1

9

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Division / Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

55596.

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Ecrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

RAVIKANTI DEVENDER

Date of Birth — Date de naissance

11 06 1952

Country of Birth — Lieu de naissance

INDIA

Permanent Address — Résidence fixe

201 BOLL ST W # 423A
OTTAWA ONT Canada K1P 7E2

Title of Thesis — Titre de la thèse

ANALYSIS OF COMPATIBLE DIGITAL SIGNAL PROCESSOR
ARCHITECTURES

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

M.A. Eng.

Year this degree conferred — Année d'obtention de ce grade

1982

Name of Supervisor — Nom du directeur de thèse

V.A. BROWN

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film

L'auteur se réserve les autres droits de publication, ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur

Date

11 05 1982

Signature

R. Devender

MULTIBUS COMPATIBLE DIGITAL SIGNAL PROCESSOR ARCHITECTURES

by

C

Ravikanti Devender

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the Degree of
Master of Engineering:

Department of Systems & Computer Engineering
Faculty of Engineering
Carleton University
Ottawa, Ontario

May 1982

The undersigned recommend to
the Faculty of Graduate Studies and Research
the acceptance of the thesis

MULTIBUS Compatible Digital Signal Processor Architectures

by

Ravikanti Devender

in partial fulfillment of the requirements
for the degree of
Master of Engineering

Thesis Supervisor

Chairman,
Department of Systems &
Computer Engineering

Department of Systems & Computer Engineering
Faculty of Engineering
Carleton University

May 1982

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor, Professor B.A. Bowen, for his guidance, advice, understanding and tireless efforts to improve my style.

The author is indebted to the Communications Research Centre (CRC) of the Department of Communications, Canada, for their financial support in the form of a Research Grant (#03SU-36001 1 2197).

I owe special thanks to D.E. Freberg of CRC, Canada, for his discussions and suggestions.

To my fellow graduate students and friends, especially S. Boucouris and R.P. Reddy, I am appreciative for the fruitful discussions and comments on this project.

Mrs E. Carlyle's assistance in typing is appreciated very much indeed.

Last, but certainly not least, a very special thank you to my wife Rani and daughter Taruna, who provided a welcome break from the trials and tribulations of graduate studies

Abstract

A detailed design of a MULTIBUS compatible digital signal processor is presented in several alternative configurations. The processor contains sufficient intelligence to execute digital signal processing algorithms as directed. The processor acts as an array processor to an 8086 microprocessor. By the use of a specially designed dual-port memory, data flow bottlenecks usually found in such configurations are eliminated. The throughput is enhanced by high performance components and an optimized architectural design (a second order IIR filter executes in 3.2 microseconds; an in-place complex butterfly computation takes 3 microseconds). The super DSP configuration presented contains special addressing techniques for matrix manipulations.

Table of Contents

	<u>Page #</u>
1. Introduction	1
2. General Architecture	3
2.1 Introduction	3
2.2 CPU Considerations	5
2.3 ADC and DAC Considerations	7
2.4 Memory Considerations	7
2.5 DSP Considerations	10
2.6 Summary	10
3. Memory Design	11
3.1 Introduction	11
3.2 Design Methodology	11
3.3 Memory Mapping onto the MULTIBUS	14
3.4 MULTIBUS Information Transfer Protocol	14
3.5 Physical Design	19
4. DSP Design	28
4.1 Introduction	28
4.2 Communication Protocol	28
4.2.1 Semaphore Mechanism	28
4.2.2 Communication Protocol Methods	31
4.3 Parameter Block	33
4.3.1 Parameter Block Description	33
4.3.2 Examples of General Digital Signal Processing Instructions	35
4.4 Block and Stream Processing	38
4.5 Peripheral Circuits	42
4.6 Alternative Configurations	48
4.6.1 Block Processing	48
4.6.1.1 Single Memory Block Configuration	49
4.6.1.2 Multi Memory Block Configuration	53
4.6.2 Stream Processing	57
4.6.2.1 Single Memory Block Configuration	57
4.6.2.2 Multimemory Block Configuration	60
4.6.3 DSP with ALU	62
4.6.4 Super DSP	62
4.6.5 Estimation of Efficiency in different architectures	81
5. Summary	87
6. References	89

List of Figures

	<u>Page #</u>
Figure 1: Digital Signal Processing System	4
Figure 2: ISBC 86/12A Single Board Computer Block Diagram	6
Figure 3: Memory and Multibus Configuration	8
Figure 4: Interleaved Memory Requests	11
Figure 5: Multibus Address Space and Memory Mapping	12
Figure 6: Read AC Timing	17
Figure 7: Write AC Timing	17
Figure 8: Dual-port Memory Implementation Principle	19
Figure 9: Minimum and Maximum Mode 8086	20
Figure 10: 8086 Minimum and Maximum Mode (MULTIBUS) Command Timing	21
Figure 11: Multibus Read Write Decode	22
Figure 12(a): 64K x 16 Bit Memory	23
Figure 12(b): Chip Select Decoding	24
Figure 12(c): Multibus Buffer and Control	25
Figure 13: (Block and Stream Signal Processing	38
Figure 14: An Example of Block Processing: FIR Filtering in the Frequency Domain	39
Figure 15: Examples of Stream Processing: FIR Filtering in the Time Domain	40
Figure 16(a): 16 Bit Presettable Binary Counter	42
Figure 16(b): 16 Bit Presettable Up/Down Binary Counter	43
Figure 17: 16 Bit Transparent Latch (with 3-state outputs)	45

List of Figures

	<u>Page #</u>
Figure 18: 18 Bit 3-State Buffer (with common 3-input nor enable)	46
Figure 19: 12 Bit Wide 4-Unit Multiplexer	47
Figure 20: Hardware Architecture for Block Processing (Single Memory Configuration)	50
Figure 21: Block Diagram for Block Processing (Single Memory Configuration)	51
Figure 22: Micro Instruction	54
Figure 23(a): Hardware Architecture for Block Processing (Multi Memory Configuration)	55
Figure 23(b): Block Diagram for Block Processing: (Multi Memory Configuration)	56
Figure 24: Hardware Architecture for Stream Processing (Single Memory Configuration)	58
Figure 25: Hardware Architecture for Stream Processing (Multi Memory Configuration)	61
Figure 26: 16-Bit ALU	63
Figure 27: ALU Based DSP	64
Figure 28: Micro Instruction Bits for ALU	65
Figure 29(a): Super DSP Architecture	66
(b): Extra Microinstruction Bits for Super DSP	67
Figure 30(a): Basic Butterfly used in FFT	69
(b) Memory Values and Pointer Locations	69
(c): Concatenation of DPR and DPC	69
Figure 31(a): Memory Storage Organization and Parameter Block (2nd Order IIR Filter)	73
(b): Parameter Block for Generalized IIR Filter.	73
Figure 32: 16-Point FFT Flow Graph	76
Figure 33 (a): Utilization graph for block processing	84
(b): Utilization graph for stream processing	85
(c): Utilization graph for generalized IIR filter (Super DSP architecture)	86

1 INTRODUCTION

While it is generally admitted that digital signal processing has distinct advantages over the analog techniques until recently digital processing could not compete in most applications due to cost, size and power constraints. Not surprisingly, advances in VLSI technology in the last few years have changed this situation. The hardware multiplier solved the worst problem in the digital area, and it later evolved to become a full arithmetic unit for DSP for fixed point applications. Out of the three hardware elements needed for DSP, namely storage, addition, and multiplication implementing the first two was no problem in the early seventies but it took high speed VLSI, introduced in the late seventies to make a single-chip, useful word size multiplier a reality. Added to this, reductions in the cost, size, higher speed and performance of modern digital components have made digital signal processing an attractive option in the implementation of signal processors.

A digital signal processor is a machine which implements one or more appropriate numerical algorithms. Typically these algorithms might provide filter functions, spectral analysis or some form of signal correlation. For off-line or low bandwidth applications they could be implemented in a general purpose computer or mini/micro computer. However, in any high performance system a dedicated, or at least optimized processor must be constructed. Only microprogrammed, bipolar bit-sliced, devices provide the speed and flexibility necessary for such a processor, or specialized DSP chips such as the AMI's S2811(75) or the NEC's μ PD7720 (73,74) can be used but they have limited memories.

Consider a mini computer. It cannot process signals of more than a few kilohertz in real time. As a general purpose signal processor, it is simply not equipped to perform the complex computations fast enough. To overcome this problem attached array processors were developed such as the Analogic AP400.

Similarly, the single chip microprocessors such as the INTEL 8085, 8086, or Texas Instruments 9900 do not provide the necessary speed for real time processing of signals in the audio-frequency range. This thesis presents the design of a high performance digital signal processor to act as a peripheral to an Intel 8086 microprocessor in a MULTIBUS system.

The MAC (Multiplier Accumulator) chip recently produced by TRW forms a basic computing element for performing several types of basic operations with suitable inputs. This device provides the computational element for almost all digital signal processing functions. In order to utilize effectively this device a support processor is required to organize the flow of tasks and provide the data flow and control. This function is usually provided by a microprocessor.

The thesis has two broad aspects - first, to consider the architecture of a general purpose system based on the Intel MULTIBUS controlled by the 16-bit 8086 microprocessor; second, to consider the design of the digital processor board to accomplish a wide variety of DSP basis functions. The digital processor proposed makes use of a bit-sliced microcontroller to meet the speed requirements, and also to give greater flexibility in various system structures.

The thesis is organized as follows. The global architecture of the digital signal processor is presented in Chapter 2. Different blocks of the global architecture are broadly considered here. Memory design is presented in detail in Chapter 3. A dual port memory concept is the highlight of the digital signal processor. Finally, in Chapter 4, the communication protocol between the host CPU and the digital signal processor, is presented. Then, after giving a brief introduction to block and stream processing, various configurations of the digital signal processor are presented giving the details of the design and advantages and disadvantages. Depending on the performance requirements and funds available, one of the configurations can be chosen for prototyping. The thesis ends with a summary, and an extensive list of references useful in designing a digital signal processor.

2. General Architecture

A processor architecture is described that permits a substantial increase in throughput from that obtained with more conventional designs. The goals of the design are to eliminate the bus congestion problem usually found in the array processor configuration and to enhance the overall system throughput.

2.1 Introduction:

A discussion of the high-level general architecture is presented in this section. Figure 1 shows a block diagram of the high level architecture of the proposed digital signal processor around the Intel's multibus environment. The 8086 processor based CPU system will execute user programs and act as a task coordinator and resource allocator. For high throughput the ADC and DAC will contain sufficient intelligence for DMA transfers as directed or required. The digital signal processor will contain sufficient intelligence to execute digital signal processing algorithms as directed by the 8086 CPU system. The DSP works concurrently with the 8086 CPU. It will contain the MAC chip, bit slice micro-processor, sequencer, etc. The micro-program controller will send a series of data and coefficient words into the MAC and read out the answers. Under these constraints, with the DSP accessing the memory via the MULTIBUS, the system throughput is reduced drastically as the MULTIBUS is tied up during the task execution by the DSP. The system throughput can be increased by having a dual-ported memory in the DSP, with the second port accessible to DSP. Then the MULTIBUS is free during the task execution by the DSP. The dual-ported memory falls into the host CPU's memory space even though it resides in the DSP, and it will be transparent to the host CPU system even during the task execution by the DSP. Thus the traditional bus congestion problem associated with the array processors is overcome. The use of high speed components and an optimized architecture increases the throughput of the system, achieving the performance goals.

DIGITAL SIGNAL PROCESSING SYSTEM

FIGURE:1

A very high level architecture was presented above. Another outstanding feature of the architecture is its modularity. For example we can have 2 CPU systems and 2 DSP modules connected to the MULTIBUS. Each block presented above is discussed in greater detail below.

2.2 CPU Considerations

As discussed earlier, an Intel 8086 processor based CPU system is suggested. This system can be an iSBC 86/12A. Memory expansion (if required) can be done using iSBC 300 boards. An iSBC 86/12A single board computer block diagram is shown in Figure 2. Some of the salient features of the board are given below. It has

- 8086 - 16-bit HMOS microprocessor central processor unit
- 32K-bytes of dual-port read/write memory expandable on-board to 64K bytes with on-board refresh
- sockets for up to 16K bytes of ROM expandable on-board to 32K bytes
- 24 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators.
- programmable synchronous/asynchronous RS232C compatible serial interface with software selectable band rates
- two programmable 16-bit BCD or binary timers/event counters
- 9 levels of vectored interrupt control, expandable to 65 levels
- Multibus interface for multimaster configurations and system expansion

2.3 ADC & DAC Considerations

A multibus-compatible ADC and DAC are assumed available with sufficient intelligence for DMA transfers. A brief description of the functioning of the ADC is given below.

During block processing, the ADC should send one new sample to the host-CPU memory (8086) by DMA. Once the required block of data samples are available in the memory, the host CPU can send command signals to the DSP to process the data block. During stream processing, the host CPU after waiting for the DSP signal, will send the command to deliver the new sample to the ADC, which in turn will send the new sample to the DSP's New Sample register. Once this is done the host CPU will signal the DSP and the process continues as discussed in Chapter 4.

2.4 Memory Considerations

The DSP requires a minimum of three main memories, plus the scratchpads stack and registers. The main memories are:

- 1) DATA memory
- 2) PROGRAM memory
- 3) COEFFICIENT memory

The memories required for more general applications are.

- 1) DATA memory expandable to 64K words
- 2) PROGRAM memory expandable to 16K words
- 3) COEFFICIENT memory expandable to 16K words

Figure 3 shows the block diagram of the dual port memory. Here each of the memories is shown as being dual ported. This has been done for several reasons:

- 1) To permit downline loading of programs, data and coefficients
- 2) To facilitate DMA transfers to and from the DSP

DSP MEMORY AND MULTIBUS CONFIGURATION

FIGURE: 3

- 3) To perform address translation. The memories should be addressable within the one megabyte address space of the 8086 (i.e., 20 bit address space), and as well as addressable by the DSP (i.e., < 16 bit address space)
- 4) To provide a communication mechanism between the CPU and the DSP

Some of the important design decisions and the reasons for the options chosen are given below:

- 1) Separation of program and data storage (i.e., array processor format) is used, to facilitate parallelism and pipelining of the arithmetic operations, for increased efficiency and throughput. Without this separation, the duty cycles of the multiplier and memory access would be reduced, severely limiting the maximum operating speed of the system.
- 2) The coefficients are stored in random access memory (RAM) to permit coefficient changes. With fixed coefficients, the user would be unable to reprogram the board to handle different types of filters, or to change from digital filtering to spectral analysis. Furthermore, adaptive filters which are commonly used in radar, speech synthesis, and communications, require changeable coefficients. However, a dedicated non-adaptive filter or FFT processor could be built with a coefficient read only memory (ROM) instead, to reduce hardware and software complexity significantly.
- 3) A hardware multiplier/accumulator is assumed for any high speed digital signal processing system; otherwise the multiplications will require excessive amounts of time.
- 4) Separate memories are used for data and coefficients, to permit parallel access into the data and coefficient inputs of the multiplier. Otherwise, the multiplier would have to wait through two separate memory access periods (coefficient and data) for each multiply cycle.

2.5 DSP Considerations

The DSP is dealt with in depth in Chapter 4, with different alternatives. Hence it is not discussed here.

2.6 Summary

Starting with the high level architecture of the overall system some of the salient features of the proposed architecture were presented above. The next Chapter describes the memory design in greater detail, and in Chapter 4, the DSP design in different alternative configurations is presented.

3. Memory Design

3.1 Introduction

The memory organization was described in Section 2.4 along with the many advantages of the chosen system. In this section the design is presented. As discussed earlier in Section 2, a dual port memory is required for our application.

3.2 Design Methodology

Basically we should have dual access to the memory, one by CPU and the other by the DSP. There are two basic approaches to the dual port problem:

Method 1:

To use a moderate speed memory with a smart controller. The requests would be buffered, the MULTIBUS held busy and a cycle stolen when available.

Method 2:

To use a high speed memory and a simple controller. Requests would then be interleaved, using the DSP clock for synchronization as shown in Figure 4

Method 2 described above is chosen for the design for the following advantages:

- 1) A simple, easy to debug controller. Several flipflops and data selectors or TRI-STATE logic will make the design easy. The DSP clock drives the steering logic.
- 2) High DMA and MULTIBUS memory bandwidth. This has been the traditional problem in array processors.

Even though this method has the advantages given above it is not without any disadvantages. The disadvantages are.

- 1) Memory chips are marginally more expensive as they are high speed memories [access time in the order of 45-70 ns].
- 2) High speed memories are power hungry.

INTERLEAVED MEMORY REQUESTS

FIGURE:4

MULTIBUS ADDRESS SPACE AND MEMORY MAPPING

FIGURE:5

3.3 Memory Mapping onto the MULTIBUS

Keeping in mind that the 8086 uses segment registers to convert the 16-bit address to a 20 bit address, a rough mapping of the DSP memory onto the 1 megabyte, 20 bit CPU address space is shown in Figure 5.

3.4 MULTIBUS Information Transfer Protocol

Before going into the detailed design of memory the MULTIBUS information transfer protocol is described below. The MULTIBUS structure provides a common element for communication between a wide variety of system modules such as single board computers memory, digital and analog I/O expansion boards, and peripheral controllers. An overview of the MULTIBUS description is presented below. The signal lines on the MULTIBUS can be grouped in the following categories.

- 20 address lines
- 16 bidirectional data lines
- 8 multilevel interrupt lines
- bus control lines
- timing and power supply lines

Modules that use the MULTIBUS system bus have a master-slave relationship. A bus master module can drive the command and address lines; it can control the bus. A Single Board Computer is an example of a bus master. A bus slave cannot control the bus. Memory and I/O expansion boards are examples of bus slaves. The MULTIBUS architecture provides for both 8 and 16-bit bus masters and slaves.

Notice that a system may have a number of bus masters. Bus arbitration is required when more than one master requests control of the bus at the same time. A bus clock is usually provided by one of the bus masters and may be derived independently from the processor clock. The bus clock provides a timing reference for resolving bus contention among multiple requests from bus masters. For example, a processor and a DMA (direct memory access) module may both request control of the bus. This feature allows different speed masters to share resources on the same bus. Actual transfers via the bus, however, proceed asynchronously with respect to the bus clock. Thus, the transfer speed is dependent on the transmitting and receiving devices only.

The bus design prevents slow master modules from being excluded in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. Once a bus request is granted single or multiple read/write transfers can proceed. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations and high-speed direct-memory-access (DMA) operations. However, the master-slave capabilities of the bus are by no means limited to these two applications.

Coming to the Information Transfer Protocol lines, a bus master provides separate read/write command signals for memory and I/O devices such as MRDC/, MWTC/, IORC/, IOWC/, as explained below. (N.B.: whenever '/' is used, it indicates active low).

When a read/write command is active, the address signals must be stabilized at all slaves on the bus. For this reason, the protocol requires that a bus master must issue address signals (and data signals for a write operation) at least 50 ns ahead of issuing a read/write command to the bus, initiating the data transfer. The bus master must keep address signals unchanged until at least 50 ns after the read/write command is turned off, terminating the data transfer. A bus slave must provide an acknowledge signal to the bus master in response to a read or write command signal.

MULTIBUS COMMAND SIGNALS:

MRDC/: Memory read command; indicates that the address of a memory location has been placed on the system address lines and specifies that the contents (8 or 16 bits) of the addressed location are to be read and placed on the system data bus. MRDC/ is asynchronous with respect to BCLK/.

MWTC/: Memory write command; indicates that the address of a memory location has been placed on the system address lines and that data (8 or 16 bits) has been placed on the system data bus. MWTC/ specifies that the data is to be written into the addressed memory location. MWTC/ is asynchronous with respect to BCLK/.

IORC/: I/O read command; indicates that the address of an input port has been placed on the system address bus and that the data (8 or 16 bits) at that input port is to be read and placed on the system data bus. IORC/ is asynchronous with respect to BCLK/.

IOWC/: I/O write command; indicates that the address of an output port has been placed on the system address bus and that the contents of the system data bus (8 or 16 bits) are to be output to the addressed port. IOWC/ is asynchronous with respect to BCLK/.

XACK/: Transfer acknowledge signal, the required response of a slave board which indicates that the specified read/write operation has been completed. That is, data has been placed on, or accepted from, the system data bus lines. XACK/ is asynchronous with respect to BCLK/.

Read and write data timing diagrams are given in Figure 6 and Figure 7; they are explained below.

Read Data:

Figure 6 shows the read operation timing diagram. The address must be stable (t_{AS}) for a minimum of 50 ns before command (IORC/ or MRDC/). This time is typically used by the bus interface to decode the address and thus provide the required device selects. The device selects establish the data paths on the user system in anticipation of the strobe signal (command) which will follow. The minimum command pulse width is 100 ns. The address must remain stable for at least 50 ns following the command (t_{AH}). Valid data should not be driven onto the bus prior to command and must not be removed until the command is cleared. The XACK/ signal, which is a response indicating the specified read/write operation has been completed, must coincide or follow both the read access and valid data (t_{DXL}). XACK/ must be held until the command is cleared (t_{XAH}).

Write Data:

The write operation AC timing diagram is shown in Figure 7. During a write data transfer valid data must be presented simultaneously with a stable address. Thus, the write data setup time (t_{DS}) has the same requirement as the address setup time (t_{AS}). The requirement for stable data both before and after command (IOWC/ or MWTC/) enables the bus interface circuitry to latch data on either the leading or trailing edge of command.

Figure 6: Read AC Timing

Figure 7: Write AC Timing

3.5 Physical Design

The basic design methodology of the dual port memory was described in Section 3.2. How this principle is physically implemented is presented in this section. First the circuit implementation is presented at the block diagram level and then the detailed circuit is presented.

Figure 8 shows how the interleaving principle described earlier can be implemented. Tri-state buffers (74LS365) can be used to select the address lines, Tri-state buffers (74LS365) or bidirectional buffers (8286 Transceivers) can be used in the data line path. In the case of the 8286, CLK, $\overline{CLK}$ are used to select the chip by connecting it to the OE pin, and the $\overline{RD}/\overline{WR}$ is used to select the direction of the chip. The detailed circuit based on this approach for a 64K x 16 bit memory is shown in Figure 12. The memory chips are 2149H-3 (maximum address access time 55 ns, maximum chip select access time 25 ns). Since the multibus address is 20 bits wide the decoder decodes the high order 4 bits to an OK signal, which in turn with CLK is used to select the buffers. Similar circuits for 16K or other values can be built with a changed decoder or by properly choosing 8205 enable pins. This dual port memory is used in the DSP and is shown as a block in future diagrams.

At this point let us look at the signals sent by the 8086 over the MULTIBUS for memory requests. The basic 8086 in the maximum mode configuration is shown in Figure 9(b). Figure 9(a) shows the minimum mode configuration.

The 8288 bus controller originates all bus commands. These bus command signals were explained in Section 3.4 under MULTIBUS information transfer protocol.

The maximum mode command timing for the 8086 is shown in Figure 10. From this it is clear that if CLK is used for CPU and $\overline{CLK}$ for DSP timings, and if both 8086 and DSP operate at 5 MHz clock (i.e., 200ns, cycle time) then for writes there is no problem. For reads we need a latch at the output of the memory so that the data read is latched onto the CPU at the rising edge of MRDC signal. A latch on the read side of the DSP is also advisable if the data is used elsewhere without being latched. In case the MULTIBUS signals are of different frequency we need a synchronizing mechanism with the DSP clock. This circuit is shown below in Figure 11(b).

DUALPORT MEMORY IMPLEMENTATION PRINCIPLE

FIGURE:8

(a) Minimum Mode 8086

(b) Maximum Mode 8086 (MULTIBUS mode)

Figure 10: 8086 Minimum and Maximum Mode (MULTIBUS) Command Timing

(a) MULTIBUS READ DECODE

(b) Change $\overline{MRDC}$ to $\overline{MWTC}$,

OUT to $\overline{WR}$, to get MULTIBUS WRITE DECODE

MULTIBUS READ, WRITE DECODE

FIGURE 11

64 K X 16 bit Memory

(All ICs are 2149H-3)

FIGURE :12 A

CHIP SELECT DECODING

FIGURE:12 B

MULTIBUS BUFFER AND CONTROL

FIGURE: 12(c)

As shown in Figure 11(b) if the MULTIBUS memory request command is of lower frequency than the DSP clock, then the circuit shown generates a single WR pulse for each memory write command and hence synchronizes the memory with the MULTIBUS. A similar circuit is shown for read in Figure 11(a). The INIT signal used is described below.

The initialization signal resets the entire system to a known internal state. Normally INIT may be driven by one of the bus masters or by an external source such as a front panel reset switch. This signal is used to initiate the flipflops used to generate the WR and RD signals for the memory.

By incorporating these synchronization circuits and feeding RD, & WR signals towards the memory we achieve the purpose of one memory read and write per MULTIBUS memory read and write cycle.

4. DSP Design

4.1 Introduction

Since semaphores are used in the communication protocol, they are described below for completeness.

The DSP design that permits a substantial increase in the throughput from the conventional designs is presented in this section. This section is organized as follows. First the basic communication protocol between the DSP and the host CPU (8086 system) is presented. Then the parameter block used to set up the DSP tasks is described. Finally the DSP hardware architecture is presented in several alternative configurations.

4.2 Communication Protocol

Communication between the host CPU and the DSP is described below. The mechanism uses the DSP as a slave processor with a semaphore, described in Section 4.2.1.

4.2.1 Semaphore Mechanism

Semaphores are used when two or more processors are in execution concurrently and must cooperate during the computation. Here the concern lies both in the synchronization of the processors so that the parallel computation can be carried out correctly, and also in techniques for creating and terminating parallel execution paths within a program.

To illustrate an example of the problem, say we want to perform a summation in parallel. The program is to form the sum of the elements of the vector $V[i]$, $1 \leq i \leq N$. To do this in parallel, the program forks into N branches, each of which performs the statement

$$SUM := SUM + V[i]$$

for i varying from 1 to N . Without synchronization and interlocking, the execution might occur as follows:

1. Processor 1 fetches the value of SUM from memory.
2. Processor 2 fetches the value of SUM from memory.

3. Processor 1 adds $V[i_1]$ to its private value of SUM and restores the new value of SUM in memory.
4. Processor 2 adds $V[i_2]$ to its private value of SUM, different now from the value in memory, and restores it in memory.

This sequence of instructions produces incorrect results, because the effect of adding $V[i_1]$ by processor 1 has been lost. The problem arises because processors 1 and 2 copy the value of SUM and operate on their private copies. For the program to be correct, a processor must be able to fetch and update the value of SUM without any intervening memory references to SUM.

These problems can be overcome by using operations called WAIT and SIGNAL. Both of these operations supply a memory address as a parameter, and operate on the contents of that memory address. The operations must be indivisible in the sense that, once initiated, the item at the special address is fetched, modified, and returned to memory, and no other process may access the same memory cell until this operation is complete. The idea is to use WAIT and SIGNAL as shown below.

```
comment: several processors may be executing statements
        before the WAIT and after SIGNAL;
WAIT (FLAG);
```

```
comment: critical section. At most one processor
        reaches this point at any time;
SUM:=SUM+V[i];
SIGNAL ((FLAG));
```

The statements between WAIT and its corresponding SIGNAL are called the critical section. The Boolean variable FLAG controls access to the critical section. The WAIT operation on FLAG permits a process to continue if and only if no other process is in the critical section. When a process leaves the critical section, it signals, which in turn permits exactly one of the waiting processes to enter the critical section.

In an ALGOL-like notation the actions of WAIT and SIGNAL are.

```

procedure WAIT (FLAG);
  begin FLAG:=FLAG - 1;
        if FLAG < 0 then suspend this process;
  end;
procedure SIGNAL (FLAG)
  begin FLAG:=FLAG+1;
        if FLAG < 0 then awaken a program suspended by
          this flag;
  end;

```

FLAG is initialized to 1 at the beginning of the computation. Except during execution of WAIT and SIGNAL, the value of FLAG is the negative of the number of processes waiting to enter the critical section. Thus if a process performs a WAIT and discovers the value of FLAG to be 0, it can proceed. Likewise, if a process performs a SIGNAL and finds the value of FLAG to be less than -1, at least one other process is waiting.

The action of process suspension and process-awakening is assumed to be carried on by operating system programs. The variables used as arguments of WAIT and SIGNAL are known as semaphores and are assumed to have special attributes. Each semaphore has a queue associated with it, and the entries in the queue refer to processes suspended because of the value of that variable. The operating system suspends a process by storing its state, and by enqueueing sufficient information to restart the program.

When a process is suspended, its processor is available for other computation. The operating system selects a process from a queue of ready-to-run processes, and restarts its computation.

During a signal operation, if a process needs to be restarted, the operating system selects one from the queue associated with the queue semaphore and places this on the ready-to-run queue.

The WAIT and SIGNAL operations can be used as general-purpose operations for synchronizing processes, not necessarily just to create critical sections. There are many ways of implementing these operations. One possible implementation of a WAIT is literally to wait by placing the program in a loop that continually tests the semaphore until a signal is received. Although this appears to be inefficient, it may actually be the most efficient way to implement the WAIT in some instances. This type of implementation mechanism is used in the system and is described in Section 4.6.1.1 in greater detail.

4.2.2 Communication Protocol Methods

The host CPU sets up a parameter block (which is discussed in detail in Section 4.3) and tells the DSP by signalling a semaphore. The DSP waits on this semaphore and then processes the data, and signals back to the CPU. This type of communication is possible as the 8086 is well equipped for the semaphore mechanism. Two such protocol mechanisms are discussed below.

a) Method 1 (3 semaphore mechanism)

DSP waits on Sema 1

DSP indicates to CPU that it has taken the parameter block by Sema 2

DSP indicates that it is done by Sema 3

VAR sema 1, sema 2, sema 3 : semaphore

Initialization:

Sema 1:=0,

Sema 2:=1,

Sema 3:=0,

; host CPU procedure:

COBEGIN:

WAIT (Sema 2),

set the task for DSP by writing the parameter block,

SIGNAL (Sema 1),

WAIT (Sema 3),

use the results.

COEND.

```

; DSP procedure:
  COBEGIN.
    WAIT (Sema 1)
    take parameter block
    SIGNAL (Sema 2)
    Do the allocated task
    SIGNAL (Sema 3)
  COEND.

```

The advantage here is, the DSP takes the PB and signals the CPU, so that the CPU can allocate another task to the DSP without waiting for the DSP to complete the previously allocated task. This is similar to pre-fetching the instruction procedure.

b) Method 2 (2 semaphore mechanism)

Here we can have only 2 semaphores. But pre-fetching cannot be done.

```

Sema 1: CPU sets it once PB is set.
Sema 2: DSP sets it once it is done with the
        allocated task.
VAR sema 1, sema 2: semaphore

```

```

; CPU procedure:
  COBEGIN:
    set parameter block
    SIGNAL (Sema 1)
    WAIT (Sema 2)
    use the results
  COEND.

```

```

; DSP procedure:

  COBEGIN.
    wait (Sema 1)
    take PB data, etc.
    Do the task
    SIGNAL (Sema 2)

```

```

  COEND:

```

The above mentioned semaphore mechanism can be implemented on the proposed system using the LOCK prefix in conjunction with the XCHG (exchange register with memory) instruction available in the 8086 instruction set. The common resources here are the parameter block, data block, result block, etc. The parameter block is explained in detail below.

4.3⁶ Parameter Block

The parameter block described in the communication protocol mechanism is described in greater detail with examples of some of the DSP instructions.

4.3.1 Parameter Block Description

The parameter block concept gives great flexibility to the user. Each DSP instruction will have a parameter block. This block, which consists of up to (say 16 or 8) words of main memory, contains all control information such as array lengths and starting addresses. The DSP will not alter the parameter block, allowing more than one DSP instruction to use the same parameter block if necessary.

In general, programming the DSP consists of setting up a parameter block and executing a DSP instruction. The digital signal processing instructions are executed by the DSP for the host CPU, i.e., it is a separate processor controlled by the 8086 through semaphores and parameter blocks.

The format of the parameter block is shown in Table 1. In general, each parameter word has a specific function. This organization simplifies programming, especially when several operations are to be performed on one array: a single parameter block may be used for several instructions, each of which uses different words from the block. The parameter block can be stored at a fixed place in the memory all the time.

Word	Function
0	Element Count
1	Index of destination array
2	Index of first source array
3	Index of second source array
4	Main memory pointer
5	Destination step value
6	First operand step value
7	Second operand step value
8	Error mask
9	Continuation register

Table 1

More complicated tasks can be handled by the DSP by increasing the parameter block length to 16 and by specifying error masks and continuation registers, etc. For example, the continuation register can be used to control the DSP response to various error conditions. Instead of DSP stopping work after detecting an error, (say overflow or underflow) we can specify the DSP to continue work after detecting a particular type of error.

4.3.2 Examples of General Digital Signal Processing Instructions

A description of setting up parameter blocks for some general digital signal processing instructions is presented below.

1) Convolution:

CONVR (convolution of real arrays)

M-1

$$Z_n = \sum_{m=0}^{M-1} X_{n-m} Y_m$$

m=0

n = 0, 1, ..., N-1

Here it is assumed that X has a length (N+M-1). The first (M-1) elements are initial conditions. They are stored in elements with negative subscripts X(-M+1) to X(-1):

CONVRZ assumes that all initial conditions are zero. The number of elements in X is equal to N.

PB:

Word	Name	Contents
0	N	Element count for source array X
1	Z ₀ INDEX	Index of real array Z
2	X ₀ INDEX	Index of real array X
3	Y ₀ INDEX	Index of real array Y
4	M	Element count for source array Y

2) Correlation

$$Z_n = \sum_{m=0}^{M-1} X_{n+m} Y_m$$

$$n = 0, 1, \dots, N-1$$

Z_n is the correlation product of the source arrays X and Y , and places the result in the destination array Z .

Again we can have

CORR (correlation of real arrays)
CORC (correlation of complex arrays)

The PB is the same as that for convolution process.

3) FFTC: (Fast Fourier Transform of Complex Array)

$$X_n = \sum_{k=0}^{N-1} X w^{nk}$$

$$w = e^{2\pi i/N} ; \text{ for the forward transform}$$

$$w = e^{-2\pi i/N} ; \text{ for the inverse transform}$$

The parameter block is shown below:

Word	Name	Contents
0	N	Element count must be a power of 2)
1	X_0 INDEX	Index of complex array X

Coefficient ROM is assumed to be in DSP. Results can be stored back in array X in bit reversed form.

4) Power Spectral Density: (Magnitude squared calculation)
(PSD)

$$Z_k = |X_k|^2 = X_k \cdot X_k^*$$

Parameter block:

Word	Name	Contents
0	N	No. of points in a sequence
1	Z ₀ INDEX	Index of Result array Z _k
2	X ₀ INDEX	Index of Result array X

Some other functions can be specified.

5) FIR operation:

$$Y(n) = \sum_{k=0}^N a_k x(n-k)$$

This can also be done as previously by specifying
a_k's. and X₀ index; Y₀ index.

4.4 Block and Stream Processing

Before going into the final design of the digital signal processor, methods used in processing digital signals are presented below. Digital signals can be processed on a block or stream oriented basis, as shown in Figure 13. Figure 14(a) illustrates block processing(91).

Samples are accumulated on a block basis by a sampling unit and then the entire block of samples is handed over to a processing unit which must produce a result before the next block of samples arrives. Stream processing is illustrated by Figure 13(b): Samples are made available one at a time to the processing unit which must produce a result before the next sample arrives. By definition, block processing units perform their calculations on a block of past samples. Stream processing units, on the other hand, may perform their calculations using past sample values or they may perform their calculations recursively, based only on the current samples and on an internal state which has been computed from previous samples as they arrived.

Examples of block and stream processing are provided in Figures 14 and 15 respectively(adopted from Reference 91). As shown by Figure 14, finite impulse response filtering in the frequency domain is an example of block processing. The signal is broken into blocks, each block of which is transformed and multiplied against the precalculated transform of the finite impulse response of the desired filter. The inverse transformations for each block are then combined for the different blocks by the overlapped add method to produce the final filtered output. The calculation is inherently block oriented because a block of signal samples must be accumulated before the FFT calculation can begin.

Finite impulse response filtering may also be done directly in the time domain as shown by Figure 15(a). Here the finite impulse response of the desired filter is directly convolved with the signal time sequence to produce a filtered time sequence. The convolution is a sum of weighted past values of the signal time sequence. At any time the number of past values of this signal which must be included in the calculation is determined by the number of samples in the finite impulse response of the filter. This number of

a) Block Processing b) Stream Processing

Figure 13 : Block and Stream Signal Processing

Figure 14: An Example of Block Processing: FIR Filtering in the Frequency Domain

a) SLIDING WINDOW FILTERING

b) RECURSIVE FILTERING

Figure 15 : Examples of Stream Processing:
FIR Filtering in the Time Domain

samples represents a time "window" through which the filter looks at the past of the signal. Therefore, this approach may be characterized as "sliding window-filtering". The window is however, not a block, in the sense in which we have used the term to characterize block processing, because samples are not accumulated for the entire window before they are made available to the filtering unit and the results are calculated for each sample rather than for a block of samples. Thus for each signal sample a weighted sum over a window of past samples must be performed by the filter for each sample before the next sample arrives. Note the assumption that this window of samples is available to the filter; it can be made so in a signal processing system either in shared memory or in the private memory of the filter unit.

An alternate approach to stream processing, not restricted to finite impulse filtering, is indicated in Figure 15(b). In this approach the difference equations defining a generalized linear filter are formulated in state terms such that the filtered output may be produced as a weighted sum of the current state components and the last input sample. The number of state components required depends on the order of the filter.

4.5 Peripheral Circuits

Some of the peripheral circuits used in the DSP such as counters, latches, multiplexers, and buffers are described in this section. Brief description of the hardware is given as most of the circuits are self-explanatory.

Counters

Figure 16A and 16B show the detailed circuit diagram of 16 bit presetable binary counters. Figure 16A shows the circuit implementation using 74163 where presetting is also synchronous, whereas Figure 16B shows using 74191 which is having the facility of up/down counting with the same number of control inputs. Both 74163 and 74191 are 16 pin ICS and available in low power Schottkey series dissipating some power. Hence unless the cost difference is too much, the circuit using 74191 is advisable as it gives the extra facility of down counting also.

CONTROL INPUTS:

1) LD , 2) COUNT , 3) ENABLE

16-BIT PRESETTABLE BINARY COUNTER

FIGURE:16 A

CONTROL INPUTS:

- 1) $\overline{LOAD}$ (Asynchronous)
- 2) $\overline{UP/DOWN}$, 3) $\overline{ENABLE}$

16-BIT PRESETTABLE UP/DOWN BINARY COUNTER

FIGURE:16 B

Control Inputs: 1) LE-LATCH ENABLE, 2) $\overline{OE}$ - OUTPUT ENABLE

16 BIT TRANSPARENT LATCH (With 3 state Outputs)

FIGURE:17

CONTROL INPUTS:

- 1) $\overline{E1}$
- 2) $\overline{E2}$

Make $\overline{E2}=0$, to have $\overline{E1}$ as control pin.

18 BIT 3-STATE BUFFER

(With common 2-input NOR Enable)

IC1 to IC6 are 74LS153

Control Inputs:

- 1) S0
- 2) S1
- 3) $\overline{E}$

12 BIT WIDE 4-INPUT MULTIPLEXER

FIGURE:19

Latches

Figure 17 shows the circuit diagram for a 16 bit transparent latch with 3 state outputs. When LE is high, data is transparent. When LE is low the data meeting the setup time is latched. Data appears on the bus when $\overline{OE}$ (output-enable) is low. When $\overline{OE}$ is high, the output is in the high impedance state.

Buffers

Figure 18 shows three 74LS365A chips cascaded to obtain an 18 bit 3-state buffer, with common 2 input NOR enable. By grounding one of these pins, the other pin can be used as the enable buffer pin.

Multiplexers

Figure 19 shows the 12 bit wide 4 input multiplexer using 74LS153 chips. If 3 state output is required, 74LS253 can be used which is completely pin compatible with 74LS153. $\overline{E}$ (enable) can be grounded in our application. If $\overline{E}$ is high, then output is "low" irrespective of S0, S1. If some future use is predicted then E can be included in the micro-instruction.

4.6 Alternative Configurations

Different configurations of the digital signal processor are presented in this section, describing the advantages and disadvantages of each architecture.

4.6.1 Block Processing

An architecture for block processing is presented in this section. Two alternatives are proposed, one using a single block of memory and the other using separate blocks of memory for data and coefficients, etc.

4.6.1.1 Single Memory Block Configuration

Figure 20(a) shows the hardware architecture configuration for block processing using a single memory. Figure 20(b) shows the parameter block used in explaining the working of architecture. Figure 21 shows the same circuit in a simplified block diagram way. Here the program, data, and coefficients reside in the same block of memory. The output results are sent to the output register. If the results are required to be stored back, the microprogram changes slightly then only a micro-sequencer is used, without a bit slice ALU. In the event the application does not need an ALU this architecture is advised.

The start address of parameter block is assumed to be in MAR and LATCH. (A fixed location is to be allotted for this purpose - then it can be hardwired and loaded into MAR at the end of every instruction (MACRO), so that it points to the current PB). The micro instructions for execution of

$$\sum_{i=1}^n a_i x_i$$

are given below. The MAC chip forms the basic computational component.

```
; JMAP, CONT, etc. are 2910 sequencer instructions.
; DP = Data Pointer
; CP = Coefficient Pointer
; MAR = Memory Address Register
; PE = Processing Element
; Micro Instructions are given below
1 JMAP: Mem→Box A; check for an instruction
2 CONT, INR MAR
3 CONT, LOAD DP, INR MAR
4 CONT LOAD CP, INR MAR
5 PUSH COND LD CNTR
6 CONT, MOV ((DP))→PE
7 CONT, MOV ((CP))→PE, INC DP
8 REPEAT LOOP CNTR≠0, INC CP, (INC DP can be done here
                                also) LATCH OUT REG.
```

```
LOAD 'MAR', ((MAR))=LOAD 0 (signalling)
```

```
JMAP or JMAP, INR MAR, if (PBS) instructions are in
sequence
so the microprogram length = 8 lines of code.
```

HARDWARE ARCHITECTURE FOR BLOCK PROCESSING (SINGLE MEMORY CONFIGURATION)

BLOCK DIAGRAM FOR BLOCK PROCESSING

(Single Memory Configuration)

FIGURE:21

computation time = $(5+3n) 0.2 \mu\text{sec.}$

$(n= 1, \dots, 2^{12})$

Assuming 200 ns ($0.2 \mu\text{s}$) cycle time.

computation time = $(5+3n) 0.2 \mu\text{sec.}$ So say we have to compute 50 KHZ sample rate what is the block length

$$(5+3n) 0.2 \times 10^{-6} = \frac{1}{50 \times 10^3}$$

$$n = 95/3 = 31$$

So we can go up to 31 tap FIR filter for a 50 KHZ sample rate.

In the microprogram memory, at the location 0, JMAP instruction should be there and in the PB where MAR is pointing to start with, a '0' should be there, so that 2910 sequencer will loop and wait till the host CPU sets up a PB, and once the PB is set up it starts executing the given instruction. Clearly as a single memory is used the MAC is idle for a while. All the memories shown are dual ported. The processing element shown is a MAC (Multiplier Accumulator chip) and is described below.

The MAC:

The TRW multiplier accumulator chips are low power high speed TTL LSI devices. They are capable of performing 16x16 bit multiplication and product accumulation. The numerical system can be either two's complement or unsigned magnitude. The outputs can be either two's complement or unsigned magnitude. The output contents can be added to or subtracted from the next product, or the accumulate function can be disabled for multiply only. Initial data can be preloaded into the output register directly. For more details see the TRW data sheets.

The MAC chip requires 11 control pins such as enable CLK X, CLK Y, CLK P, and ACC, RND, SUB, TC, PREL, TSL, TSM, TSX. Some of them can be permanently chosen, (such as TC, TWO's complement) but for the sake of generality they are included in the microcode.

Figure 22 shows the microinstruction word with detailed explanation. The 2910 sequencer needs 21 bits, the MAC chip 11 bits, etc. Some of them can be avoided as explained earlier.

4.6.1.2 Multi Memory Block Configuration

This architecture is similar to the one discussed in 4.6.1.1 except that separate memory blocks are used to store programs, data and coefficients. Once again it is stressed that all the memory blocks used are qual ported but are shown only linking with DSP as the DSP functions are explained here. Some of the micro bits and buffers used to control the output of MAR, DP, CP, etc., are not required now as we have separate busses for each memory. Figure 23 shows this configuration. Micro instructions for executing

$$\sum_{i=1}^n a_i x_i$$

are given below.

```
J MAP, MEM (MAR) => BOX A ; check for an instruction
CONT, INR MAR
CONT, LOAD DP, INR MAR
CONT, LOAD CP, INR MAR
PUSH COND LD CNTR
CONT, MOV ((DP)) => PE, MOV ((CP)) => PE
REPEAT LOOP CONTR≠0, INC CP, INC DP, Latch out Reg
LOAD MAR (with latch out) & MEM (MAR) => 0 (signalling)
JMAP.
(JMAP, INR MAR, if instructions are in sequence)
```

then the microprogram length = 7 instructions
 computation time = (5+2n) 0.2 μs

Clearly one microinstruction is reduced in the inner loop and this saves a lot of time when evaluating large sequences.

HARDWARE ARCHITECTURE FOR BLOCK PROCESSING(MULTIMEMORY CONFIGURATION)

FIGURE 23A

BLOCK DIAGRAM FOR BLOCK PROCESSING

(MULTI MEMORY CONFIGURATION)

FIGURE: 23B

Assuming 50 KHZ sample rate, what is the block length?

$$(5+2n) 0.2 = 20$$

$$2n = 95 \text{ or } n = 47$$

So we can go up to 47 tap FIR filter with this architecture for a 50 KHZ sample rate. Compare this with 31 tap FIR filter earlier obtained with single memory block configuration.

4.6.2 Stream Processing

A modified architecture for stream processing is presented in this section. Two alternatives are proposed, one with single memory block to store programs, data and coefficients, and the other using separate memory blocks to store programs data and coefficients. These configurations are not exclusively meant for stream processing. They can also do a simple block processing. They are presented separately for easy understanding of complicated hardware development.

4.6.2.1 Single Memory Block Configuration

Hardware architecture for stream processing using single memory is shown in Figure 24. ADC loads the new sample value called NS (for New Sample) into NS register. In stream processing what we need is data transfer, specially an up or down shift by one word of all data words in a specific field, thereby inserting one new data word and deleting the oldest. In hard wired logic, this is generally done with shift registers. In programmable logic it can be implemented by a modulo arithmetic address calculation. Doing this requires a time consuming software solution or additional hardware. Both approaches are not suitable for the high speed and low cost system we are seeking here. A special structure capable of doing this job efficiently is shown in Figure 24. This is achieved by using two temporary registers TR1, TR2 connected as shown. This two register feedback path connects the output of the RAM to its input working as a FIFO structure and allowing RAM to work as a shift register. This is necessary because, unlike a register, a RAM cannot be read from and written into at the same time, so shifting data from RAM location j to $j+1$ requires that the data in location $j+1$ be read out of RAM before the data from location j can be

HARDWARE ARCHITECTURE FOR STREAM PROCESSING

(SINGLE MEMORY CONFIGURATION)

FIGURE 24

written into it, and this means that for some time the data from both locations must be held outside RAM and hence the need of TR1 and TR2. Overall box "B" is the extra hardware added to the previous one shown in Figure 21, an approach like this is used in the $\mu\text{pd}7720$ [also ref. 1].

The x_i input to processing element (PE) is from the output of TR1, thus allowing both A_i and X_i to be available simultaneously to PE. The PE need not wait as both elements are available at the same time. The extra hardware when compared to Figure 21 add up few more micro control bits (6) and is shown in Figure 22(b). Other mechanism is assumed same as explained in Section 4.6.1. Microprogram for

$$\sum_{i=1}^n a_i x_i$$

is shown below:

```

STRT:  JMAP ; check for an instruction (MAR) ==> Box A
        CONT, INR MAR
        CONT, LOAD DP, INR MAR
        CONT, LOAD CP, INR MAR

        PUSH/COND LD CNTR, MOV NS ==> TR1
        CONT, MOV (TR1) ==> PE, also to TR2, MOV ((CP)) ==> PE
        CONT, MOV ((DP)) ==> TR1 INC CP
        CONT, MOV (TR2) ==> ((DP))
        RPT LOOP, CNTR#0, INC DP

        CONT, LOAD (MAR = LATCH). MEM(MAR) = 0 data
                                                (signalling)
        JUMP, STRT

```

TOTAL = 11, Micro instructions
 inner loop = 4 instructions

computation time per block of data = $(5+4n) 0.2 \mu\text{s}$

If 50 KHZ data is to be processed (20 μs time period)
 $(5+4n) 0.2 = 20$
 $4n = 95$
 $n = 95/4 = 23$

Therefore FIR filter with 23 taps can be implemented at 50 KHZ input signal sample rate.

4.6.2.2 Multimemory Block Configuration

In Section 4.6.2.1 we have seen a novel configuration for data transfers required in a stream processing environment. Now its slightly modified version is described in this section. The modification is mainly in the memory organization. Here individual memories are used to store data, program and coefficients. Figure 25 shows this configuration as a block diagram. The micro program for

$$\sum_{i=1}^n a_i x_i$$

is given below.

```

STRT:  JMAP  MEM(MAR) ==> BOXA; check for an instruction
        CONT, INR MAR
        CONT, LOAD DP, INR MAR
        CONT, LOAD CP, INR MAR

        PUSH/COND LD CNTR, MOV NS ==> TR1
        CONT, MOV (TR1) ==> PE, MOV((CP)) ==> PE
        CONT, MOV ((DP)) ==> TR1, INC CP
        CONT, MOV (TR2) ==> ((DP))
        RPT LOOP CNTR != 0, INC DP
        CONT, LOAD MEM (MAR latch) = 0 data
        JUMP, STRT

```

Here we could not achieve any reduction in the number of microinstructions. Hence, unless large data memory is required (of the order of 64K words) we need not go for individual memories for data, coefficients and programs. This is due to the novel configuration of using TR1 and TR2. We are able to send both x_i and a_i to the processing element. But if we are just processing blocks of data this new architecture will be useful as we do not need any data transfers, as it was proved in Section 4.6.1.2. We can add an ALU to the box A in our diagrams to make the DSP more flexible. That is discussed in the next section.

FIGURE 25

HARDWARE ARCHITECTURE FOR STREAM PROCESSING

(MULTI MEMORY CONFIGURATION)

4.6.3 DSP with ALU

As explained in the previous section adding an arithmetic logic unit to the DSP will enhance its capabilities, and make it more flexible. Complete independency of the DSP over the host CPU for logical and arithmetic computations is also achieved by this, which is necessary for any serious DSP work. We can add a 2901 based RALU (Register and Arithmetic Logic Unit) or 2903 based RALU for more flexibility and more complex computational requirement. The ALU was not added to the previous DSP configurations to avoid complexity at the early stages.

The Am 2903 based ALU is presented in this section (as 2901 is only a single input port device) and how this can be attached to the previously designed DSP is discussed later. For more details about the Am 2903 refer to 'manufacturers' data sheets. Figure 26(a) shows the 16 bit ALU with look-ahead carry based on Am 2902 super slices. Forgetting the internal connections, it can be shown in Figure 26(b) as a block diagram, having two 16 bit data inputs and 16 bit output. When it is incorporated into the DSP, the DSP will be capable of doing arithmetic, logical and shifting operations and thus enhancing its capabilities. The DSP units ALU is shown in Figure 27. It has all the capabilities of the DSP discussed earlier with added ALU capabilities. This ALU can be added to any one of the DSPs discussed earlier. The microinstruction bits needed for the ALU are shown in Figure 28.

4.6.4 Super DSP

The super DSP hardware configuration is shown in Figure 29(a) and Figure 29(b) gives the extra microinstruction bits required for this configuration. Added to all the features discussed earlier, it has extra address manipulation techniques. The address counters are split into two 8 bit counters instead of one single 16 bit counter. Using them as column and row addresses, matrix manipulation algorithms can be efficiently computed. Row and column address can be selected from the instruction data bus (IDB), program data bus, A bus or B bus. Row and column addresses can be incremented/decremented separately. This is the feature which will help immensely in matrix manipulation/computational algorithms. All the other features discussed

DATA A BUS DATA B BUS

(A)

FIGURE:26 16-BIT ALU

Y OUT PUT BUS (B)

FIGURE:27 ALU BASED DSP

MICRO INSTRUCTION BITS FOR ALU

FIGURE 28

EXTRA MICRO INSTRUCTION BITS FOR SUPER DSP

FIGURE: 29B

earlier are retained. TR1 (ZA) and ZB register/latch acts as a delay element. We can store the value acquired in the previous cycle. Thus the overall flexibility comes from the multiplexers and microprogrammed controller and the dual port memory which allow the basic inner product calculation and matrix manipulation in an efficient manner. It can be configured and controlled to be used in a wide variety of possible digital signal processing algorithms. The output of MAC can be fed back into both the memories, which allows the programming of recursive algorithms and temporary storage of immediate results. Provision to store DP (Data Pointer) value is also provided. This is very useful in some cases. See example of FFT computation. Because of all these facilities, it is referred to as the Super DSP. Examples of microinstructions for IIR filter, FFT, complex butterfly, etc., are presented below.

Examples of Computation of a Butterfly:

A butterfly is a basic computational element in an FFT. A microcoded algorithm to implement it on the Super DSP is given below. Figure 30(a) shows the basic butterfly. It requires one complex multiplication and two complex additions. While computing the butterfly DPR and DPC are concatenated to work as a single DP of 16 bit length. This is done using CONCAT signal (microinstruction bit) as shown in Figure 30(c). When $CONCAT = 1$, $INC\ DPC$ means $INC\ DP$; when $CONCAT = 0$, we can increment DPC and DPR separately by $INC\ DPC$ and $INC\ DPR$. Figure 30(c) shows how to use two separate 8 bit counters as DPC and DPR and concatenate them to get a single 16 bit counter. Figure 30(b) shows the memory values and pointer locations before and after butterfly computation. The results are stored back at the same location for in place computation of FFT. Two micro algorithms are given below, one using MAC for some additions and other using MAC as a multiplier only.

Micro Algorithm (a):

[MAC is used for some additions also]. Initially (R15) = (DP), (R16) = 4 is assumed.

Basic Butterfly used in FFT

Figure: 30 (a)

Before butterfly computation

After butterfly computation

Memory values and pointer locations

Figure: 30 (b)

Concatenation of DPR and DPC

Figure: 30 (c)

<u>Clock Cycle</u>	<u>Action Performed</u>
T ₁	Get x_1, a_1 , store in ZA, ZB also in MAC chip, $x_1 a_1$ in MAC accumulator
T ₂	Get x_1', a_1' onto their A and B busses, also into MAC and $-x_1' a_1'$ to accumulate - so MAC output = $(x_1 a_1) - (x_1' a_1')$
T ₃	store $(x_1 a_1 - x_1' a_1')$ in R ₁ register of ALU; via A-bus (R ₁ = real part)
T ₄	Get x_1', a_1 to MAC - preload "0", get $x_1' a_1$
T ₅	Get x_1, a_1' to MAC and get $(x_1' a_1 + x_1 a_1')$
T ₆	Store result in R ₂ of accumulator (R ₂ imaginary part)
T ₇	Get x_2 to R ₃ in ALU per form $(R_3 + R_1)$ in ALU put in R ₅ ($=y_1$)
T ₈	Get x_2' to R ₄ in ALU, per form $(R_4 + R_2)$ in ALU, put in R ₆ ($=y_1'$)
T ₉	$(R_3 - R_1)$ in R ₇ ($=y_2$)
T ₁₀	$(R_4 - R_2)$ in R ₈ ($=y_2'$)
T ₁₁	Load DP with previous address
T ₁₂	Load y_1 , INC DPC
T ₁₃	Load y_1' , INC DPC
T ₁₄	Load y_2 , INC DPC
T ₁₅	Load y_2' , INC DPC
T ₁₆	$R_{15} = R_{15} - R_{16} = DP - 4,$
T ₁₇	Load DP = R ₁₅ for next set of FFT

Results

For 1 FFT butterfly, with in place computation it needs 15, "T" cycles, i.e., $15 \times 2 = 3 \mu s$.

Micro algorithm (b) (MAC used as a multiplier)

Initially when loading DP (used in concat mode) same value is also loaded in R15 in ALU. At the start, (R15 = (DP),

START:

<u>Clock Cycle</u>	<u>Action Performed</u>
T ₁	Get x_1, a_1 ; store in ZA, ZB, load in MAC to get $x_1 a_1$ at the output
T ₂	Load (R ₁) = $x_1 a_1$, INC DPC INC CPC
T ₃	Get $x_1' a_1'$ onto A and B bus, load MAC get $x_1' a_1'$
T ₄	Load (R ₂) = $x_1' a_1'$
T ₅	Load MAC with x_1', a_1 ; get $x_1' a_1$
T ₆	Load (R ₃) = $x_1' a_1$
T ₇	Load MAC with x_1, a_1' , get $x_1 a_1'$
T ₈	Load (R ₄) = $x_1 a_1'$, INC DPC
T ₉	Perform $R_1 = R_1 - R_2 = x_1 a_1 - x_1' a_1'$
T ₁₀	Perform $R_3 = R_3 + R_4 = x_1' a_1 + x_1 a_1'$
T ₁₁	(R ₂) = x_2 (put in ALU)
T ₁₂	(R ₄) = x_2' (put in ALU)
T ₁₃	$R_5 = R_1 + R_2 = y_1$; DEC DPC
T ₁₄	$R_6 = R_3 + R_4 = y_1'$; DEC DPC
T ₁₅	$R_7 = R_2 - R_1 = y_2$; DEC DPC
T ₁₆	$R_8 = R_4 - R_3 = y_2'$; DEC DPC

[Now Butterfly is computed; it should be stored at the same place as x_1 , x_1' , x_2 , x_2']. How to do that is shown below. Note, DP now points to x_1 .

T₁₇ Load (R₅) = y, at pointed by DP, i.e.,
 ((DP)) = (R₅) = y₁, INC DP

T₁₈ Load ((DP)) = R₆ = y₁ ; INC DP

T₁₉ Load ((DP)) = R₇ = y₂; INC DP

T₂₀ Load ((DP)) = R₈ = y₂'; INC DP

In place computation of butterfly is over now the DP and CP point to next set of values, to perform another butterfly.

Results:

This method requires 4 μ s for (20 x 0.2 = 4 μ s) butterfly computation.

IIR FILTER Example.-

An example of microcode for IIR filter is given below. First an example for 2nd order IIR filter section is given. Then a generalized example is presented.

Case 1: Second order IIR FILTER

A recursive second order section is described by

$$y(k) = -C_1 y(k-1) - C_0 y(k-2) + b_2 x(k) + b_1 x(k-1) + b_0 x(k-2)$$

MEMORY STORAGE ORGANIZATION AND PARAMETER BLOCK
(2nd ORDER IIR FILTER)

FIGURE:31A

PARAMETER BLOCK

INSTRUCTION
START ADDR. DM
START ADDR. CM
M
N

PARAMETER BLOCK FOR GENERALIZED IIR FILTER

FIGURE:31B

The above described recursive section can be implemented on super DSP as presented below.

```

STRT: JMAP; check for an instruction
      CONT, INR MAR
      CONT, LOAD DP, INR MAR
      CONT, LOAD CP, INR MAR

      ; output of MAC is y(k-1)

      CONT, y(k-1) → MAC also to R0 in ALU
      ((CP)) = C1 → MAC, y(k-2) = ((DP)) → ZA,
      MOV (NS) → TR1; MAC out will be y(k-1)C1

      CONT (R0) = y(k-1) → ((DP)), INC CP,
      TR1 → TR2

      CONT, (ZA) → MAC, ((CP)) = C0 → MAC, INC DP
      CONT, (TR2) → MAC, ((CP)) → MAC
      CONT, ((DP)) → (TR1), INC CP
      CONT, (TR2) → ((DP))
      CONT, (TR1) → (TR2) also MAC, ((CP)) → MAC, INC DP
      CONT, ((DP)) → (TR1), INC CP
      CONT, (TR2) → ((DP))
      CONT, (TR1) → (TR2) also MAC, ((CP)) → MAC
      ; MAC out = y(k)
      CONT, LOAD MEM (MAR=Latch)=0 data,
      MAC → OUT

      JUMP STRT

```

TOTAL = 16 instructions.

Time of execution = $16 \times 0.2 = 3.2 \mu\text{sec}$.

The necessary storage organization and the parameter block for this case are shown in Figure 31(a).

Case 2: Generalized IIR Filter Example

A generalized IIR filter can be completely summarized by the difference equation

$$y(n) = \sum_{k=0}^n a(k) x(n-k) - \sum_{k=1}^m b(k) y(n-k)$$

Implementation of this difference equation on super DSP is presented below.

STRT. JMAP; check for an instruction

CONT, INR MAR

CONT, LOAD DP, INR MAR

CONT, LOAD CP, INR MAR

PUSH/COND LD CNTR, MOV MAC OUT TO TR1, TR2; M is pushed

CONT, MOV (TR2) → MAC, MOV ((CP)) → MAC

CONT, MOV ((DP)) → TR1, INC CP

CONT, MOV (TR2) → ((DP))

RPT LOOP, CNTR ≠ 0, INC DP

INR MAR

PUSH/COND LD CNTR, MOV NS TR1 N is pushed

CONT, MOV (TR1) → TR2 also MAC, MOV ((CP)) → PE

CONT, MOV ((DP)) → TR1, INC CP

CONT, MOV (TR2) → ((DP))

RPT LOOP, CNTR ≠ 0, INC DP

CONT, LOAD MEM (MAR=LATCH)=0 data, MAC → OUT

JUMP, STRT

The parameter block used is shown in Figure 31(b).

Time of execution = $(9 + 4(M+N)) \times 0.2 \text{ } \mu\text{sec}$.

For the second order filter described in Case 1 above, $M = 1$, $N = 2$.

Time of execution = 21×0.2
= $4.2 \text{ } \mu\text{sec}$.

FIGURE 32 16-POINT FFT FLOW GRAPH

FFT. Computation Example:

An example of computing a 16-point FFT on the super DSP is presented below. Previously described butterfly computation is used here without rewriting the whole microprogram. Flow graph of the 16-point FFT is given in Figure 32 for easy understanding of the microprogram. Basically, the simple butterfly described earlier is computed many times with different data values located at different memory locations.

Microinstructions for 16-point FFT:

STRT: JMAP; check for an instruction

CONT, INR MAR

CONT, LOAD DP, INR MAR

CONT, LOAD CP, INR MAR

CONT, LOAD R₁₅ with (DP);

JSRP: BFLY ; 1st pass is over

; now data values are to be rearranged for the next pass computation.

; the algorithm is to bring 2 sample values to be interchanged to ALU and store them back in the required shuffled way.

; after the 1st pass locations to be interchanged are (2,3), (6,7), (10,11), (14,15).

; to do this we have saved our start address of DP in R₁₅. Let us also assume R₁₄ = 4 is there.

; so to start with R₁₅ = start addr of DM
R₁₄ = 4

; before calling SHUFFLE routine we should place proper addresses in R₀, R₁.

CONT, R₀ = R₁₅+1 (i.e., 2)

CONT, R₁ = R₀+1 (i.e., 3)

JSRP SHUFFLE

; now RPT LOOP CNTR should be loaded with 3 via
MUX, etc., from microinstruction word

PUSH/COND LD CNTR (=3)

CONT, R0 = R0+R14

CONT, R1 = R1+R14

JSRP : SHUFFLE

REPEAT LOOP CNTR $\neq$ 0

; shuffled data is ready now

call BFLY now

JSRP : BFLY; 2nd pass is over.

; now again shuffle the data and repeat as above

; the samples to be interchanged now are (2,5),
(4,7), (10,13) and (12,15).

CONT, R0 = R15 + 1 (=2)

CONT, R1 = R15 + R14 (=5)

JSRP : SHUFFLE

CONT, R0 = R1-1 (=4)

CONT, R1 = R0+R14-1 (=7)

JSRP : SHUFFLE

CONT, R0 = R1+R14-1 (=10)

CONT, R1 = R0+R14-1 (=13)

JSRP : SHUFFLE

CONT, R0 = R1-1 (=12)

CONT, R1 = R0+R14-1 (=15)

JSRP : SHUFFLE

shuffled data is now available.

JSRP : BFLY ; 3rd pass is over

; the samples to be interchanged after the 3rd
pass are (2,9), (4,11), (6,13), (8,15)


```

CONT, R0 = R15+1 (=2)
CONT, R1 = R15+R14
CONT, R1 = R1+R14 (=9)
JSRP : SHUFFLE
; R13=2 is assumed here

```

```

PUSH COND LD CNTR (=3)
CONT, R0 = R0 + R13
CONT, R1 = R1 + R13
JSRP : SHUFFLE

```

```

REPEAT LOOP CNTR  $\neq$  0
; shuffled data is now available
JSRP : BFLY ; the last pass is over

```

```

; the subroutines used are presented now

```

```

; BFLY subroutine is equivalent to a pass, when
data values are properly arranged

```

```

; it takes  $8 \times 20 = 160$  micro cycles.

```

```

BFLY : PUSH/COND LD CNTR (=8)
Basic butterfly micro instructions given in the
earlier example

```

```

REPEAT LOOP, CNTR  $\neq$  0
CRTN

```

```

; the SHUFFLE subroutine shuffles the data values
; the addresses of values to be shuffled are
; assumed to be available in R0, R1.
; it takes 13 micro cycles.

```

```

SHUFFLE: CONT, (R0)  $\rightarrow$  DP
CONT, ((DP))  $\rightarrow$  R2, INC DP
CONT, ((DP))  $\rightarrow$  R3 ; (R2=X1, R3=Y1)
CONT, (R1)  $\rightarrow$  DP
CONT, ((DP))  $\rightarrow$  R4, INC DP
CONT, ((DP))  $\rightarrow$  R5 ; (R4=X2, R5=Y2)

```

```

CONT (R0)→DP
CONT, (R4)→((DP)), INC DP
CONT (R5)→((DP)), (X2,Y2 replaced at
                    X1Y1)
CONT, (R1)→DP
CONT, (R2)→((DP)), INC DP
CONT, (R3)→((DP)), (X1,Y1 replaced at
                    X2Y2)
CRTN

```

Total number of micro cycles = 834

Time of execution = $834 \times 0.2 \mu\text{sec}$
 = 0.1668 milliseconds

Computation time of a 16 point complex FFT = 0.1668
 .milliseconds

4 6.5 Estimation of Efficiency in different architectures:

If a meaningful statement about the architectures of digital signal processors and their utility in specific applications is to be made, an estimation or measurement must be made of their relative performance.

The term estimation (or efficiency estimate/utilization estimate) is more appropriate because, in many cases, insufficient data or a sufficient probability of error exists that measurement, per se, of the efficiency is not possible.

Utilization as an estimator of architectural performance has been chosen since it allows architectures to be characterized without the requirement to provide correction factors for particular technologies or functional element performance (i.e., actual multiply or add time).

As various signal processing algorithms lead finally to the calculation of inner products, the following definition of efficiency (utilization) is chosen. (90)

$$\text{Utilization} = \frac{\text{Theoretical minimum execution time} \times 100\%}{\text{Actual execution time}}$$

$$= \frac{T_{\min} \times 100\%}{T_{\text{act}}}$$

where

T_{act} = The actual execution time

$T_{\min} = K_M T_M + K_A T_A$, wherein

K_M = number of multiplies

T_M = multiply time

K_A = number of additions

T_A = addition time

As explained earlier, the general form of most digital signal processing problems requires iterative computation of the sum of products hence the utilization in evaluating

$$Y = \sum_{i=1}^N a_i x_i$$

is calculated.

From section 4.6.1.1 for single memory block configuration while doing block processing we have

$$T_{act} = (5 + 3N)T \quad ; T = \text{One clock cycle period}$$

In our case $T_A = T_M = T$

$$K_M = N$$

$$K_A = N-1$$

therefore $T_{min} = (2N-1)T$

$$\text{Utilization} = \eta = (2N-1)/(3N+5)$$

For large values of N , this approaches to 66.6%

From section 4.6.1.2, for multimemory configuration while doing block processing, we can similarly show that,

$$\text{Utilization} = (2N-1)/(2N-5)$$

this approaches 100% for large values of N . This is the advantage of using multimemory configuration, which is also used in the super DSP configuration. Utilization graphs for these two configurations are shown in Figure 33A.

For stream processing, from section 4.6.2, it can be seen that the utilization is same for both single and multimemory configurations and is given by

$$\text{Utilization} = (2N-1)/(4N-5)$$

which approaches 50% for large values of N . (See Figure 33B). This lesser efficiency is due to the loss of clock cycles in rearranging data values.

The utilization values for the super DSP architecture are presented below for various digital signal processing algorithmic evaluations.

Complex Butterfly:

K_M = number of real multiplies = 4

K_A = number of real additions = 6

$T_M = T_A = T$

therefore $T_{min} = 10T$, and from section 4.6.4 we have $T_{act} = 15T$, hence

Utilization = $(T_{min}/T_{act}) \times 100\% = 66.6\%$

Second order IIR filter:

Here we have $K_M = 5$ and $K_A = 4$ giving $T_{min} = 9T$, also $T_{act} = 16T$, hence

Utilization = $(9/16) \times 100\% = 56.25\%$

Generalized IIR filter:

In this case $K_M = (m+n)$ and $K_A = (m-1) + (n-1) + 1 = m+n-1$ therefore $T_{min} = (2(m+n)-1)T$ and $T_{act} = (4(m+n) + 9)T$, hence

Utilization = $2m + 2n - 1 / 4m + 4n + 9$

This is shown in the graphical form in Figure 33C.

16 Point FFT:

$T_{act} = 834 T$

There are 4 passes and 8 butterflies per pass, ie. a total of 32 butterflies. Each butterfly has 4 real multiplications and 6 additions.

$T_{min} = 32(4 + 6)T = 320 T$, hence

Utilization = $320/834 = 38.37\%$

UTILIZATION GRAPHS FOR BLOCK PROCESSING: 1) Single memory configuration
2) Multiple memory configuration

$$\text{for } Y = \sum_{i=1}^N A_i X_i$$

FIGURE 33A

UTILIZATION GRAPH FOR STREAM PROCESSING: Same for single and multiple memory configuration.

$$\text{for } Y = \sum_{i=1}^N A_i X_i$$

FIGURE 53B

N →

UTILIZATION GRAPH FOR GENERALIZED IIR FILTER: Super DSP configuration

FIGURE 33C

(M+N) →

10 20 30 40 50 60 70 80 90 100 120 140 160 180 200

5. Summary

Because of the advances in VLSI technology some of the functions which could not be implemented in hardware in the early 70's are now implementable, thus removing many of the bottlenecks in real-time digital signal processing. One of the outstanding examples is the TRW multiplier/accumulator chip. Even though advances in the VLSI technology have eliminated to some extent the problems faced by the real time digital signal processing, the traditional problem of bus band-width congestion faced by the array processors still remains. A solution to this problem is presented by making use of a dual-ported memory. The overall system throughput is increased by the use of high-performance components in an optimized architectural design. Various signal processing algorithms lead finally to the calculation of inner products. Various hardware configurations were presented, with details of useage and advantages, to implement this inner product.

A general architecture was presented in Chapter 1. Memory design was dealt with in detail in Chapter 2. The communication protocols were described in Chapter 4. After giving an introduction to block and stream processing, various configurations of the DSP were presented (starting with a simple DSP to a super DSP, which is also useable in adaptive digital signal processing). Even though the system is not build due to lack of funds the design was thoroughly checked. Depending on the availability of funds and perfomance requirement one of the configurations can be chosen for prototyping.

The chip count power consumption and performance for FIR filtering, for different configurations is presented in Table 2. Table 3 gives the execution times of typical algorithms compared to several processors.

TABLE 2

Configuration	Chip Count (approx) #	Power* (approx) watts	Performance	
			50KHz (sample rate) FIR # of taps	Block processing Stream Processing
Single memory block processing	51	7	31	
Multimemory block processing	42	6.75	47	
Single memory stream processing	66	7.5	47	23
Multimemory stream processing	57	7.25	47	23
Super DSP	78	7.5	47	23

*Excluding memory and ALU. ALU board needs 10 chips and 7.5 W power.
64K x 16 Bit wide memory needs 278 chips and 30 W power.

2 2
OF / DE

53. Jake Burma, "Most Microprocessor Compatible ADC's Still Require Extra Support," EDN, May 5, 1980, pp 161-164
54. Michael K. Stauffer, "Math Processor Chips Boost UC Computing Power," EDN, Aug. 20, 1980, pp 113-120
55. Louis Schirm IV, "A Family of High Speed Floating Pt. Arithmetic Chips," IEEE, 1981, ICASSP, pp 374-377
56. John R. Mior, "Bit Slice Devices for Signal Processing," IEEE, 1980, ICASSP, pp 372-375
57. John Eldon, "Single Board Digital Signal Processor," IEEE, 1981, ICASSP, pp 662-665
58. Bernard J. New, "The AM 29500 Signal Processing Family," IEEE, 1981, ICASSP, pp 378-381
59. Robert H. Sperry, "A Microprogrammed Signal Processor," IEEE, 1981, ICASSP, pp 579-582
60. James Brubaker, "14-Bit DAC Mates with UPS, Settles in Less than 1 US," Electronic Design, Apr. 1981, pp 147-151
61. Lawrence Lowe, "Multiplexed DMA with the MC6800," Microprocessors and Micro Syst. (MPMS), May 1979, pp 185-187
62. A.M - Ahmed, et al, "Design and Construction of an Array Processor," MPMS, April 1980, pp 83-88
63. Dogan Ibrahim, "Design of a Microprogrammable Computer with Bit-Slice Devices," MPMS, Mar. 1980, pp 57-62
64. Alan Clements, "Computer System Buses," MPMS, Nov. 1979, pp 413-420
65. Dogan Ibrahim, "Part 2 for 63," MPMS, Apr. 1980, pp 95-100
- 66a. Alan Clements, "Memories in Microprocessor Systems," MPMS, June 1979, pp 227-235
- 66b. Alan Clements, "Memories in Microprocessor Systems," MPMS, July/Aug 1979, pp 284-287
- 66c. Alan Clements, "Memories in Microprocessor Systems," MPMS, Sept 1979, pp 327-336

6. References

1. Architecture

- 1.1 General Architecture, 13, 14, 15, 16, 21, 62
- 1.2 Signal Processor Architecture, 1, 2, 9, 12, 25, 26, 32, 36, 37, 39, 45, 49, 57, 78, 79
- 1.3 Bit Slice Techniques and Controller-Architecture, 42, 56, 63, 65

2. Support Chips

- 2.1 Signal Processing, 5, 19, 23, 30, 34, 44, 54, 55, 58, 73, 74, 75
- 2.2 Microprocessors, 68, 69, 70a, 70b
- 2.3 Multipliers, 77

3. Buses

- 3.1 Bus Structure and Organization, 47a, 47b, 47c, 64, 71
- 3.2 Support Chips, 7, 41

4. ADC and DAC, 7, 53, 60, 67

5. Memory Design, 66a, 66b, 66c

6. DMA, 61

7. General References

- 7.1 Signal Processing, 10, 20, 33, 46, 52, 59
- 7.2 Modems and Microprocessors, 24, 28, 29, 31, 35, 38
- 7.3 Miscellaneous, 3, 4, 6, 11, 17, 18, 22, 27, 40, 43, 48, 50, 51, 72, 76a, b, c, d and e

8. Books and Application Notes, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89

1. H.J. Schloss and G. Ballenberger, "A High Speed Small Scale Processor," Signal Processing, Jan. 1981, pp 29-36
2. H.J. Kolb, "A Signal Processor Using Bit-Slice Elements for the Audio Frequency Range," Signal Processing, Oct. 1980, pp 339-346
3. Dale Mrazek, "Bit-Slice Parts Approach ECL Speeds with TTL Power Levels," Electronics, Nov. 9, 1978, pp 107-112
4. Thomas M. Hedges, "Replacing Hardwired Logic with Microcode," Electronics, Nov. 9, 1978, pp 125-129
5. William F. Arnold, "Now for the Math-Processing Chips," Electronics, July 5, 1979, pp 99-100
6. Morris Krieger "Structured Assembly Language Suits Programmers and Microprocessors," Electronics, January 17, 1980, pp 118-122
7. John A. Posa "Chip Gives Processors Protocols," Electronics, May 22, 1980, pp 167
8. Roger Allan, "The Inside News on Data Converters," Electronics, July 17, 1980, pp 101-112
9. J.E. Merritt, "AMD 2901 - Application to Signal Processing," IEEE 1979, pp 80-84
10. F.J. Goodman and J.J. Welsh, "Digital Signal Processing of the ATMAC Microprocessor," IEEE 1979, pp 73-79
11. "Innovative Design - LSI-11 or 11/23 Number Crunching for under \$4K"
12. George Agule, "CMOS/SOS Microsignal Processor Applications," IEEE 1979, pp 65-72
13. Walter J. Karplus, "Architectural and Software Issues in the Design and Application of Peripheral Array Processors," Computer, Sept. 1981, pp 11-17

14. Alan E. Charlesworth, "An Approach to Scientific Array Processing: The Architectural Design of the AP-120B/FPS-164 Family," Computer, Sept. 1981, pp 18-27
15. Edmund U. Cohler, "Functionally Parallel Architecture for Array Processors," Computer, Sept. 1981, pp 28-36
16. Neil Maron, "Integrating an Array Processor into a Scientific Computing System," Computer, Sept. 1981, pp 41-44
17. Brian Brode, "Precompilation of Fortran Programs to Facilitate Array Processing," Computer, Sept. 1981, pp 46-51
18. Thelma Louie, "Array Processors A Selected Bibliography," Computer, Sept. 1981, pp 53-57
19. Raymond P. Capece, "Digital n-MOS Chip Handles Analog Signals at Record 200 KHZ," Electronics, Dec. 4, 1980, pp 79-80
20. Raymond L. Veenkorut, "MICRO-VECTOR Processors - A Modular Approach to Signal Processing," 1979, IEEE, pp 85-86
21. J.R. Boddie, "Digital Signal Processor - Architecture and Performance," The Bell System Technical Journal, Sept. 1981, pp 1431-1463
22. Paul D. Stigall, "A Performance Study of a Microcomputer - Implemented FSK Receiver," MICRO, Feb. 1981, pp 43-51
23. A. Kanemasa, et al, "An LSI Chip Set for DSP Hardware Implementation," IEEE, pp 644-647, 1981
24. Karl I. Nordling, et al, "Programming a Modem"
25. J.P. Agarwal, et al, "Hardware Considerations in FFT Processors, pp 618-621
26. Abraham Peled, "A Digital Signal Processing System," pp 636-639

27. M. Bellanger "Digital Processing Parameters in Communication Systems," IEEE, 1979, ICC, pp 37.1.1-4.
28. S.A. Madrid, "Implementation of Digital Signal Processing in a Bit Slice Microprocessor," IEEE, 1979, ICC, pp 37.2.1-5
29. Dale M. Walsh, "Microprocessor Controlled 4800 Bps Modem, Lower Cost Versatility," IEEE, 1979, ICC, pp 51.6.1-4
30. JK.R. Boddie "An Integrated Circuit Digital Signal Processor," IEEE, 1980, ICC, pp 11.1.1-5
31. J.M. de Carvalho, "A Digital Modem Based on Correlation Detection," IEEE, NTC 1980, pp 76.6.1-5
32. Antonio Ruiz, "An Architecture for DSP," IEEE, 1980, pp 137-141
33. A.E. Pione "A General Purpose Digital Filter," IEEE, 1980, pp 129-136
34. Kazuo Murano, et al, "Design and Realization of an LSI Processor for Digital Signal Processing," FUJITSU Scientific and Technical Journal, March 1979, pp 39-53
35. Laurence Polle, "Microprocessors in Modems," CEE, June 1978, pp 22-26
36. Ismail I. Eldumiati, "The DSP Architecture and Applications Overview," IEEE Symposium on Circuits and Systems, 1981, pp 888-890
37. S.A. White, "An Architecture for a High-Speed Digital Signal Processing Device," IEEE Symposium on Circuits and Systems, 1981, pp 893-896
38. Piet J. Van Gerwen, et al, "Microprocessor Implementation of High-Speed Data Modems," IEEE Trans. on Comm., Feb. 1977, pp 238-250
39. J.P. Agarwal, et al, "Hardware Modifications in Radix-2 Cascade FFT Processor," IEEE Trans. on ASSP, April 1978, pp 171-172

40. A. Pomerleau, et al, "Realization of a DFT Module for Incorporation in FFT Processors " Proc. of IEEE, Jan. 1977, pp 173-174
41. James Nadir, "Bus Arbiter Streamlines Multiprocessor Design," Computer Design, June 1980, pp 103-109
42. David Hooley, "Bit Slice Technique Minimizes Microcontroller Cost/Complexity," Computer Design, Oct. 1979, pp 105-113
43. Lynn E. Costlow, "Optimizing Microprocessor I/O Techniques," Computer Design, April 1981, pp 151-160
44. William J. Finn, "LSI Hardware Implements Signal Processing Algorithms," Computer Design, March 1980, pp 137-142
45. J.M. Glass, "A Programmable Signal Processing Architecture," IEEE 1979, ICASSP, pp 702-705
46. B.F. Pfister, et al, "Microprocessor Based Real Time Speech Processor," IEEE, 1979, ICASSP, pp 872-875
- 47a. Carl Warren, "Designers Guide to Micro-Computer Buses," EDN, May 27, 1981, pp 159-173
- 47b. Carl Warren, "Compare Micro-Computer Bus Specs to Find the Bus You Need," EDN, June 10, 1981, pp 141-153
- 47c. Carl Warren, "High-Performance Buses Clear a Path for Future Micro-Computers," EDN, June 24, 1981, pp 157-187
48. Robert H. Cushman, "Digital Processing Tools Present Design Challenges," EDN, May 13, 1981, pp 103-109
49. Robert H. Cushman, "Signal Processing Design Awaits Digital Takeover," EDN, June 24, 1981, pp 119-128
50. Robert H. Cushman, "Digitization is on the Way for FFT Designs," EDN, Aug. 5, 1981, pp 99-106
51. George D. Huffman, "Programmable Memories make Product Redesign Easy," EDN, Jan. 21, 1981, pp 106-117
52. Robert E. Holm, "Implement Complex Analog Filters with a Real-Time Signal Processor," EDN, Nov. 20, 1979, pp 171-180

53. Jake Burma, "Most Microprocessor Compatible ADC's Still. . . Require Extra Support," EDN, May 5, 1980, pp 161-164
54. Michael K. Stauffer, "Math Processor Chips Boost UC Computing Power," EDN, Aug. 20, 1980, pp 113-120
55. Louis Schirm IV, "A Family of High Speed Floating Pt. Arithmetic Chips," IEEE, 1981, ICASSP, pp 374-377
56. John R. Mior, "Bit Slice Devices for Signal Processing," IEEE, 1980, ICASSP, pp 372-375
57. John Eldon, "Single Board Digital Signal Processor," IEEE, 1981, ICASSP, pp 662-665
58. Bernard J. New, "The AM 29500 Signal Processing Family," IEEE, 1981, ICASSP, pp 378-381
59. Robert H. Sperry, "A Microprogrammed Signal Processor," IEEE, 1981, ICASSP, pp 579-582
60. James Brubaker, "14-Bit DAC Mates with UPS, Settles in Less than 1 US," Electronic Design, Apr. 1981, pp 147-151
61. Lawrence Lowe, "Multiplexed DMA with the MC6800," Microprocessors and Micro Syst. (MPMS), May 1979, pp 185-187
62. A.M. Ahmed, et al, "Design and Construction of an Array Processor," MPMS, April 1980, pp 83-88
63. Dogan Ibrahim, "Design of a Microprogrammable Computer with Bit-Slice Devices," MPMS, Mar. 1980, pp 57-62
64. Alan Clements, "Computer System Buses," MPMS, Nov. 1979, pp 413-420
65. Dogan Ibrahim, "Part 2 for 63," MPMS, Apr. 1980, pp 95-100
- 66a. Alan Clements, "Memories in Microprocessor Systems," MPMS, June 1979, pp 227-235
- 66b. Alan Clements, "Memories in Microprocessor Systems," MPMS, July/Aug 1979, pp 284-287
- 66c. Alan Clements, "Memories in Microprocessor Systems," MPMS, Sept 1979, pp 327-336

67. W. Check, "Microcontroller Includes a-d Converter for Lowest-Cost Analog Interfacing " Electronics, May 25, 1978, pp 122-127
68. B. Jeffrey Katz, et al, "8086 Microcomputer Bridges the Gap Between 8- and 16-bit Designs," Electronics Feb. 16, 1978, pp 99-104
69. Robert Garrow, "16-Bit Single-Board Computer Maintains 8-Bit Family Ties," Electronics, Oct. 12, 1978, pp 105-110
70. W. Thomas Adams, "How Bit-Slice Families Compare," (2 parts), Electronics, Aug. 3, 1978, pp 91-98
71. Gordon Force, "Microprocessor Bus Standard Could Cure Designers' Woes," Electronics, June 20, 1978, pp 113-118
72. Timothy Mulrooney, "Microprogramming a Minicomputer for Fast Signal Processing," Electronics, Mar. 16, 1978, pp 136-141
73. NEC - UPD 7720 - Preliminary Manual
74. NEC - UPD 7720 - User's Manual
75. S2811 - Manual
- 76a. Willard Bucklen, Dr. John Eldon, Louis Schiorm, et al, "Designers Guide to DSP - Part 1, Understanding the Fundamentals Signal-Processing Tasks," EDN, March 18, 1981, pp 133-144.
- 76b. Willard Bucklen, Dr. John Eldon, Louis Schiorm, et al, "Designers Guide to DSP - Part 2, Single-Chip Digital Multipliers form Basic DSP Building Blocks," EDN, April 14, 1981, pp 153-163
- 76c. Willard Bucklen, Dr. John Eldon, Louis Schiorm, et al, "Designers Guide to DSP - Part 3, Digital Processing Facilitates Signal Analysis," EDN, April 15, 1981, pp 133-146
- 76d. Willard Bucklen, Dr. John Eldon, Louis Schiorm, et al, "Designers Guide to DSP - Part 4, DACs and Multifunction Chips Enhance your DSP Options," EDN, April 29, 1981, pp 161-171

- 76e. Willard Bucklen, Dr. John Eldon, Louis Schiorm, et al,
"Designers Guide to DSP - Part 5, Digital Correlators
Illustrate DSP Trends," EDN, May 13, 1981, pp 133-143
- 77. L. Schirm, IV, "Multiplier Accumulator Application Notes,"
TRW AP Note, Jan. 1980
- 78. R.J. Karwoski "A Four-Cycle Butterfly Arithmetic
Architecture," TRW AP Note, April 1980
- 79. Louis Schirm, IV, "Packaging a Signal Processor onto a
Single Digital Board," Electronics, Dec. 20, 1979, pp
109-115
- 80. R.J. Karwoski, "An Introduction to Digital Spectrum Analysis
Including a High Speed FFT Processor Design," TRW AP Note,
June 1980

Books

- 81. Rabiner and Gold, "Theory and Application of Digital Signal
Processing," Prentice-Hall, 1975
- 82. B.A. Bowen and W.R. Brown, "VLSI Systems Design for Digital
Signal Processing," Prentice-Hall, 1982
- 83. Stephen P. Morse, "The 8086 Primer " Hayden Book Company,
Inc., 1978
- 84. John Mick and Jim Brick, "Bit Slice Microprocessor Design,"
McGraw Hill, 1980
- 85. Cecil Moore, "Multitasking for the 8086," Intel AP Note, AP
61, July 1979
- 86. George Alexy and Gary Miller, "DMA with the 8257 DMA
Controller," Intel AP Note, AP 48, Feb. 1979
- 87. Joe Barthmaier, "Intel Multibus Interfacing," Intel AP Note,
AP 28A, Jan. 1979
- 88. Direct Memory Access Controller, Hardware Reference Manual,
Intel ISBC 501, Manual Order No. 9800294-02
- 89. Intel Multibus Specification, Intel Manual Order No.
9800683-03

90. Milan Kutcha Developing High-Efficiency Digital Signal Processors from digital signal processing algorithms. - MSDL Tech Note 82-3.
91. B.A. Bowen et al, Analysis and Evaluation of VLSI/VHSI Technology on High Performance Digital Signal Processors - Phase 1B Report. 1981.

END

0 610 6.8 3

FIN