LOOP Y
STD 6,Y | *
LDD 8,Y | LDA STATUS
STD 8,X | ANDA #%11111110
LDD 0, Y | STA CONTRL
STD 10,Y | MRA #%00001001
STD 10, X | STA CONTRL
LDD 12,Y | SYNC
STD 12,X | ANDA #%11110111
LDD 14,Y | STA CONTRL
STD 14,X | STA STATUS
LDD 16,Y | *
STD 16,X | *
LDD 18,Y | LDX #OUT
STD 18,X | JSR EXCHG
LDD 20,Y | LDY #IN2
STD 20,X | LDA #15
LDD 22,Y | STA CNT
STD 22,X | LOOP Z LDD ,X++
LDD 24,Y | STD OUT2M
STD 24,X | STD ,Y++
LDD 26,Y | STD OUT1M
STD 26,X | DEC CNT
LDD 28,Y | NE LOOP Z
STD 28,X | LDA ACI A5P
ANDA #%11110111 | LSPA
STA CONTRL | BCC LOOP X
ANDA #7F | STD OUT2M
Appendix-D

<table>
<thead>
<tr>
<th>CMPA</th>
<th>#E</th>
<th>DEC</th>
<th>CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBEQ</td>
<td>BEGIN</td>
<td>BNE</td>
<td>LOPP</td>
</tr>
<tr>
<td>JSR</td>
<td>CRLF</td>
<td>LDA</td>
<td>ACIASR</td>
</tr>
<tr>
<td>JSR</td>
<td>PFX</td>
<td>LSRA</td>
<td></td>
</tr>
<tr>
<td>LDX</td>
<td>#OUT</td>
<td>BCC</td>
<td>SK9</td>
</tr>
<tr>
<td>LDY</td>
<td>#IN2</td>
<td>LDA</td>
<td>ACIASR</td>
</tr>
<tr>
<td>LDA</td>
<td>#15</td>
<td>ANDA</td>
<td>#$7F</td>
</tr>
<tr>
<td>STA</td>
<td>CNT</td>
<td>CMPS</td>
<td>#1</td>
</tr>
<tr>
<td>LDD</td>
<td>,Y++</td>
<td>LBEQ</td>
<td>SKP1</td>
</tr>
<tr>
<td>STD</td>
<td>OUT1M</td>
<td>LBRA</td>
<td>SKP2</td>
</tr>
<tr>
<td>LDD</td>
<td>,X++</td>
<td>#*</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>OUT2M</td>
<td>LARAY</td>
<td>LDA</td>
</tr>
<tr>
<td>DEC</td>
<td>CNT</td>
<td>STA</td>
<td>CNT</td>
</tr>
<tr>
<td>BNE</td>
<td>LOOP</td>
<td>AGAIN</td>
<td>CLR</td>
</tr>
<tr>
<td>LDA</td>
<td>ACIASR</td>
<td>LOOP2</td>
<td>CLR</td>
</tr>
<tr>
<td>LSRA</td>
<td></td>
<td>LOOP1</td>
<td>LDA</td>
</tr>
<tr>
<td>LDA</td>
<td>ACIASR</td>
<td>INC</td>
<td>CNT2</td>
</tr>
<tr>
<td>ANDA</td>
<td>#$7F</td>
<td>LSRA</td>
<td></td>
</tr>
<tr>
<td>CMPA</td>
<td>#1</td>
<td>LSRA</td>
<td></td>
</tr>
<tr>
<td>LBEQ</td>
<td>SKP1</td>
<td>LSRA</td>
<td></td>
</tr>
<tr>
<td>CMPA</td>
<td>#2</td>
<td>JSR</td>
<td>CONVA</td>
</tr>
<tr>
<td>LBEQ</td>
<td>SKP2</td>
<td>JSR</td>
<td>TXR</td>
</tr>
<tr>
<td>LBRA</td>
<td>LOOPX</td>
<td>LDA</td>
<td>,X*</td>
</tr>
<tr>
<td>DSP</td>
<td>JSR</td>
<td>CRLF</td>
<td>JSR</td>
</tr>
<tr>
<td>LDX</td>
<td>#MSG5</td>
<td>JSR</td>
<td>TXR</td>
</tr>
<tr>
<td>LDA</td>
<td>#7</td>
<td>INC</td>
<td>CNT2</td>
</tr>
<tr>
<td>STA</td>
<td>CNT</td>
<td>DEC</td>
<td>CNT</td>
</tr>
<tr>
<td>JSR</td>
<td>DSPLY</td>
<td>BEQ</td>
<td>SET</td>
</tr>
<tr>
<td>JSR</td>
<td>CRLF</td>
<td>LDA</td>
<td>CNT2</td>
</tr>
<tr>
<td>JSR</td>
<td>PFX</td>
<td>CMPA</td>
<td>#4</td>
</tr>
<tr>
<td>JSR</td>
<td>CRLF</td>
<td>BEQ</td>
<td>CHK1</td>
</tr>
<tr>
<td>JSR</td>
<td>#IN1</td>
<td>BRA</td>
<td>LOOP1</td>
</tr>
<tr>
<td>JSR</td>
<td>ARAY</td>
<td>OVER</td>
<td>JSR</td>
</tr>
<tr>
<td>JSR</td>
<td>CRLF</td>
<td>BRA</td>
<td>AGAIN</td>
</tr>
<tr>
<td>JSR</td>
<td>CRLF</td>
<td>CHK1</td>
<td>LDA</td>
</tr>
<tr>
<td>LDX</td>
<td>#MSG6</td>
<td>CMPA</td>
<td>#4</td>
</tr>
<tr>
<td>LDA</td>
<td>#7</td>
<td>BEQ</td>
<td>OVER</td>
</tr>
<tr>
<td>STA</td>
<td>CNT</td>
<td>LDA</td>
<td>#$20</td>
</tr>
<tr>
<td>JSR</td>
<td>DSPLY</td>
<td>JSR</td>
<td>TXR</td>
</tr>
<tr>
<td>JSR</td>
<td>CRLF</td>
<td>INC</td>
<td>CNT1</td>
</tr>
<tr>
<td>LDX</td>
<td>#IN2</td>
<td>BRA</td>
<td>LOOP2</td>
</tr>
<tr>
<td>JSR</td>
<td>ARAY</td>
<td>SET</td>
<td>RTS</td>
</tr>
<tr>
<td>JSR</td>
<td>CRLF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR</td>
<td>PFX</td>
<td>VALID</td>
<td>SUBA</td>
</tr>
<tr>
<td>Sk9</td>
<td>#IN1</td>
<td>CMPA</td>
<td>#9</td>
</tr>
<tr>
<td>LDX</td>
<td>#IN2</td>
<td>BHI</td>
<td>CHK1</td>
</tr>
<tr>
<td>LDA</td>
<td>#15</td>
<td>PTS</td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>CNT</td>
<td>CHK1</td>
<td>SUBA</td>
</tr>
<tr>
<td>LOOP</td>
<td>LDD</td>
<td>,X++</td>
<td>CMPA</td>
</tr>
<tr>
<td>STD</td>
<td>OUT1M</td>
<td>PLS</td>
<td>OK</td>
</tr>
<tr>
<td>LDD</td>
<td>,Y++</td>
<td>RRA</td>
<td>ERMG3</td>
</tr>
<tr>
<td>OK</td>
<td>RTS</td>
<td>MSG1</td>
<td>FNC</td>
</tr>
</tbody>
</table>
Appendix-D

* PFX
  LDX #MSG7
  LDA #6
  STA CNT
  JSR DSPLY
  RTS
  JSR #MSG7
  JSR CRLF
  JSR OSPLY
  JSR CRLF
  LBRA HJIT

  ERMSG1
  JSR CRLF
  LDX #MSG1
  LOA #17
  STA CNT
  JSR CRLF
  JSR DSPLY
  JSR CRLF
  LBRA INIT

  ERMSG3
  JSR CRLF
  LDX #MSG2
  LOA #17
  STA CNT
  JSR CRLF
  JSR DSPLY
  JSR CRLF
  LBRA INIT

* NAM

** ******************************************************************************
* WINograd's 15 point algorithm
** ******************************************************************************

  LDX #AX
  LDY #ARYIN
  LDD 20,Y
  LDD 26,Y
  LDD 22,X
  LDD 24,Y
  LDD 3,Y
  LDD 10,Y
  STO 10,X
  LDD 16,Y
  STO 12,X
  LDD 22,Y
  STO 14,X
  LDD 28,Y
  STO 16,X
  LDD 4,Y

  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

  LDD 18,Y
  STD 6,X
  LDD 24,Y
  STD 3,X
  SKP2
  LDD 10,X
  ADDD 20,X
  RCS JMP1
  CMPD #65521
  BLO JMP2
  ADDD #15
  STD TMP1
  ADDD #X
  RCS JMP3
  CMPD #65521

? 'Invalid HEX Digit'
? 'Enter Response'
? 'Convolution'
? 'Array 1'
? 'Array 2'
? 'CONV: '
? 'Enter Values'

? 'Invalid HEX Digit'
? 'Enter Response'
? 'Convolution'
? 'Array 1'
? 'Array 2'
? 'CONV: '
? 'Enter Values'

? Org $0081

? Org $FFFF

? Equ $F800

? End Start
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Instruction</th>
<th>Address</th>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCS JMP29</td>
<td>#65521</td>
<td>JMP42</td>
<td>ADDD #15</td>
<td>STD</td>
<td>16,Y</td>
</tr>
<tr>
<td>CMPD</td>
<td>JMP43</td>
<td>STD</td>
<td>LDD</td>
<td>16,X</td>
<td></td>
</tr>
<tr>
<td>BLD JMP30</td>
<td></td>
<td>LDD</td>
<td>SUBD</td>
<td>14,X</td>
<td></td>
</tr>
<tr>
<td>ADDD #15</td>
<td></td>
<td>BCC</td>
<td>JMP44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD 4,Y</td>
<td></td>
<td>ADDD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDD 6,X</td>
<td></td>
<td>JMP44</td>
<td>STD</td>
<td>22,Y</td>
<td></td>
</tr>
<tr>
<td>SUBD 4,Y</td>
<td></td>
<td>ADDD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC JMP31</td>
<td></td>
<td>STD</td>
<td>LDD</td>
<td>18,Y</td>
<td></td>
</tr>
<tr>
<td>ADDD #65521</td>
<td></td>
<td>BCS</td>
<td>JMP45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD 10,Y</td>
<td></td>
<td>ADDD</td>
<td>#15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD 6,Y</td>
<td></td>
<td>JMP46</td>
<td>STD</td>
<td>20,Y</td>
<td></td>
</tr>
<tr>
<td>BCS JMP32</td>
<td></td>
<td>LDD</td>
<td>BCS</td>
<td>JMP47</td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td></td>
<td>ADDD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLO JMP33</td>
<td></td>
<td>JMP47</td>
<td>STD</td>
<td>TMP1</td>
<td></td>
</tr>
<tr>
<td>ADDD #15</td>
<td></td>
<td>STD</td>
<td>ADDD</td>
<td>14,Y</td>
<td></td>
</tr>
<tr>
<td>JMP34</td>
<td></td>
<td>BCS</td>
<td>JMP49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD #15</td>
<td></td>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD TMP1</td>
<td></td>
<td>BLO</td>
<td>JMP50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD ,X</td>
<td></td>
<td>BCS</td>
<td>JMP36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td></td>
<td>LDD</td>
<td>STD</td>
<td>12,Y</td>
<td></td>
</tr>
<tr>
<td>BLO JMP37</td>
<td></td>
<td>ADDD</td>
<td>14,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDD ,Y</td>
<td></td>
<td>STD</td>
<td>ADDD</td>
<td>16,Y</td>
<td></td>
</tr>
<tr>
<td>ADDD #15</td>
<td></td>
<td>BCC</td>
<td>JMP31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP36</td>
<td></td>
<td>ADDD</td>
<td>STD</td>
<td>16,Y</td>
<td></td>
</tr>
<tr>
<td>STD 4,Y</td>
<td></td>
<td>LDD</td>
<td>ADDD</td>
<td>22,X</td>
<td></td>
</tr>
<tr>
<td>LDD TMP1</td>
<td></td>
<td>STD</td>
<td>ADDD</td>
<td>28,X</td>
<td></td>
</tr>
<tr>
<td>STD 2,Y</td>
<td></td>
<td>LDD</td>
<td>BCS</td>
<td>JMP52</td>
<td></td>
</tr>
<tr>
<td>LDD 12,X</td>
<td></td>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD 18,X</td>
<td></td>
<td>BLD</td>
<td>JMP53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCS JMP39</td>
<td></td>
<td>JMP52</td>
<td>ADDD</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td></td>
<td>STD</td>
<td>26,Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLO JMP40</td>
<td></td>
<td>LDD</td>
<td>22,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD #15</td>
<td></td>
<td>SUBD</td>
<td>28,X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STD 14,Y</td>
<td></td>
<td>BCC</td>
<td>JMP54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDD 12,X</td>
<td></td>
<td>ADDD</td>
<td>STD</td>
<td>24,X</td>
<td></td>
</tr>
<tr>
<td>SUBD 18,X</td>
<td></td>
<td>LDD</td>
<td>ADDD</td>
<td>26,X</td>
<td></td>
</tr>
<tr>
<td>BCC JMP41</td>
<td></td>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD #65521</td>
<td></td>
<td>STD</td>
<td>ADDD</td>
<td>26,X</td>
<td></td>
</tr>
<tr>
<td>STD 18,Y</td>
<td></td>
<td>BCS</td>
<td>JMP56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDD 14,X</td>
<td></td>
<td>CMPD</td>
<td>#65521</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD 16,X</td>
<td></td>
<td>BLD</td>
<td>JMP57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCS JMP42</td>
<td></td>
<td>ADDD</td>
<td>STD</td>
<td>28,Y</td>
<td></td>
</tr>
<tr>
<td>CMPD #65521</td>
<td></td>
<td>STD</td>
<td>LDD</td>
<td>26,X</td>
<td></td>
</tr>
<tr>
<td>BLO JMP43</td>
<td></td>
<td>BLO</td>
<td>JMP43</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Indicates a special instruction or condition.
SUBD 24,X
BCC JMP58
ADDD #65521
JMP58
STD 34,Y
ADDD 30,Y
BCS JMP59
CMPD #65521
BLO JMP60
ADDD #15
JMP59
STD 32,Y
LDD 26,Y
ADDD 28,Y
BCS JMP61
CMPD #65521
BLO JMP62
ADDD #15
JMP61
STD TMP1
ADD 20,X
BCS JMP63
CMPD #65521
BLO JMP64
ADDD #15
JMP63
STD 24,Y
LDD 26,Y
SUBD 28,Y
BCC JMP65
ADDD #65521
JMP65
STD 28,Y
LDD TMP1
ADD 26,Y
SKIP6
ADD 32,Y
STO 34,Y
ADD 30,Y
STO 36,Y
ADD 28,Y
STO 38,Y
BLO JMP66
...

* * * MULTIPLICATION * *

CLRA
STA IND
LDA #Z
LDD #15

LOOP
LDA FRD
BEQ OVER1
LDY #COEF
BRA OVER2
LDY #COEFF

OVER1
LDY #COEF

OVER2
LDA IND
LDD A,Y
STD MLTR
LDD S
STD MLTR
LDX #MLTR
LDY #MLTN
LDU PROD1
CLR ,U
CLR 1,1
LDA 1,X
LOA 1,Y
MUL
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPD #65521</td>
<td>SKP75 STD 26,X</td>
</tr>
<tr>
<td>BLO JUP77</td>
<td>LDD TMP1</td>
</tr>
<tr>
<td>JUP76 ADDD #15</td>
<td>STD 22,X</td>
</tr>
<tr>
<td>JUP77 STD 14,X</td>
<td>LDD 24,X</td>
</tr>
<tr>
<td>LDD 12,X</td>
<td>ADDD 34,Y</td>
</tr>
<tr>
<td>ADDD 18,X</td>
<td>BCS SKP76</td>
</tr>
<tr>
<td>BCS JUP78</td>
<td>CMPD #65521</td>
</tr>
<tr>
<td>CMPO #65521</td>
<td>BLO SKP77</td>
</tr>
<tr>
<td>BLO JUP79</td>
<td>SKP76 ADDD #15</td>
</tr>
<tr>
<td>JUP78 ADDD #15</td>
<td>SKP77 STD 24,X</td>
</tr>
<tr>
<td>JUP79 STD TMP1</td>
<td>LDD 22,X</td>
</tr>
<tr>
<td>LDD 12,X</td>
<td>ADDD 28,X</td>
</tr>
<tr>
<td>SUBD 18,X</td>
<td>BCS SKP78</td>
</tr>
<tr>
<td>BCS JUP80</td>
<td>CMPD #65521</td>
</tr>
<tr>
<td>ADDD #65521</td>
<td>BLO SKP79</td>
</tr>
<tr>
<td>JUP80 STD 18,X</td>
<td>SKP78 ADDD #15</td>
</tr>
<tr>
<td>LDD TMP1</td>
<td>SKP79 STD TMP1</td>
</tr>
<tr>
<td>STD 12,X</td>
<td>LDD 22,X</td>
</tr>
<tr>
<td></td>
<td>SUBD 28,X</td>
</tr>
<tr>
<td></td>
<td>BCC SKP80</td>
</tr>
<tr>
<td></td>
<td>ADDD #65521</td>
</tr>
<tr>
<td>SKP67 ADDD #15</td>
<td>* 3-POINT POST-WEAVE *</td>
</tr>
<tr>
<td>SKP68 STD 22,X</td>
<td>** *********** *********** **</td>
</tr>
<tr>
<td></td>
<td>LDD 24,Y</td>
</tr>
<tr>
<td></td>
<td>ADDD 20,X</td>
</tr>
<tr>
<td></td>
<td>ADDD 20,Y</td>
</tr>
<tr>
<td></td>
<td>BCS SKP67</td>
</tr>
<tr>
<td></td>
<td>CMPD #65521</td>
</tr>
<tr>
<td></td>
<td>BLO SKP68</td>
</tr>
<tr>
<td></td>
<td>LDD 26,Y</td>
</tr>
<tr>
<td></td>
<td>LDD 28,X</td>
</tr>
<tr>
<td></td>
<td>CMPD #65521</td>
</tr>
<tr>
<td></td>
<td>BCC SKP70</td>
</tr>
<tr>
<td>SKP69 ADDD #15</td>
<td>JMP81 ADDD #15</td>
</tr>
<tr>
<td>SKP70 STD 34,Y</td>
<td>JMP82 STD 10,X</td>
</tr>
<tr>
<td>LDD 30,Y</td>
<td>LDD 21,X</td>
</tr>
<tr>
<td>SUBD 32,Y</td>
<td>ADDD 12,X</td>
</tr>
<tr>
<td>BCS SKP71</td>
<td>BCS JMP93</td>
</tr>
<tr>
<td>ADDD #65521</td>
<td>CMPD #65521</td>
</tr>
<tr>
<td>SKP71 STD 28,X</td>
<td>BLC JMP84</td>
</tr>
<tr>
<td>LDD 22,X</td>
<td>JMP83 ADDD #15</td>
</tr>
<tr>
<td>ADDD 28,Y</td>
<td>JMP84 STD 12,X</td>
</tr>
<tr>
<td>BCS SKP72</td>
<td>LDD 4,X</td>
</tr>
<tr>
<td>CMPD #65521</td>
<td>ADDD 14,X</td>
</tr>
<tr>
<td>SKP72 ADDD #15</td>
<td>BCS JMP85</td>
</tr>
<tr>
<td>SKP73 STD TMP1</td>
<td>CMPD #65521</td>
</tr>
<tr>
<td>LDD 22,X</td>
<td>JMP86 ADDD #15</td>
</tr>
<tr>
<td>ADDD 28,Y</td>
<td>JMP85 STD 14,X</td>
</tr>
<tr>
<td>BCC SKP74</td>
<td>LDD 6,X</td>
</tr>
<tr>
<td>ADDD #65521</td>
<td>ADDD 16,X</td>
</tr>
<tr>
<td>SKP74 STD 24,X</td>
<td>BCS JMP37</td>
</tr>
<tr>
<td>SUBD 34,Y</td>
<td>CMPD #65521</td>
</tr>
<tr>
<td>BCC SKP75</td>
<td>BLO JMP88</td>
</tr>
<tr>
<td>ADDD #65521</td>
<td>JMP87 ADDD #15</td>
</tr>
</tbody>
</table>
Appendix-D

JMP88
STD 16,X
LDD 8,X
ADD 18,X
BAS JMP89
CMPD #65521
BLD JMP90

JMP89
ADD #15
JMP100

JMP90
STD 18,X
LDD 10,X
ADD 20,X
BAS JMP91
CMPD #65521
BLD JMP92

JMP91
ADD #15
JMP101

JMP92
STD TMP1
LDD 10,X
SUBD 20,X
BAS JMP93
CMPD #65521
BLD JMP94

JMP93
ADD #15
LDX #AX

JMP94
STD 22,X
LDD TMP1
STD 12,X
LDD 14,X
ADD 24,X
BAS JMP95
CMPD #65521
BLD JMP96

JMP95
ADD #15
LDD 26,X

JMP96
STD TMP1
LDD 14,X
SUBD 24,X
BAS JMP97
CMPD #65521

JMP97
STD 24,X
LDD TMP1
STD 14,X
LDD 16,X
ADD 26,X
BAS JMP98
CMPD #65521
### Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD 24, Y</td>
<td>FDB 16087,29032,8748</td>
</tr>
<tr>
<td>LDD 16, X</td>
<td>FDB 23174,43615,1465</td>
</tr>
<tr>
<td>STD 26, Y</td>
<td>COEFR FDB 61153,5460,18364</td>
</tr>
<tr>
<td>LDD 28, X</td>
<td>FDB 46773,20640,5493</td>
</tr>
<tr>
<td>STD 28, Y</td>
<td>FDB 6552,57331,37975</td>
</tr>
<tr>
<td>COEFF FDB 1,6379,13376</td>
<td>FDB 29504,28641,12521</td>
</tr>
<tr>
<td>FDB 19136,18005,48647</td>
<td>FDB 5913,24748,21938</td>
</tr>
<tr>
<td>FDB 32759,8192,45457</td>
<td>END STRT</td>
</tr>
<tr>
<td>FDB 36817,5753,25311</td>
<td>*</td>
</tr>
</tbody>
</table>
Appendix-E

Backplane wiring connections for the parallel microprocessor system
Backplane pin connections for the parallel microprocessor system.

Flow of data from **SOURCE (TX)** → **DESTINATION (RX)**

**BOARD NO A - (PROCESSOR NO 1 2 3)**

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

**BOARD NO B - (PROCESSOR 4 5 16)**

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DATA IN</td>
<td>4 5</td>
</tr>
<tr>
<td>9-12</td>
<td>CLOCK</td>
<td>4 5</td>
</tr>
<tr>
<td>13-20</td>
<td>DATA OUT</td>
<td>4 5</td>
</tr>
<tr>
<td>21-24</td>
<td>DE</td>
<td>4 5</td>
</tr>
<tr>
<td>25-32</td>
<td>TX</td>
<td>4 → 9</td>
</tr>
<tr>
<td>33-36</td>
<td>CLOCK</td>
<td>4 → 9</td>
</tr>
<tr>
<td>39-44</td>
<td>RX</td>
<td>9 → 4</td>
</tr>
<tr>
<td>45-46</td>
<td>CLOCK</td>
<td>9 → 4</td>
</tr>
<tr>
<td>47-54</td>
<td>RX</td>
<td>3 → 4</td>
</tr>
<tr>
<td>55-56</td>
<td>CLOCK</td>
<td>3 → 4</td>
</tr>
<tr>
<td>57-64</td>
<td>TX</td>
<td>5 → 10</td>
</tr>
<tr>
<td>65-68</td>
<td>CLOCK</td>
<td>5 → 10</td>
</tr>
<tr>
<td>PIN#</td>
<td>FUNCTION</td>
<td>PROCESSOR#</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>57-64</td>
<td>TX</td>
<td>5 -&gt; 2</td>
</tr>
<tr>
<td>65-68</td>
<td>CLOCK</td>
<td>5 -&gt; 2</td>
</tr>
<tr>
<td>69-76</td>
<td>RX</td>
<td>10 -&gt; 5</td>
</tr>
<tr>
<td>77-78</td>
<td>CLOCK</td>
<td>10 -&gt; 5</td>
</tr>
<tr>
<td>79-86</td>
<td>RX</td>
<td>2 -&gt; 5</td>
</tr>
<tr>
<td>87-88</td>
<td>CLOCK</td>
<td>2 -&gt; 5</td>
</tr>
</tbody>
</table>

**BOARD NO C - (PROCESSOR 6 7 8)**

**SIDE A**

**SIDE B**

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DATA IN</td>
<td>6 7 8</td>
<td>1-8</td>
<td>RX</td>
<td>10 -&gt; 7</td>
</tr>
<tr>
<td>9-14</td>
<td>CLOCK</td>
<td>6 7 8</td>
<td>9-10</td>
<td>CLOCK</td>
<td>10 -&gt; 7</td>
</tr>
<tr>
<td>15-22</td>
<td>DATA OUT</td>
<td>6 7 8</td>
<td>11-18</td>
<td>TX</td>
<td>8 -&gt; 13</td>
</tr>
<tr>
<td>23-28</td>
<td>DE</td>
<td>6 7 8</td>
<td>19-24</td>
<td>CLOCK</td>
<td>8 -&gt; 13</td>
</tr>
<tr>
<td>29-36</td>
<td>TX</td>
<td>6 -&gt; 11</td>
<td>11-18</td>
<td>TX</td>
<td>8 -&gt; 3</td>
</tr>
<tr>
<td>37-40</td>
<td>CLOCK</td>
<td>6 -&gt; 11</td>
<td>19-24</td>
<td>CLOCK</td>
<td>8 -&gt; 3</td>
</tr>
<tr>
<td>29-36</td>
<td>TX</td>
<td>6 -&gt; 1</td>
<td>11-18</td>
<td>TX</td>
<td>8 -&gt; 9</td>
</tr>
<tr>
<td>37-40</td>
<td>CLOCK</td>
<td>6 -&gt; 1</td>
<td>19-24</td>
<td>CLOCK</td>
<td>8 -&gt; 9</td>
</tr>
<tr>
<td>41-48</td>
<td>RX</td>
<td>11 -&gt; 6</td>
<td>125-32</td>
<td>RX</td>
<td>13 -&gt; 8</td>
</tr>
<tr>
<td>49-50</td>
<td>CLOCK</td>
<td>11 -&gt; 6</td>
<td>133-34</td>
<td>CLOCK</td>
<td>13 -&gt; 8</td>
</tr>
<tr>
<td>51-58</td>
<td>RX</td>
<td>1 -&gt; 6</td>
<td>135-42</td>
<td>RX</td>
<td>2 -&gt; 3</td>
</tr>
<tr>
<td>59-60</td>
<td>CLOCK</td>
<td>1 -&gt; 6</td>
<td>143-44</td>
<td>CLOCK</td>
<td>3 -&gt; 3</td>
</tr>
<tr>
<td>61-68</td>
<td>TX</td>
<td>7 -&gt; 12</td>
<td>149-52</td>
<td>RX</td>
<td>9 -&gt; 3</td>
</tr>
<tr>
<td>69-74</td>
<td>CLOCK</td>
<td>7 -&gt; 12</td>
<td>153-54</td>
<td>CLOCK</td>
<td>9 -&gt; 3</td>
</tr>
<tr>
<td>61-68</td>
<td>TX</td>
<td>7 -&gt; 2</td>
<td>174</td>
<td>STATUS OUT</td>
<td></td>
</tr>
<tr>
<td>69-74</td>
<td>CLOCK</td>
<td>7 -&gt; 2</td>
<td>75</td>
<td>SYNC OUT</td>
<td></td>
</tr>
<tr>
<td>61-68</td>
<td>TX</td>
<td>7 -&gt; 10</td>
<td>76</td>
<td>SYNC IN</td>
<td></td>
</tr>
<tr>
<td>75-52</td>
<td>RX</td>
<td>12 -&gt; 7</td>
<td>77</td>
<td>SYSTEM CLOCK</td>
<td></td>
</tr>
<tr>
<td>83-84</td>
<td>CLOCK</td>
<td>12 -&gt; 7</td>
<td>73</td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>85-92</td>
<td>RX</td>
<td>2 -&gt; 7</td>
<td>79</td>
<td>RESET</td>
<td></td>
</tr>
<tr>
<td>93-94</td>
<td>CLOCK</td>
<td>2 -&gt; 7</td>
<td>129-61-93</td>
<td>+VCC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>132-64-96</td>
<td>GROUND</td>
<td></td>
</tr>
</tbody>
</table>

**BOARD NO D - (PROCESSOR 9 10 17)**

**SIDE A**

**SIDE B**

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DATA IN</td>
<td>9 10</td>
<td>1-8</td>
<td>RX</td>
<td>5 -&gt; 10</td>
</tr>
<tr>
<td>9-12</td>
<td>CLOCK</td>
<td>9 10</td>
<td>9-10</td>
<td>CLOCK</td>
<td>5 -&gt; 10</td>
</tr>
<tr>
<td>13-20</td>
<td>DATA OUT</td>
<td>9 10</td>
<td>11-18</td>
<td>RX</td>
<td>7 -&gt; 10</td>
</tr>
<tr>
<td>21-24</td>
<td>DE</td>
<td>9 10</td>
<td>19-20</td>
<td>CLOCK</td>
<td>7 -&gt; 10</td>
</tr>
<tr>
<td>25-32</td>
<td>TX</td>
<td>9 -&gt; 14</td>
<td>174</td>
<td>STATUS OUT</td>
<td></td>
</tr>
<tr>
<td>33-38</td>
<td>CLOCK</td>
<td>9 -&gt; 14</td>
<td>75</td>
<td>SYNC OUT</td>
<td></td>
</tr>
<tr>
<td>25-32</td>
<td>TX</td>
<td>9 -&gt; 4</td>
<td>76</td>
<td>SYNC IN</td>
<td></td>
</tr>
<tr>
<td>32-38</td>
<td>CLOCK</td>
<td>9 -&gt; 4</td>
<td>77</td>
<td>SYSTEM CLOCK</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix-E

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-38</td>
<td>CLOCK</td>
<td>9 -&gt; 3</td>
<td>178</td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>39-46</td>
<td>RX</td>
<td>14 -&gt; 9</td>
<td>179</td>
<td>RESET</td>
<td></td>
</tr>
<tr>
<td>47-48</td>
<td>CLOCK</td>
<td>14 -&gt; 9</td>
<td>29-61-93</td>
<td>+VCC</td>
<td></td>
</tr>
<tr>
<td>49-56</td>
<td>RX</td>
<td>4 -&gt; 9</td>
<td>32-64-96</td>
<td>GROUND</td>
<td></td>
</tr>
<tr>
<td>57-58</td>
<td>CLOCK</td>
<td>4 -&gt; 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59-66</td>
<td>RX</td>
<td>8 -&gt; 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67-68</td>
<td>CLOCK</td>
<td>8 -&gt; 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69-76</td>
<td>TX</td>
<td>10 -&gt; 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77-82</td>
<td>CLOCK</td>
<td>10 -&gt; 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69-76</td>
<td>TX</td>
<td>10 -&gt; 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77-92</td>
<td>CLOCK</td>
<td>10 -&gt; 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77-92</td>
<td>TY</td>
<td>10 -&gt; 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>83-90</td>
<td>RX</td>
<td>15 -&gt; 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91-92</td>
<td>CLOCK</td>
<td>15 -&gt; 10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BOARD MC E - (PROCESSOR 11 12 13)**

**SIDE A**

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DATA IN</td>
<td>11 12 13</td>
</tr>
<tr>
<td>8-14</td>
<td>CLOCK</td>
<td>11 12 13</td>
</tr>
<tr>
<td>15-22</td>
<td>DATA OUT</td>
<td>11 12 13</td>
</tr>
<tr>
<td>23-28</td>
<td>IOE</td>
<td>11 12 13</td>
</tr>
<tr>
<td>29-36</td>
<td>TX</td>
<td>11 -&gt; 6</td>
</tr>
<tr>
<td>37-38</td>
<td>CLOCK</td>
<td>11 -&gt; 6</td>
</tr>
<tr>
<td>39-46</td>
<td>RX</td>
<td>8 -&gt; 11</td>
</tr>
<tr>
<td>47-48</td>
<td>CLOCK</td>
<td>6 -&gt; 11</td>
</tr>
<tr>
<td>49-56</td>
<td>TX</td>
<td>12 -&gt; 7</td>
</tr>
<tr>
<td>57-60</td>
<td>CLOCK</td>
<td>12 -&gt; 7</td>
</tr>
<tr>
<td>49-56</td>
<td>TX</td>
<td>12 -&gt; 15</td>
</tr>
<tr>
<td>57-60</td>
<td>CLOCK</td>
<td>12 -&gt; 15</td>
</tr>
<tr>
<td>61-68</td>
<td>RX</td>
<td>7 -&gt; 12</td>
</tr>
<tr>
<td>69-70</td>
<td>CLOCK</td>
<td>7 -&gt; 12</td>
</tr>
<tr>
<td>71-78</td>
<td>RX</td>
<td>15 -&gt; 12</td>
</tr>
<tr>
<td>79-80</td>
<td>CLOCK</td>
<td>15 -&gt; 12</td>
</tr>
<tr>
<td>81-88</td>
<td>TX</td>
<td>13 -&gt; 8</td>
</tr>
<tr>
<td>89-92</td>
<td>CLOCK</td>
<td>13 -&gt; 8</td>
</tr>
<tr>
<td>89-92</td>
<td>TX</td>
<td>13 -&gt; 14</td>
</tr>
</tbody>
</table>

**SIDE B**

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>RX</td>
<td>9 -&gt; 13</td>
</tr>
<tr>
<td>9-10</td>
<td>CLOCK</td>
<td>9-10</td>
</tr>
<tr>
<td>11-18</td>
<td>RX</td>
<td>11-18</td>
</tr>
<tr>
<td>19-20</td>
<td>CLOCK</td>
<td>19-20</td>
</tr>
<tr>
<td>23-28</td>
<td>DE</td>
<td>23-28</td>
</tr>
<tr>
<td>29-36</td>
<td>TX</td>
<td>29-36</td>
</tr>
<tr>
<td>37-38</td>
<td>CLOCK</td>
<td>37-38</td>
</tr>
<tr>
<td>39-46</td>
<td>RX</td>
<td>39-46</td>
</tr>
<tr>
<td>47-48</td>
<td>CLOCK</td>
<td>47-48</td>
</tr>
<tr>
<td>49-56</td>
<td>TX</td>
<td>49-56</td>
</tr>
<tr>
<td>57-60</td>
<td>CLOCK</td>
<td>57-60</td>
</tr>
<tr>
<td>49-56</td>
<td>TX</td>
<td>49-56</td>
</tr>
<tr>
<td>57-60</td>
<td>CLOCK</td>
<td>57-60</td>
</tr>
<tr>
<td>61-68</td>
<td>RX</td>
<td>61-68</td>
</tr>
<tr>
<td>69-70</td>
<td>CLOCK</td>
<td>69-70</td>
</tr>
<tr>
<td>71-78</td>
<td>RX</td>
<td>71-78</td>
</tr>
<tr>
<td>73-90</td>
<td>CLOCK</td>
<td>73-90</td>
</tr>
<tr>
<td>81-88</td>
<td>TX</td>
<td>81-88</td>
</tr>
<tr>
<td>89-92</td>
<td>CLOCK</td>
<td>89-92</td>
</tr>
</tbody>
</table>

### BOARD MC E - (PROCESSOR 14 15 18)

**SIDE A**

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
<th>PROCESSOR#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DATA IN</td>
<td>14 15</td>
</tr>
<tr>
<td>9-12</td>
<td>CLOCK</td>
<td>14 15</td>
</tr>
<tr>
<td>174</td>
<td>STATUS OUT</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>SYNC OUT</td>
<td></td>
</tr>
</tbody>
</table>

**SIDE B**

<table>
<thead>
<tr>
<th>PIN#</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td>STATUS OUT</td>
</tr>
<tr>
<td>75</td>
<td>SYNC OUT</td>
</tr>
</tbody>
</table>
### Appendix-E

<table>
<thead>
<tr>
<th>Pin#</th>
<th>Function</th>
<th>Processor#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>DATA OUT</td>
<td></td>
</tr>
<tr>
<td>17-18</td>
<td>CLOCK</td>
<td>1</td>
</tr>
<tr>
<td>19-20</td>
<td>CLOCK</td>
<td>4</td>
</tr>
<tr>
<td>21-22</td>
<td>CLOCK</td>
<td>7</td>
</tr>
<tr>
<td>23-24</td>
<td>CLOCK</td>
<td>10</td>
</tr>
<tr>
<td>25-26</td>
<td>CLOCK</td>
<td>13</td>
</tr>
<tr>
<td>27-28</td>
<td>CLOCK</td>
<td>6</td>
</tr>
<tr>
<td>29-30</td>
<td>CLOCK</td>
<td>9</td>
</tr>
<tr>
<td>31-32</td>
<td>CLOCK</td>
<td>12</td>
</tr>
<tr>
<td>33-34</td>
<td>CLOCK</td>
<td>15</td>
</tr>
<tr>
<td>35-36</td>
<td>CLOCK</td>
<td>3</td>
</tr>
<tr>
<td>37-38</td>
<td>CLOCK</td>
<td>11</td>
</tr>
<tr>
<td>39-40</td>
<td>CLOCK</td>
<td>14</td>
</tr>
<tr>
<td>41-42</td>
<td>CLOCK</td>
<td>2</td>
</tr>
<tr>
<td>43-44</td>
<td>CLOCK</td>
<td>5</td>
</tr>
<tr>
<td>45-46</td>
<td>CLOCK</td>
<td>8</td>
</tr>
</tbody>
</table>

### SIDE A

<table>
<thead>
<tr>
<th>Pin#</th>
<th>Function</th>
<th>Processor#</th>
</tr>
</thead>
<tbody>
<tr>
<td>14-20</td>
<td>DATA OUT</td>
<td>176</td>
</tr>
<tr>
<td>21-24</td>
<td>CE</td>
<td>14</td>
</tr>
<tr>
<td>25-32</td>
<td>TX</td>
<td>14 -&gt; 9</td>
</tr>
<tr>
<td>33-36</td>
<td>CLOCK</td>
<td>14 -&gt; 9</td>
</tr>
<tr>
<td>34-40</td>
<td>CE</td>
<td>14</td>
</tr>
</tbody>
</table>

### CONTROL BOARD

### SIDE B

<table>
<thead>
<tr>
<th>Pin#</th>
<th>Function</th>
<th>Processor#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>STATUS IN</td>
<td></td>
</tr>
<tr>
<td>7-12</td>
<td>SYNC IN</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SYNC OUT</td>
<td></td>
</tr>
<tr>
<td>14-19</td>
<td>SYSTEM CLOCK OUTPUT TO PROCESSORS</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>RESET TO OTHER BOARDS</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>HALT TO OTHER BOARDS</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>-9V FOR RS-232 RX</td>
<td></td>
</tr>
<tr>
<td>29-61-93</td>
<td>+VCC 5V POWER FOR ALL BOARDS</td>
<td></td>
</tr>
<tr>
<td>32-64-96</td>
<td>GROUND</td>
<td></td>
</tr>
</tbody>
</table>
REFERENCES


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


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


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

Ref-6


Ref-7


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


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


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

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


(90) Signetics: 8X300 Data Sheet.

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