1985

Digital Signal Processing Capabilities of the Fujitsu MB8764

Harold B. Creech
University of Central Florida

DIGITAL SIGNAL PROCESSING CAPABILITIES OF THE FUJITSU MB8764

BY

HARALD BEARDALL CREECH
B.S.E.E., United States Coast Guard Academy, 1977

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Summer Term
1985

ABSTRACT

The Fujitsu MB8764 digital signal processing chip is designed to operate with a machine cycle of up to 10 MHz. The chip's ability to perform a 16-by-16 bit multiply and add operation in one machine cycle makes it a good candidate for real time digital signal processing. Unlike the Intel 2920 the Fujitsu MB8764 does not have an onboard analog-to-digital, digital-to-analog converter. Therefore, this paper will be restricted to the operation of this device with digital data input and output.

The use of the MB8764 as a digital filter is analyzed. Performance limitations due to finite word length, memory size and configuration, and clock rate are considered. The MB8764 capabilities in computing fast Fourier transforms are discussed. Development of a working digital filter with the MB8764 work station is presented.

ACKNOWLEDGEMENTS

I would like to thank Barry Mattox and Michael Gorlicki at Martin Marietta aerospace division for granting me access to the only MB8764 development system in Orlando; to Dr. Fred O. Simons for advising me in my graduate studies; to my wife, Terry, for giving me the time to write this paper; and to Jesus Christ, my savior, who makes all things possible.

TABLE OF CONTENTS

INTRODUCTION

Many microprocessors are available today that are specifically designed for digital signal processing. The Fujitsu MB8764 is one of the newest digital signal processing chips on the market, and has incorporated recent advances in VLSI technology into its design. Two widely used chips that may be compared with the Fujitsu MB8764 are the Intel 2920 and the TMS 320.

A comparison between the Intel 2920 and the Fujitsu MB8764 shows the MB8764 to be a much faster chip with a more extensive instruction set. The Intel 2920 offers 24-bit internal precision which is much better than the 16-bit precision offered by the MB8764. The Intel 2920 also offers an onboard ADC and DAC for analog input and output. The MB8764 accepts digital input output only. Internal RAM and program ROM are much larger in the MB8764 and only the MB8764 permits the external expansion of them.

The TMS 320 is a much closer match to the MB8764 than the Intel 2920. The MB8764 is once again the faster machine with a 0.1 μsec instruction cycle compared to the TMS 320's 0.2 μsec instruction cycle. Specifications from the manufacturers show the TMS320 and MB8764 implementing a second order filter in 2.2 μsec and 0.7 μsec respectively. Both the TMS 320 and the MB8764 use an assembly language level instruction set and neither accepts analog inputs. Internal accuracy of the TMS 320 is 16 bits but its design makes it possible to implement double-precision operations. The design of the MB8764 makes it impractical to implement double precision. The MB8764 offers more than twice as much internal RAM as the TMS 320 but only two-thirds the internal instruction ROM.

The MB8764 can be favorably compared to both of these widely distributed chips. It excels in the area of execution speed but is deficient in its internal accuracy.

DESCRIPTION OF THE MB8764

Introduction

The Fujitsu MB8764 digital signal processing chip is a VLSI, CMOS design optimized to provide high-speed processing with flexible memory operation and input/output operation. Internal and external buses provide 16-bit data transfer, and the ALU provides a 26-bit result to the accumulator. The instruction list provides the chip user with a variety of instructions, most of which are specifically designed to simplify the implementation of digital signal processing functions. Internal memory provides for a program ROM of 1024-by-24 bits, and RAM storage of 256-by-16 bits. Both ROM and RAM are expandable externally. These features are all provided on an 88-pin chip less than 31 mm square. This chapter will describe the basic operation of the blocks that make up the MB8764. Figure 1 is a block diagram of the MB 8764. The material in this section comes from references (1) and (2).

Registers on the chip can be divided into four groups: data registers, counter registers, index/address registers, and flags (see Figure 2). The functions of these registers will be explained in the following sections.

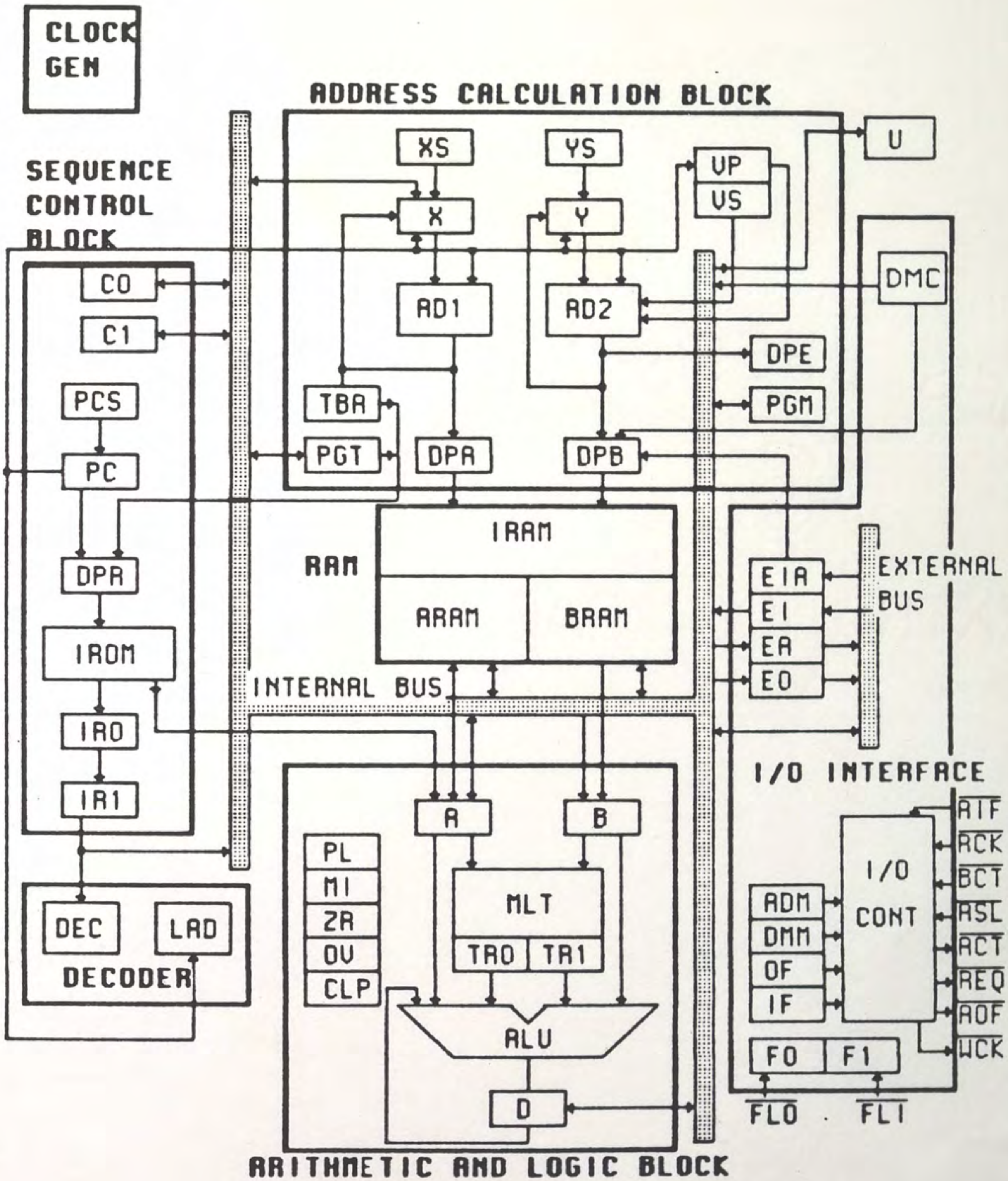
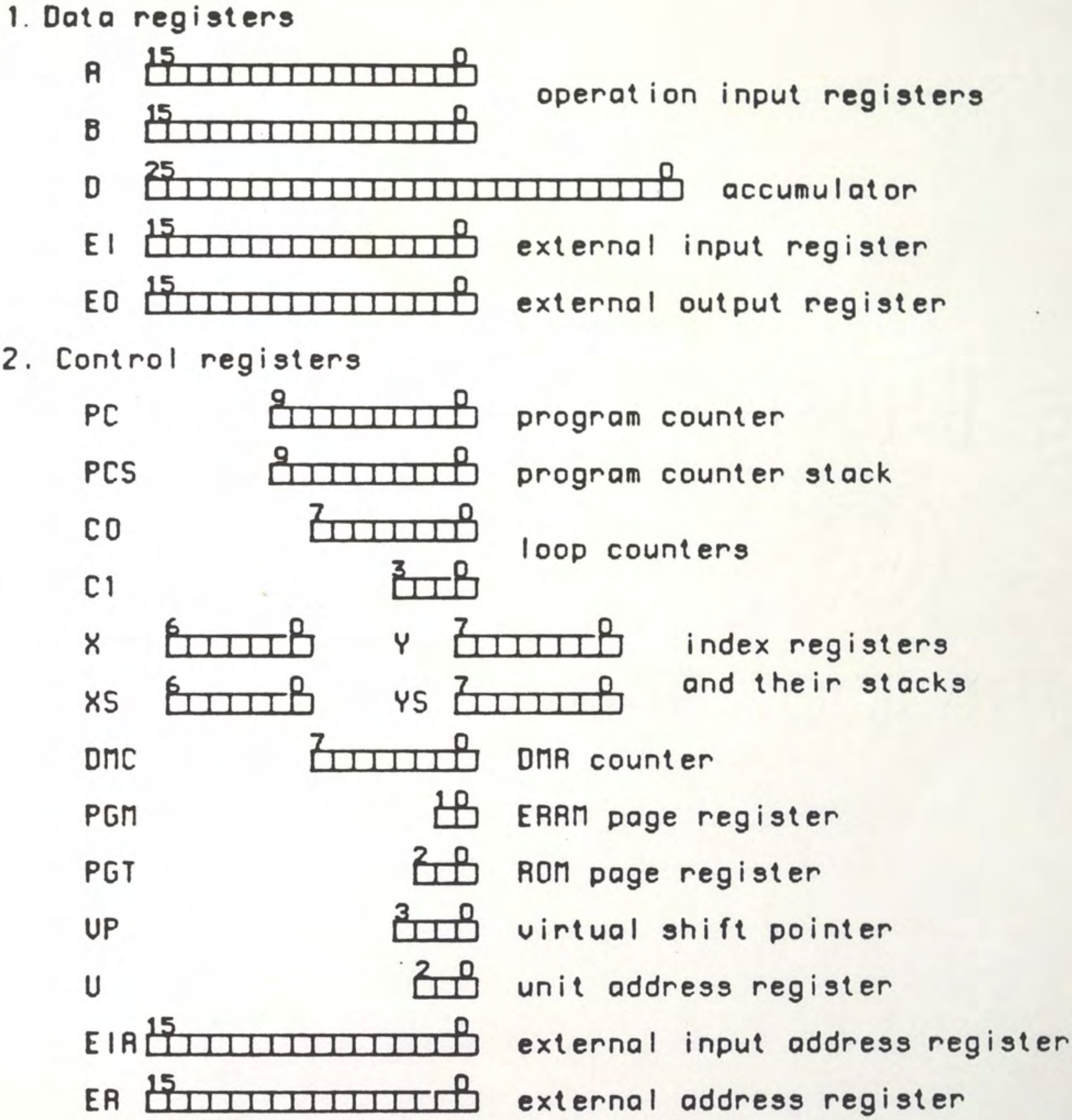


Figure 1. MB8764 Block Diagram.



<u>I/O flags</u>
FO-input flag 0
F1-input flag 1
IF-EI flag
OF-EO flag
DMM-DMA mode
ADM-address mode

<u>ALU flags</u>
PL-D positive
MI-D negative
ZA-D zero
OV-D overflow
CLP-clipping mode

<u>Other flags</u>
VP-vitual pointer mode

Figure 2. MB8764 Registers.

Clock Generator

The clock generator requires an external clock source or a crystal oscillator of 20 MHz or less for its input, and outputs a 50% duty clock source at one-half the input clock frequency. The output is used to time all internal operations; one internal clock cycle equals one instruction cycle. The majority of instructions require one instruction cycle to operate, or 0.1 μsec when using a 20 MHz external clock.

Arithmetic and Logic Block

The arithmetic and logic block accepts input into registers A, B, and D. Instructions in the MB8764 are classified as:

1) Arithmetic or logic instruction, and

2) Control instructions.

Arithmetic and logic instructions are executed in the arithmetic and logic block by the ALU with the exception of multiplication instructions. All arithmetic and logic instructions can be executed together with a control instruction; this type instruction is called a compound instruction. A compound instruction that does not include a multiplication instruction performs: 1) the control instruction specified, and 2) the arithmetic and logic instruction based on the register contents as of the previous instruction cycle. An example is shown below. (Assume B register has $0002 in it.)

Step 1 LDI:NOP #$0001 Put $0001 into the A. No math operation.

Step 2 LDI:ADD #$0005 Put $0005 into A. Add $0001 to $0002.
The D register contains $0003 in step 3.

The multiplication of the contents of register A by register B is performed during each instruction cycle, regardless of the instruction. A multiplier circuit separate from the ALU and using Booth's second-order algorithm performs the multiplication. Booth's algorithm is a simple and direct method for multiplication of signed binary numbers (3). The intermediate results of register A multiplied by register B are stored in temporary storage registers TR0 and TR1. When a multiplication instruction is given, the ALU completes the multiplication by adding TR0 and TR1. The results of multiplying two 16-bit registers would ideally result in a 32-bit number. The ALU provides a 26-bit result to the D register by rounding the addition of the two 27-bit registers TR0 and TR1 and deleting bit 25.

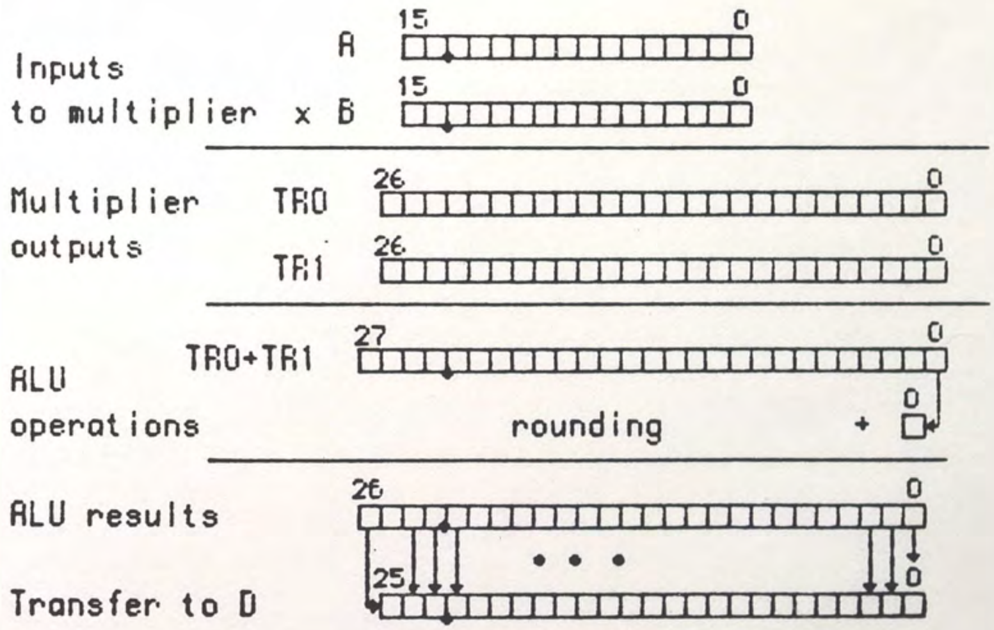


Figure 3. Multiplication.

The round-off causes an error less than plus or minus 2^{-24}. It is necessary to delete bit 25 of the ALU result to obtain the correct two's compliment number. An error results only in the case of -2 x -2 where zero is input into the D register; the overflow flag OV is set to show that an overflow has occurred. A compound instruction that involves a multiplication instruction performs:

1) The control instruction specified, and
2) The multiplication based on the register contents two instruction cycles before.

An example is shown below.

Step 1	LAB:NOP $01,$02	Data is moved from ARAM to A and BRAM to B.
Step 2	LAB:NOP $02,$03	New data is moved into A and B. Step 1 data enters the multiplier circuit.
Step 3	LAB:MUL $03,$04	New data is moved into A and B. Step 1 data multiplication is completed.

In step 4 the register will contain ($A_{step\ 1}$) x ($B_{step\ 1}$).

Division operations in the MB8764 are carried out in the ALU without the help of a specialized circuit. It requires 17 machine cycles to perform division. All other operations performed in the ALU require one machine cycle.

ALU operations are fixed point with the A and B registers having a range from -2 to 1.999938965, and the D register having a range of -4 to 3.999999881. Passing data from the 26-bit D register to the 16-bit internal bus is done as shown in Figure 4.

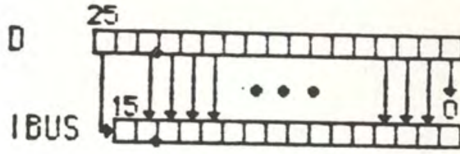


Figure 4. D Register to Internal Bus Transfer.

If bit location 24 in the D register is not zero an error of +/- 2 occurs, and the OV flag is set. The CLP flag, when set, minimizes the error by forcing data transferred to the internal bus to binary 0111111111111111 in the case of a positive overflow, and to 1000000000000000 in the case of a negative overflow.

Sequence Control Block

The sequence control block controls the execution of the program code for the MB8764. Execution is carried out in a pipeline style as shown in Figure 5, a timing diagram of a typical instruction. In the first machine cycle, step one, an instruction from program ROM is placed into the IR0 (Instruction Register zero). During step two preliminary operations are performed based on the instruction in IR0. In step three IR1 (Instruction Register one) receives the IR0 contents, and signals are passed to complete the operation based on the IR1 contents. When step four begins the instruction has completed execution, and results are in place. The steps just outlined are stepped through by a count of the internal clock with interruptions made as necessary for proper program execution.

A 10-bit PC (Program Counter) register addresses the program ROM through the DPR (ROM Pointer Register). The program counter is reset and held at zero when a pulse is sent on the hardware reset

Instruction sequence

```
n-2     LAB:NOP $0,$0
n-1     LAB:NOP $1,$1
n       LAB:MLT $2,$2
n+1     LAB:MSM $3,$3
```

Timing diagram of instruction sequence

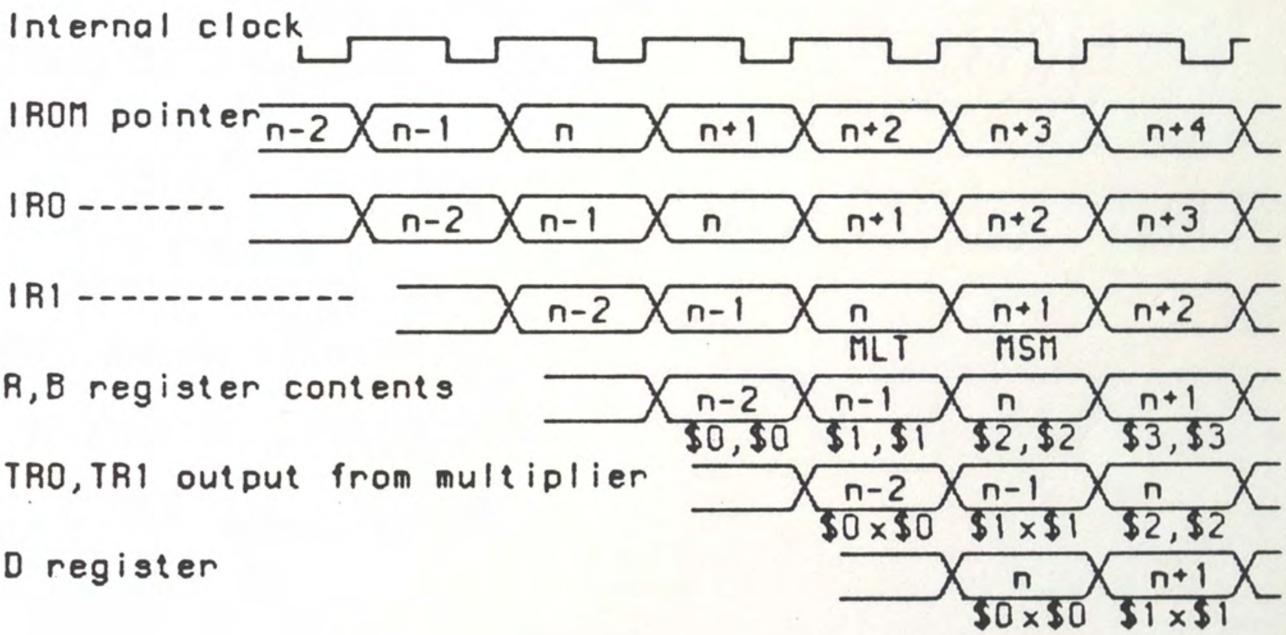


Figure 5. Timing Diagram.

pin RST. Program execution begins with the first clock pulse after the RST pulse is removed; with each internal clock period the program counter is incremented by one, unless interrupted. Interruptions to the program counter incrementing occur when a multicycle instruction is being executed. A cycle counter within the sequence control block determines the proper interruption length. Interruptions in the PC also occur when input/output operations are performed during program operations that use the external data bus or associated registers.

Program execution can be manipulated by changing the PC register value. The following instructions are used to control program execution through PC register execution.

1) Jump, and jump on condition instructions replace the PC contents with the address of the instruction to jump to.
2) Jump to subroutine instructions load the first address of the subroutine into the PC register and save the current PC value in the PCS (Program Counter Stack Register).
3) Return from subroutine instructions return the value stored in the PCS register to the PC register and increment PC.

Nesting of subroutines is not possible using the JSR instruction, because there is only one stack register. Jump instructions can be used for the same end. PC contents can be saved in RAM at the time of a jump, incremented by one and recalled by another jump instruction at the end of the subroutine.

Two loop counters, C1 (eight bits) and C2 (four bits) are located within the sequence control block. They are decremented by one with each pass through the loop and are used with a JOC instruction to control program execution.

Program instructions may be obtained from EROM or IRON, with the status of the IRM pin determining the selection. A switch between internal ROM and external ROM can only be carried out when the hardware reset signal is on, thus IROM and EROM cannot be used in the same program. IROM is a 1024-by-24 bits ROM, EROM is expandable to 4096-by-24 bits with bank switching. IROM is a mask ROM programmed to the designer's specification by Fujitsu. External ROM is not required to be a mask ROM but can be an EPROM, allowing for field production of a design.

ROM can be used to store data in any location wtihin the ROM except location zero. This location must hold an instruction because the PC accesses it after every RST pulse. ROM data is limited to 16-bit words because only the 16 least significant digits of the 24-bit ROM word are read. The 8 high order bits are set to one. The ROM address is specified by a 10-bit input into the DPR from the address calculation block.

Decoder Block

Instructions IR0 and IR1 introduced in the previous section are the inputs to the decoder block. With each increment of the program counter, new data is passed from these registers to the look ahead carry and decode registers respectively. When an instruction code is loaded into the look ahead decoder (LAD) interpretation of the code begins. The number of cycles necessary to complete the instruction is decoded, ALU operations are interpreted, and effective addresses are calculated. With the next clock pulse the instruction code moves on to the decode (DEC) register. The instruction is further decoded and then executed. The time necessary to complete the instruction execution in this step determines the number of clock cycles necessary to execute the instruction.

RAM

Internal RAM (IRAM) is divided into two equal parts of 128-by-16 bits. ARAM is located in the first 128 addresses, BRAM in the last 128. These RAM areas can be operated independently of one another or as a single unit called IRAM. External RAM (ERAM) of 1024-by-16 bits may be accessed from the chip. The ERAM is considered as either an extension of BRAM or IRAM. Address selection is made through the address calculation block, and memory data is passed directly to the A register, B register, or IBUS.

Address Calculation Block

The many modes of memory access permitted by the control commands are supported by the address calculation block. The two independent areas in RAM are accessed by two independent address indexing sections; this architecture can be seen in Figure 1. Register X and its stack XS are used for indexing ARAM only. Indexing calculations are made in the 7-bit adder AD1, and the result is passed to the ARAM pointer DPA. Register Y and its stack YS, both 8-bit registers, are used for indexing BRAM or IRAM. AD2 is used for indexing calculations, and the result is passed to the BRAM-IRAM pointer DPB. The calculation of ERAM addresses follows that of the BRAM addresses except for the final result, which is passed to the ERAM pointer DPE. Two higher-order bits of the ERAM addresses are provided by a page register PGM.

The virtual shift mode is an optional indexing mode which may be specified at any time within a program. In this mode only the four low-order bits of the Y register are used in indexing an address. In the computation of the effective address no carry is made to the fifth bit. This mode provides a 16-bit loop index at a desired location in IRAM or ERAM.

The address calculation block provides the ROM pointer DPR with the address of ROM constant data. The table address register TBA, which can be indexed by the X register, provides the seven low-order bits to DPR. The table page register PGT provides the three most significant bits to the DPR.

Input/Output Interface

Data being input to or output from the MB8764 passes through the chip's input/output interface. The interface allows the selection of three different input modes and two different output modes. An input/output controller operating independently of internal program execution regulates the flow of data. Eight hardware pins, four input and four output, connect the controller with external circuits. Four internal flags also provide an input to the controller. Two of these internal flags, the DMM and ADM, determine the input mode. These flags can be set or cleared by program control. The three input modes are:

1) The program read mode, or P mode, DMM=0, ADM=0,
2) The non-address-attached direct memory access (DMA) read mode, or D mode, DMM=1, ADM=0,
3) The address-attached DMA read mode, A mode, DMM=1, ADM=1.

The program read mode allows data to be read from an external circuit to the El register. There the data may be manipulated by the DSP as needed. The non-direct-attached DMA read mode performs the same function as the program read mode but in addition automatically transfers the data to the internal RAM address indicated by the DMC register. In the address-attached DMA read mode an address is transferred to the EIA register along with the data going to the EI register. The address passed is the address used for storing the data in IRAM.

In all input modes three pins, the AIF, RCK, and ACT pins, control the transfer of information into the MB8764. The AIF pin signals to the controller that the external device is ready to pass information to the MB8764. A zero level on the ACT pin signals that the MB8764 is ready to accept information. The RCK pin provides the write clock for the information transfer.

The two output modes are selected by the value that is entered into the fifth most significant bit of the EA register. If the bit equals zero, the E mode is selected; if one, the I mode is selected.

The E mode uses an external signal to clock the signal into the external circuit. The I mode provides a clock signal from the WCK pin to the external circuit. The output process is begun with a request to output from the REQ pin. The external circuit provides its response to the request to send data to the BCT pin. Information transfer is clocked as discussed earlier. Address information and/or data can be passed to the external circuit. In the I mode, the AOF pin controls the type of data sent. In the E mode, the ASL pin is used to provide the same function.

Summary

The Fujitsu MB8764 performs basic arithmetic functions, with the exception of division, at a very high rate. Its speed in processing arithmetic functions is due to:

1) An instruction cycle of 0.1 μsec (with 20 MHz clock),
2) A parallel pipeline structure with a multiplier circuit separate from the ALU,
3) An ability to execute compound statements.

Claims to a 0.1 μsec multiplication operation may be misunderstood. Actual time from input of the multiplicands into the A and B registers to the result being placed into the D register is 0.2 μsec. But, due to the pipeline structure, multiplication operations can be carried out one directly after another giving rise to the 0.1 μsec multiply claim. The ALU provides a 26-bit result into the D register but only 16 bits of this result are easily accessed; thus, under normal operations, the internal accuracy of the chip is limited to 16 bits.

External expansion capabilities of the ROM allow the user to develop his own working device without having Fujitsu create an internal mask ROM. A limitation when expanding ROM externally is that the chip is unable to access from internal and external ROM in the same program.

Data transfer within RAM, although adequate, could be made more flexible by allowing MOV instructions to specifically address ARAM and BRAM rather than IRAM as a whole.

Input/output operations allow a variety of modes to the user and require just a few lines of code to implement, thus they do not slow down program execution appreciably.

DESIGNING A DIGITAL FILTER ON THE MB8764

Introduction

Digital filters provide advantages over analog filters in some applications. They provide the designer with a more reliable and more flexible filter, that is reproducible to exact specifications. Two characteristics of digital devices limit the implementation of digital filters, finite processing speed and finite word length. A digital device must operate on discrete data at a finite rate of speed. For adequate performance input data is limited to frequencies of less than one-fifth the sample rate of the device. Finite word length limits the poles and zeroes of the filter to a finite number of points. This becomes critical in cases of high sample frequency to maximum signal frequency ratios.

Just as analog filter designers must consider the arrangement of discrete components, digital filter designers must consider the digital filter structure. The structure of a digital filter affects its speed of operation, its sensitivity to finite word length, and its ease of implementation. A rule of thumb that should be applied to all IIR (Infinite Inpulse Response) digital filter structures is to implement the filter in sections no greater than second-order. This reduces the sensitivity the device has to errors in the filter coefficients. A cascade or parallel combination of these second order sections is most often used by designers.

Numerous structures are available to implement second order sections. The direct structures are most frequently used because of their simplicity and speed. This chapter will show the capabilities of the MB8764 to implement digital filters designed as cascades or parallels of direct structured biquadratic sections. The advantages and disadvantages of the various designs as implemented on the MB8764 will be discussed.

Implementing a Biquadratic

Four direct structures will be analyzed and judged on their ability to implement a biquadratic section on the MB8764. The four structures are judged by the following points:

1) Time delay between input and output,
2) Length of program, and
3) Memory space required.

Clock rate for the MB8764 is assumed to be at its maximum, thus one instruction cycle equals 0.1 μsec. Each structure's model, MB8764 memory map, and computation loop program are shown in figures 6,7,8, and 9.

The 1D Structure

The 1D direct structure computes the output y(k) in terms of an effective input m(k). Two equations define its operations:

$$m(k) = x(k) - b_1 m(k-1) - b_2 m(k-2)$$

$$y(k) = a_0 m(k) + a_1 m(k-1) + a_2 m(k-2)$$

The program for a 1D structure requires 18 machine cycles or 1.8 μsec to complete one loop. Input to output delay equals 1.0 μsec. All locations in ARAM are used, with three occupied by active data. Seven locations in BRAM are used (see Figure 6).

2D Structure

The 2D structure first accepts the input, then computes output using results from the previous cycle. The governing equations are:

$$y(k) = a_0x(k) + p_1(k-1)$$

$$p_1(k) = a_1x(k) - b_1y(k) + p_2(k-1)$$

$$p_2(k) = a_2x(k) - b_2y(k)$$

The program requires 17 machine cycles or 1.9 μsec for computations between inputs. Output occurs 0.8 μsec after input. Two locations in ARAM and seven locations in BRAM are used. ARAM is not cycled (see Figure 7).

3D Structure

In the 3D structure all possible calculations are performed before the input is received. The governing equation is:

$$y(k) = a_0x(k) + a_1x(k-1) + a_2x(k-2) - b_1y(k-1) - b_2y(k-2)$$

The computation loop requires 1.6 μ sec. The delay between input and output is 0.8 μsec. Six locations in ARAM are active and cycled through the whole ARAM. Six locations in BRAM are used (see Figure 8).

4D Structure

The 4D structure is the transpose of the 3D structure. The governing equations are:

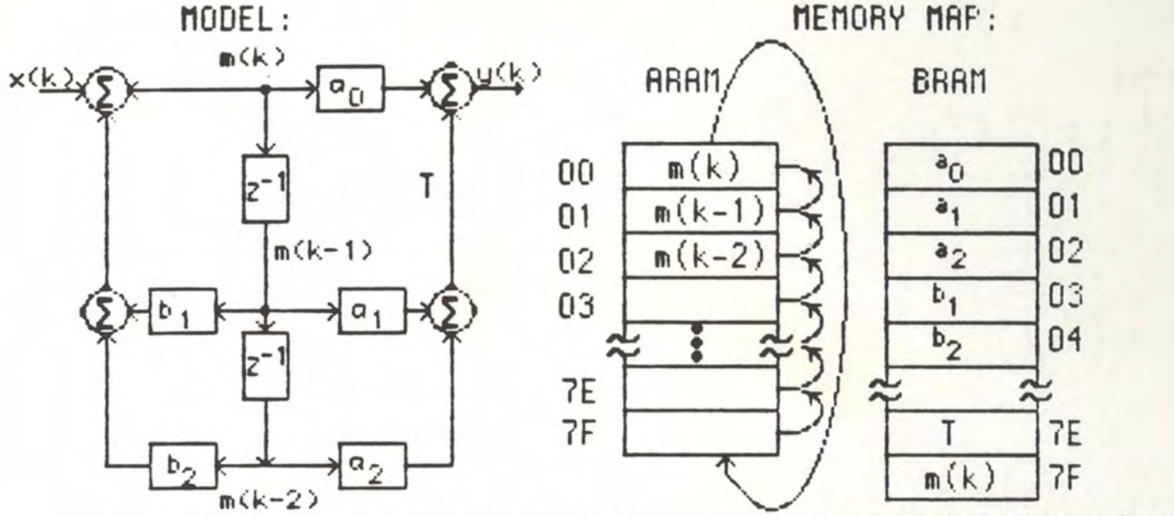
$$r_0(k) = x(k) + r_1(k-1) \qquad q_1(k) = a_1 r_0(k) + a_2 r_0(k-1)$$

$$y(k) = a_0 r_0(k) + q_1(k-1) \qquad r_1(k) = -b_1 r_0(k) - b_2 r_0(k-1)$$

This is the slowest of the four structures, requiring 2.0 μsec for the program loop, and 1.1 μsec from input to output. Six locations are rotated through ARAM, five locations are used in BRAM.

Structure Comparison Results

The 3D structure offers the fastest processing time of the four structures and shares the shortest input to output delay with the 2D structure. The 2D structure uses the least memory locations and would be the best choice in applications where the designer does not want to cycle through ARAM. In each of the programs four instructions are required for input/output and loop control. These four instructions require, as a minimum, six instruction cycles to be processed. A loop has been built into the input data instructions which causes the program to wait until new input data is received. The loop allows the speed at which data is input to control the program sample rate, thus there is no need to control sample rate by inserting lines of code. In the case where program length corresponds to the input data rate, the loop may be removed, allowing for a 0.2 μsec faster program loop. Removing the loop will allow the same input to be acted on more than once if timing of the



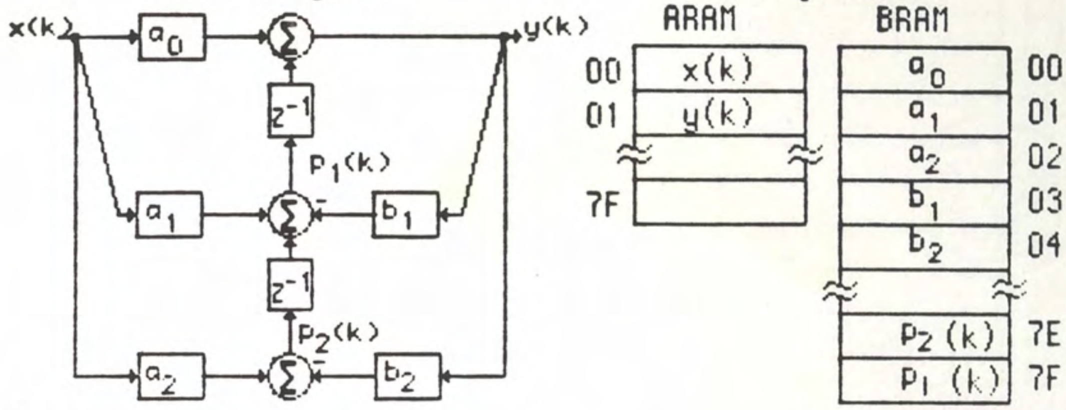
NOTE: Arrows depict movement of variable designation caused by indexing.

Program:

```
        Code                         Comments
L1   LAB:NOP $01(Y),$01(X)       *a1 into A; m(k-1) into B
     LAB:NOP $02(Y),$02(X)       *a2 into A; m(k-2) into B
     LAB:MLT $03(Y),$01(X)       *a1 x m(k-1);-b1 to A;m(k-1) to B
     LAB:MSM $04(Y),$02(X)       *(a2 x m(k-2))+(a1 x m(k-1));etc
     MOV:MLT D,$FF               *store result of last instr; etc
     NOP:MRD                     *(-b1 x m(k-1))-(b2 x m(k-2))
L2   JOC:NOP L2,IF               *loop here until input received
     MOV     *$800,EA            *set output mode and sequence
     MOV;NOP EI,A                *input to A
     MOV:SUM $00(Y),A            *m(k) found; a0 to A
     MOV:NOP D,B,$FE             *m(k) to B and to BRAM
     MBA:NOP $00(X),$FE          *m(k) to ARAM
     MOV:NOP $FF,D             *(a2 x m(k-2))+(a1 x m(k-1)) to D
     MXY:MSM *$7F,*$00           *x=x-1;y=y;y(k) calculated
     MOV:NOP D,EO                *ouput y(k)
     JMP:NOP L1                  *start loop again at L1
```

Figure 6. 1D Filter Structure Model and Program.

Block diagram: Memory map:

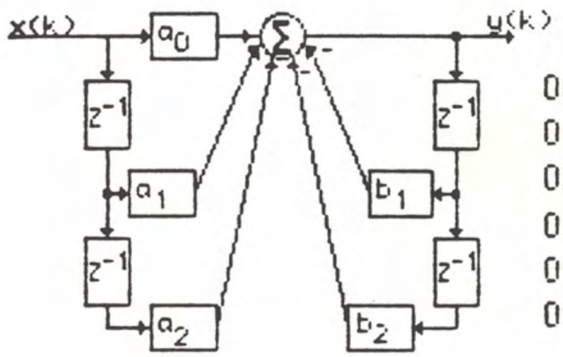


Program:

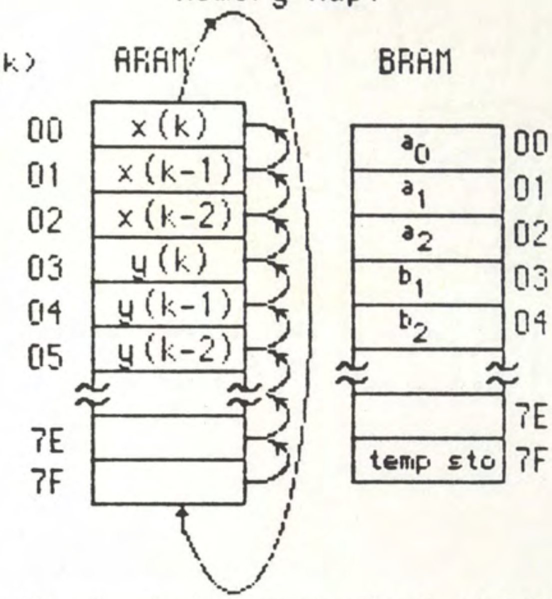
```
      Code                      Comments
L1    JOC:NOP L1,IF             *waits for input
      MOV     *$800,EA          *sets output mode and sequence
      MOV:NOP EI,$00            *x(k) to ARAM
      LAB:NOP $00,$00           *x(k) to A, a0 to B
      MOV:NOP $FF,D             *p1(k-1) to D
      LAB:MSM $00,$01           *calc y(k);x(k) to A;a1 to B
      MOV:NOP $D,EO,$01         *y(k) to output and ARAM
      LAB:MLT $01,03            *y(k) to A; b1(k) to B;x(k) x a1
      NOP                       *
      MOV:MRD $FE,A             *p2(k-1) to A
      LAB:SUM $00,$02           *calc p1(k);x(k) to A;a2 to B
      MOV:NOP D,$FF             *p1(k) to BRAM
      LAB:MLT $01,$04           *y(k) to A;b2 to B;x(k) x a2
      NOP                       *
      NOP:MRD                   *calc p2(k)
      MOV:NOP D,$FE             * p2(k) to BRAM
      JMP:NOP L1                *start loop again at L1
```

Figure 7. 2D Structure Model and Program.

Block Diagram:



Memory Map:

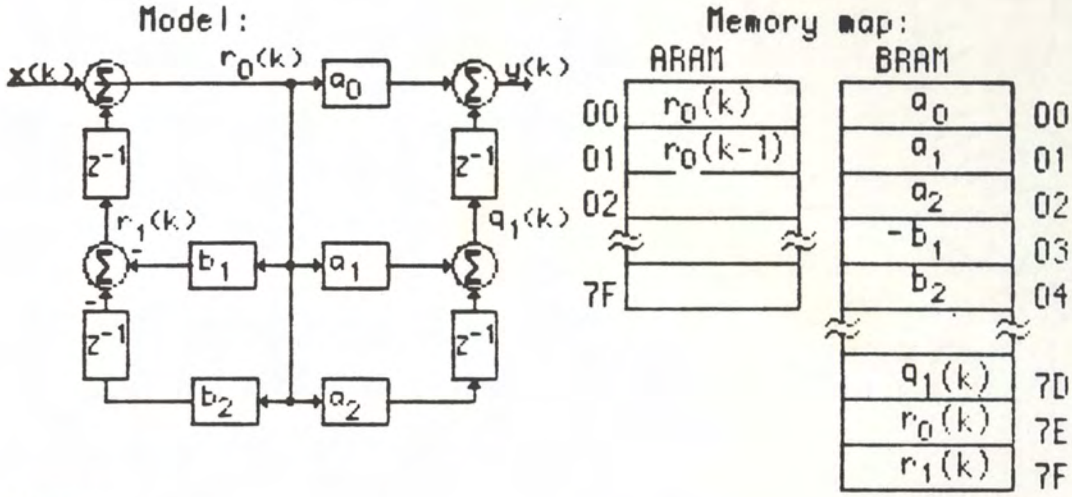


NOTE: Arrows depict movement of variable designation caused by indexing.

Program:

```
        Code                        Comments
L1      LAB:NOP $01(X),$01         *x(k-1) to A;a1 to B
        LAB:NOP $02(X),$02         *x(k-2) to A;a2 to B
        LAB:MLT $04(X),$03         *y(k-1) to A;b1 to B;x(k-1) x a1
        LAB:MSM $05(X),$04         *y(k-2) to A;b2 to B;cont calc of y(k)
        LDI:MRD #$0000             *y(k) calc made indep of # of L2 loops
L2      JIF:MRD L2                 *waits for input;cont calc of y(k)
        MOV     #$600,EA           *sets output mode and sequence
        MOV:NOP EI, B:$FF          *x(k) to B and BRAM
        MOV:NOP $80,A              *a0 to A
        MXY:NOP #$7F,#$00          *shift x index back 1
        MBA:MSM $01(X),$FF         *x(k) from BRAM to ARAM;calc y(k)
        MOV:NOP D,EO:$FF           *y(k) to output and BRAM
        MBA:NOP $04(X),$FF         *y(k) from BRAM to ARAM
        JMP L1                     *start loop again at L1
```

Figure 8. 3D Structure and Program.



Program:

```
        Code                         Comments
L1      JOC:NOP L1,IF                *wait for input
        MOV:    *$600,EA             *sets output mode and sequence
        MOV:NOP EI,A                 *x(k) to A
        MOV:NOP $FF,B                *r1(k-1) to B
        NOP:ADD                      *calc r0(k)
        MOV:NOP D,B,$FE              *r0(k) to B and BRAM
        MOV:NOP $80,A                *a0 to A
        MOV:NOP $FD,D                *q1(k-1) to D
        MBA:MSM $00(X),$FE           *calc y(k)
        MOV:NOP D,EO                 *output y(k)
        LAB:MSM $00(X),$03           *r0(k) to A;-b1 to B
        LAB:NOP $01(X),$04           *r0(k-1) to A; b2 to B
        LAB:MLT $00(X),$01           *r0(k) to A;a1 to B;-b1 x r0(k)
        LAB:MAD $01(X),$02           *r0(k-1) to A; a2 to B:calc r1(k)
        MOV:MLT D,$FF                *r1(k) to BRAM;a1 x r0(k)
        MXY:MSM *$FF,*$00            *calc q1(k);shift x index back 1
        MOV:NOP D,$FD                *q1(k) to BRAM
        JMP:NOP L1                   *start loop again at L1
```

Figure 9. 4D Structure Model and Program.

external input is slower than the program loop execution rate. The use of an input loop cannot prevent the loss of some data samples in cases where the device sample rate is slower than the input data send rate.

Programming a Multiple Biquadratic Section Filter

The 3D structure was found to perform best on the MB8764. A comparison will now be made between the N^{th} order filter programmed as a cascade and a filter programmed as a parallel combination of 3D biquadratic sections. The comparison will determine which is most suitable for the MB8764, considering advantages and disadvantages to each approach. Each program will be judged on the following aspects:

1) Time delay between input and output,
2) Length of program, and
3) Memory space required.

Of course there are countless ways to program the MB8764 to have it accomplish the calculations necessary. The programs shown are written for maximum speed in the computation loop. As in the 3D biquadratic section programmed previously, variables and initial conditions are assumed to be stored in the IRAM when the loop begins. Initialization is accomplished outside the loop and thus not shown. It is necessary to set initial conditions even if they are zero, because the MB8764 does not set all registers to zero when powered up. There is no instruction which clears all the memory locations; therefore they must be accessed one location at a time.

Cascade Realization

The cascade model offers a method of splitting a large-order filter into small sections, thus reducing the filter's sensitivity to coefficient quantization. Figure 10 shows the model for an N^{th} order cascade of 3D structure biquadratic sections. To achieve the best results with the cascade model on a fixed-point processor such as the MB8764 the designer must:

1) Balance the DC gain of the sections (this may be accomplished by proper pole zero pairing),
2) Scale each section individually to prevent overflow within the section, and
3) Arrange the sections in the order which minimizes the output noise.

From a designer's viewpoint cascade realization can be difficult to implement, because pole zero pairing and section ordering are intricate steps.

Figure 11 shows the program and Figure 12 shows the memory map used in implementing a cascade of 3D sections. The number of lines of code necessary to input initial code conditions and variables in the worst case is 16n+6, where n equals the number of biquadratic sections in the filter. Worst case implies that no variables or initial conditions share the same values. An additional two or three lines are needed for initialization. The computation loop for the program requires 10n+11 machine cycles to complete. The delay from time of input to time of output is 4n+5 machine cycles. Data is shifted through the whole ARAM, with 3n+3 locations being occupied with active data at any one time. BRAM data is stationary, and 5n+1 locations are used.

Parallel Realization

The parallel model is easier to design than the cascade model. Each section in the parallel model acts on the input x(k) and provides output to a summing junction (see Figure 13). Therefore, there is no concern about ordering of sections and no reason to pole zero pair each section. The steps necessary to implement the parallel form for a N^{th} order binomial are:

1) Perform a partial fraction expansion of the N^{th} order binomial and group the resulting terms into biquadratic sections,
2) Individually scale each section to prevent overflow within the section.

A program implementing a parallel of 3D biquadratic sections is shown in Figure 14. The program's memory map is shown in Figure 15. Initial conditions and variable input into IRAM require 14n+13 lines of code to enter for the worst case. The computation loop for the program requires 7n+11 machine cycles to complete. The delay from input instruction to output instruction, independent of the filters order n, is equal to seven machine cycles. The ARAM is used just as in the cascade program. BRAM data is stationary and uses 4n+5 locations.

Comparing the Cascade and Parallel Programs

In ease of design and in theoretical performance, the parallel model is superior to the cascade model, offering better signal-to-noise ratio (4) and fewer steps in the design process (5). But the section of the type of filter to be used is usually based on the performance of the program that is implementing it. The program implementing a parallel of 3D sections ran faster on the MB8764 for any number of sections. (For a single section, or second-order filter the cascade and parallel design are identical.) The performance of the parallel example is due to the fact that it has n-1 less multiplications to perform than the cascade and that it can perform its summations very efficiently. The fact that all calculations except one multiplication and one addition are performed prior to input of x(k) make its input to output delay very short. Figure 16 is a graph of the implementation time of an N^{th} order parallel and cascade of 3D sections.

Both programs are implemented with a minimum of loops. As a consequence much longer programs will be stored in the instruction ROM. The advantage for minimizing loops is

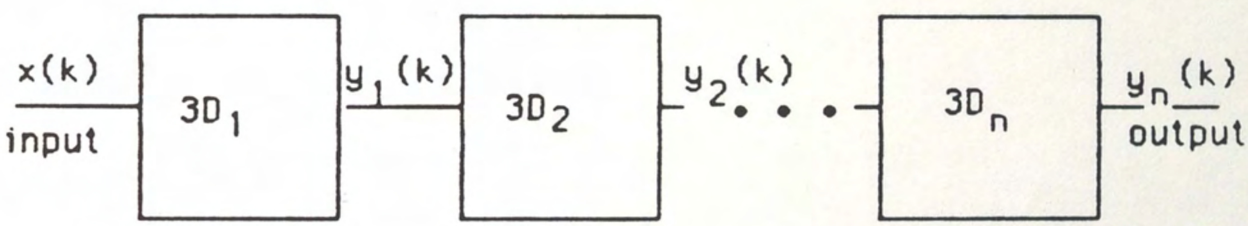


Figure 10. Cascade of 3D Sections.

```
   Code:                                 Comment:
L1 LAB:NOP $01(X),$01                    *-------------|
   LAB:NOP $02(X),$02                    *preliminary  |
   LAB:MLT $04(X),$03                    *calculation  |
   LAB:MSM $05(X),$04                    *of           |
   LAB:MRD $04(X),$06                    *y1(k)        |------------
   LAB:MRD $05(X),$07                    *-------------| begin preliminary
   MOV:MLT D,$FF                         *precalc y1(k)| calculation of y2(k)
   MBA:MSM $03(X),$FF                    *to 03(x)     |______
             .                           *
             .                           *
             .                           *
                                         *This section is repeated from
   LAB:NOP $[3i+1](X),$[5i-2]            *i=2 to n-1, where i represents the
   LAB:NOP $[3i+2](X),$[5i-1]            *biquadratic section being coded, and
   LAB:MRD $[3i+1](X),$[5i+1]            *n is the total number of biquadratic
   LAB:MRD $[3i+2](X),$[5i+2]            *sections.  In this section yi(k)
   MOV:MLT D,$FF                         *preliminary is calculated and the
   MBA:MSM $[3i+3](X),$FF                *calculations for yi+1(k) are begun.
             .                           *
             .                           *
             .                           *________________________________
   LAB:NOP $[3N+1](X),$[5n-2]            *
   LAB:NOP $[3N+2](X),$[5n-1]            *Calculations for preliminary yn(k)
   NOP:MRD                               *completed
   NOP:MRD                               *
   MOV:NOP D,$FF                         *
   MAB:NOP $[3N](X),$FF                  *________________________________
L2 JIF:NOP L2                            *wait for input
   MOV     *$800,EA                      *set output mode and sequence
```

Figure 11. Cascade of 3D Sections Program.

```
MOV:NOP EI,A:$FF             *x(k) to A and BRAM
MOV:NOP $80,B                *begin calculation of y1(k)

LAB:NOP $03(X),$FF           *continue calculations for y1(k)
MOV:MLT $85,B                *begin setup for y2(k) calculations
NOP:SUM                      *
MOV:NOP D,$FF:A              * y1(k) calculation is completed
        .                    *
        .                    *___________________________________
        .                    *
LAB:NOP $[3i](X),$FF         *This section is repeated for
MOV:MLT $[5i],B              *i=2 to n-1. It calculates the output
MBA:SUM $[3(n-1)](X),$FF     *of each section and places the
MOV:NOP D,$FF,A              *output of the previous section into
        .                    *the proper location in ARAM.
        .                    *
        .                    *___________________________________
LAB:NOP $[3n](X),$FF         *final calculations to compute
MOV:MLT $[5n],B              *filter output y(k)
MBA:SUM $[3(n-1)](X),$FF     *
MOV:NOP D,$FF:EO             *output y(k)
MBA:NOP $[3n](X),$FF         *
MXY:NOP $FF,$00              *shift X index back 1
JMP:NOP L1                   *jump back to beginning
```

Figure 11. Continued.

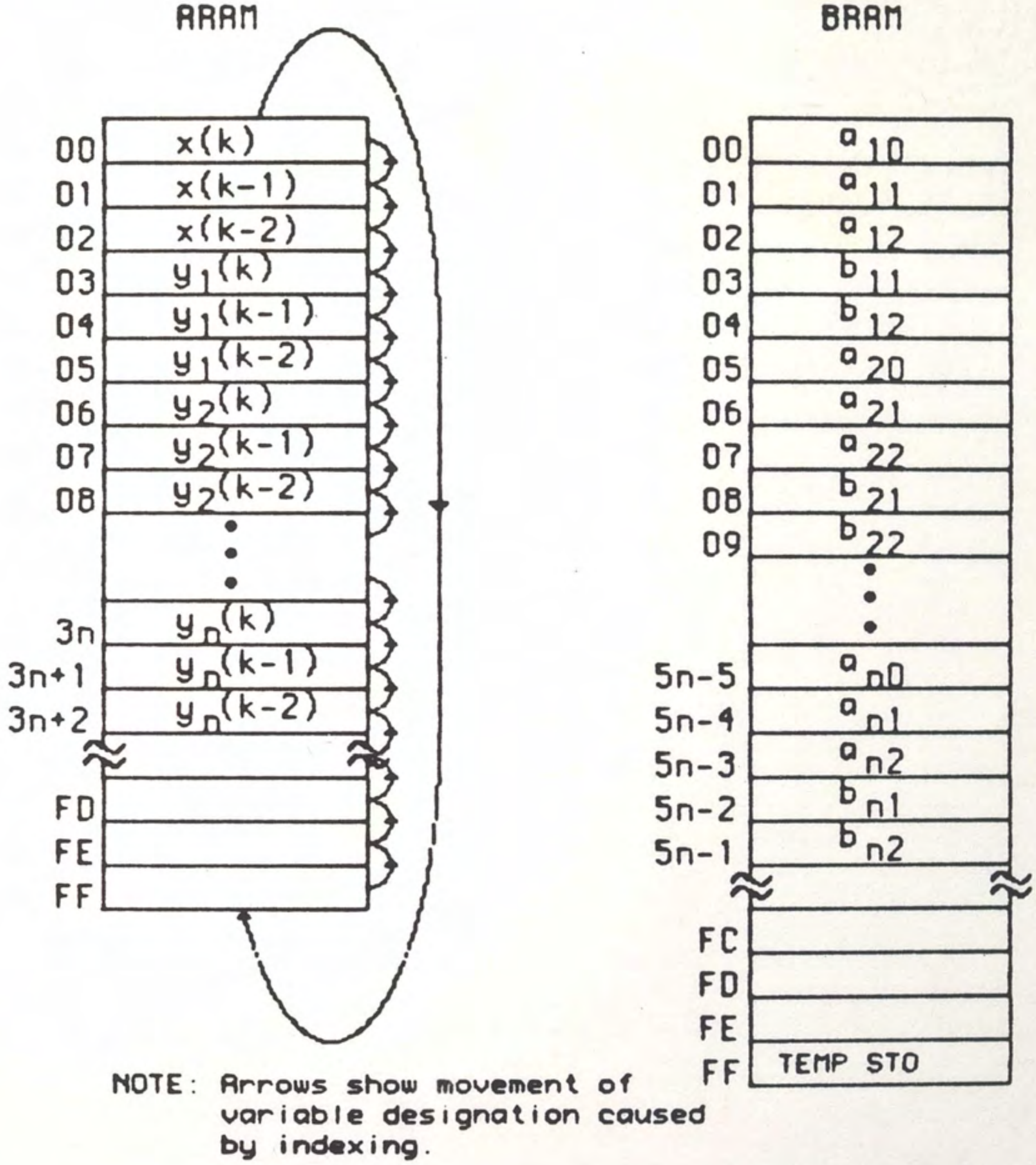
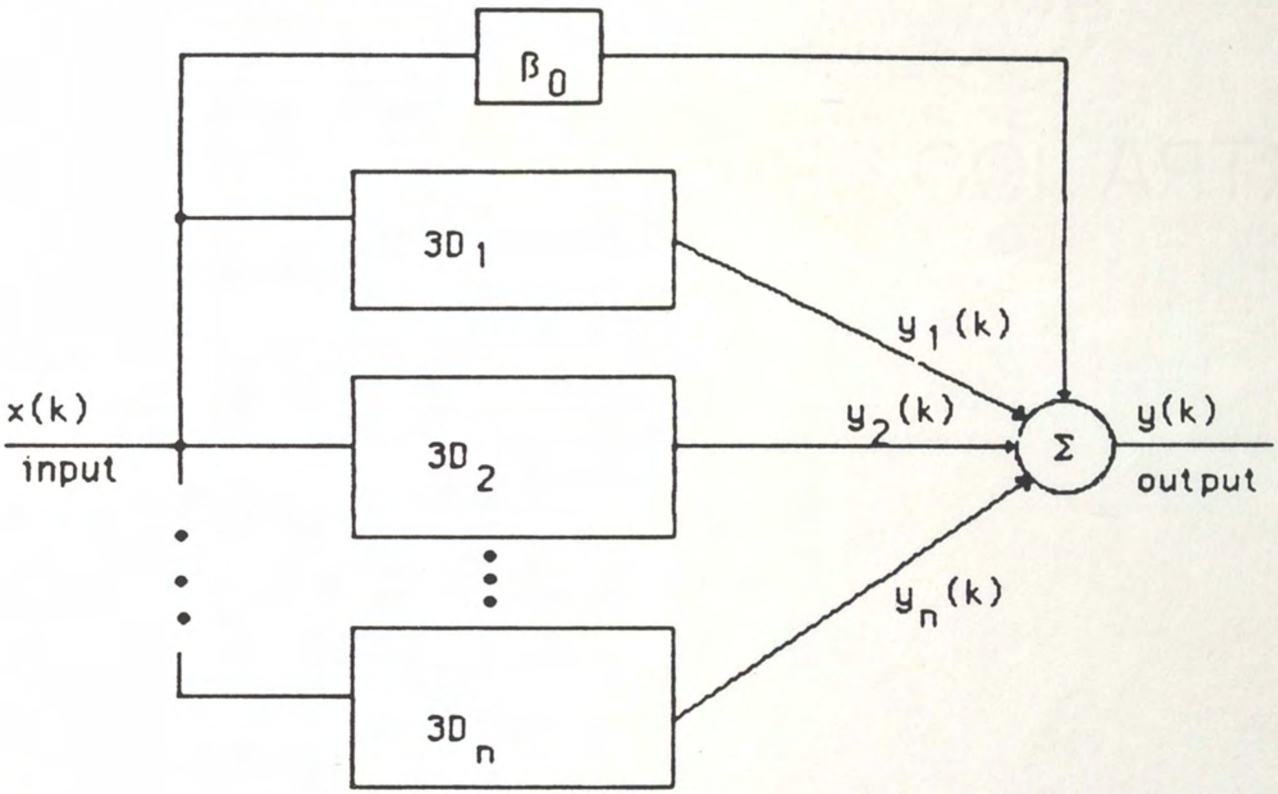


Figure 12. N^{th} Order Cascade Filter Memory Map.



NOTE: Only delayed values of x(k) are used in the 3D sections to calculate y(k)

Figure 13. Parallel of 3D Sections.

```
        Code:                               Comments:
L1      LAB:NOP $01(X),$01             *-------------|
        LAB:NOP $02(X),$02             *             |
        LAB:MLT $04(X),$03             *calculation  |
        LAB:MSM $05(X),$04             *of           |
        LAB:MRD $01(X),$05             *y1(k)        |------------
        LAB:MRD $02(X),$06             *-------------| begin
        MOV:MLT D,$FF                  * y1(k)       | calculation of y2(k)
        MBA:MSM $03(X),$FF             *to 03(x)     |
                .                      *
                .                      *
                .                      *
                                       *This section is repeated from
        LAB:NOP $[3i+1](X),$[4i-1]     *i=2 to n-1, where i represents the
        LAB:NOP $[3i+2](X),$[4i]       *biquadratic section being coded, and
        LAB:MRD $01(X),$[4i+1]         *n is the total number of biquadratic
        LAB:MRD $02(X),$[4i+2]         *sections.  In this section yi(k)
        MOV:MLT D,$FF                  *is calculated and the calculations
        MBA:MSM $[3i](X),$FF           *for yi+1(k) are begun.
                .                      *
                .                      *
                .                      *
        LAB:NOP $[3N+1](X),$[4n-1]     *
        LAB:NOP $[3N+2](X),$[4n]       *Calculations for yn(k)
        NOP:MRD                        *completed
        NOP:MRD                        *
        MOV:NOP D,$FF                  *
```

Figure 14. Parallel of 3D Sections Program.

```
      MAB:NOP $[3n](X),$FF       *________________________________
      LAB:NOP $03(X),$00         *
      LAB:SUM $06(X),$00         *   sum          Only the A register
              .                  *   all          information is used
              .                  *   y(k)'s       in the summation.
              .                  *
                                 *
      LAB:SUM $3[n-1](X),$00     *
      NOP:SUM                    *last y(k) summed________________
L2    JIF:NOP L2                 *wait for input
      MOV     #$800,EA           *set output mode and sequence
      MOV:NOP EI,A:$FF           *x(k) to A and BRAM
      MBA:NOP $00(X),$FF         *x(k) to ARAM
      MXY:MSM #$7F,#$00          *calc Σy(k)'s + x(k) x β0
      MOV:NOP D,EO,$FF           *output y(k)
      JMP:NOP L1                 *jump back to beginning
```

Figure 14. Continued.

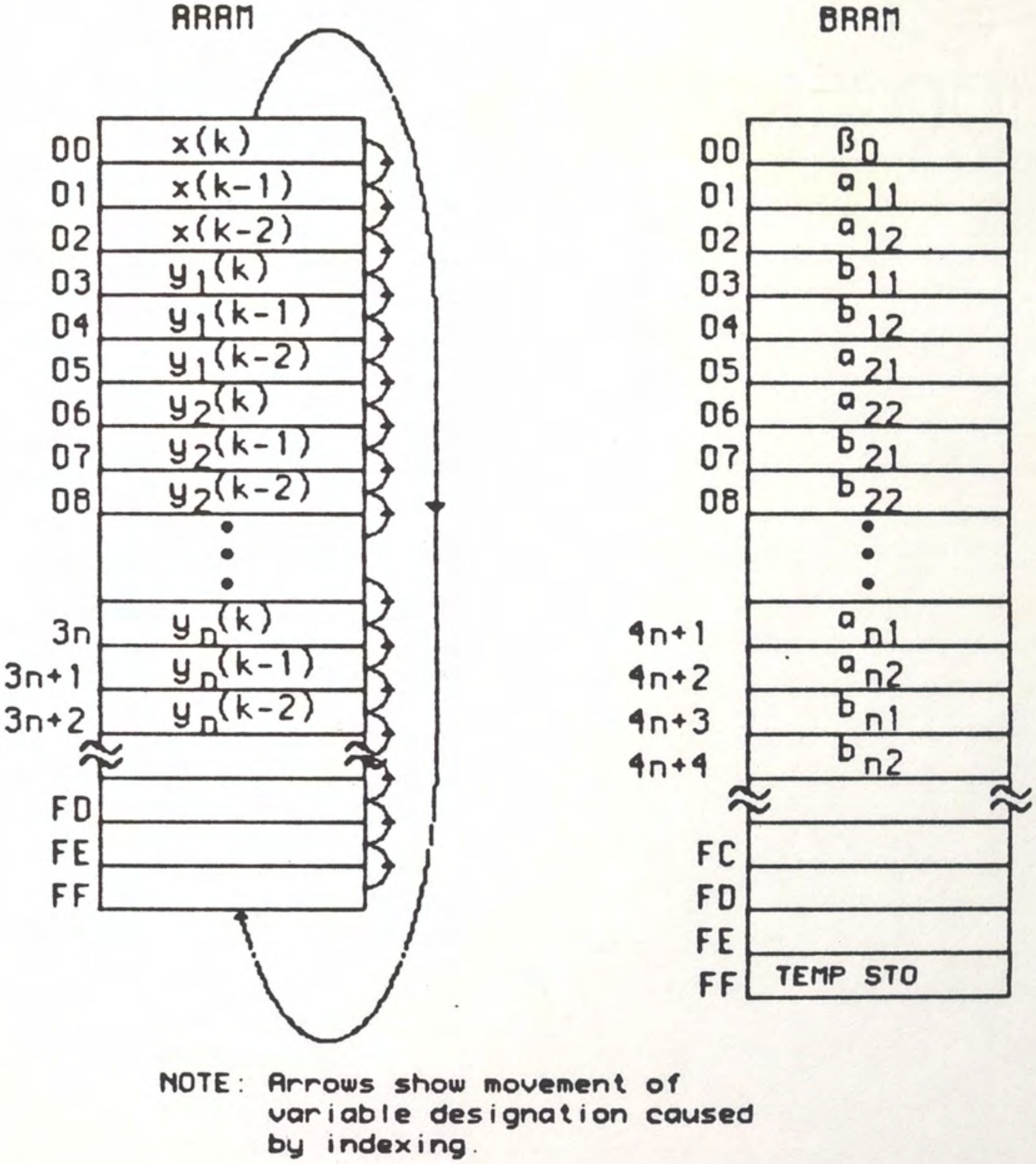


Figure 15. N^{th} Order Parallel Filter Memory Map.

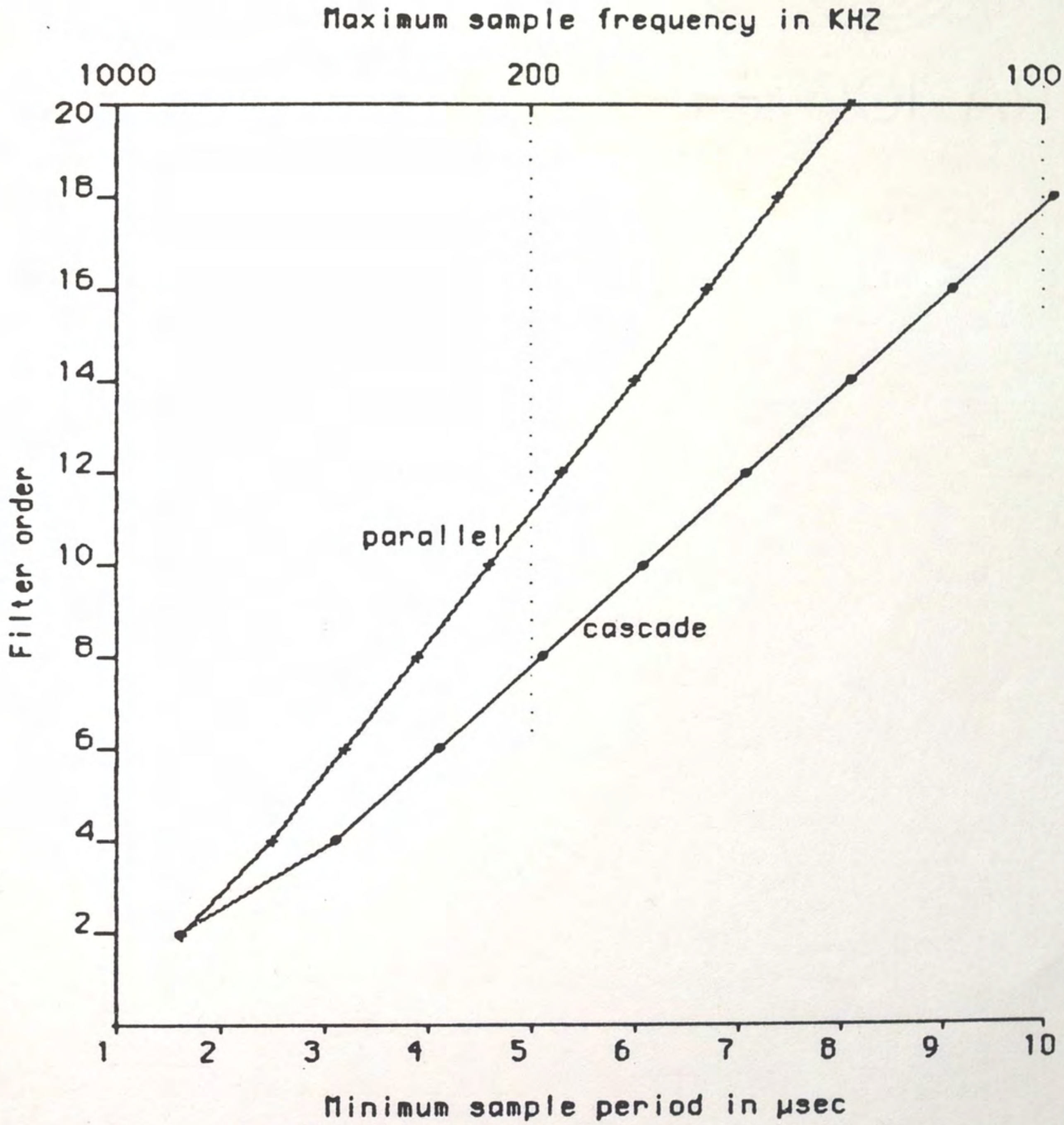


Figure 16. N^{th} Order Filter Implementation Time Graph for Parallel and Cascade Filters on the MB8764.

speed in program execution. For example an N^{th} order parallel filter with loops requires 6(n-1) additional machine cycles to execute and saves 7n-29 locations in the instruction ROM (n=number of 3D sections).

MB8764 Capabilities in Implementing Multiple Filter Programs

Designing a program to implement more than one filter with multiple inputs and outputs is easily accomplished on the MB8764. Due to limited amount of memory available, restrictions are placed on the number of and complexity of filters to be programmed together. Restrictions are also placed on the sample rate of the filters which must be integer multiples of one another. Table 1 shows the capability of the MB8764 to implement multiple filter programs of 3D sections placed in parallel. The MB8764 has only one input/output port which must be time-shared in a multiple filter program. To accomplish this time-sharing, the inputs must be synchronized to occur in a specific order.

If all filters are of the same frequency, then programming multiple filters in the one program is accomplished in three steps.

1) Arrange the calculation loops for the programs you wish to implement into a single list:
2) Remove the jump statement from the bottom of each program except the last and have that jump

statement return to the top of the first program. This makes the list of calculation loops into a single loop;

3) Change the addressing within each program to point to the section of ARAM and BRAM in which initial conditions and variables are located.

Setting initial conditions and variables into IRAM is accomplished for all filters before the calculation loop is begun. Filters with sample rates that are integer multiples of one another are implemented as in the steps listed above with the addition of a step to install counters and jump instructions to control program flow.

Chapter Summary

In this chapter the 3D biquadratic structure was found to have the fastest calculation loop of the four direct structures. The parallel implementation and cascade implementation of an N^{th} filter of 3D structures were compared. The parallel structure was shown to be superior in performance (see Table 2). From these programming examples it can be seen that the MB8764 performs mathematical functions very efficiently but this efficiency is reduced considerably when results must be moved out of the D register to ARAM or when looping is used. For best performance in speed, programs written for the MB8764 should use a minimum of transfer instructions and should avoid looping.

TABLE 1
CAPABILITY OF THE MB8764 IN PERFORMMING
MULTIPLE FILTER PROGRAMS

filter order	max # of filters	max sample frequency	approx memory use		
			IROM	ARAM	BRAM
2	16	39.06 KHZ	736	96	128
4	10	40.00 KHZ	670	90	120
6	8	39.06 KHZ	704	96	128
8	6	42.74 KHZ	654	90	120
10	5	43.48 KHZ	650	90	120
12	4	47.17 KHZ	604	84	112
14	4	41.67 KHZ	688	96	128

TABLE 2

COMPARISON BETWEEN A PARALLEL AND CASCADE FILTER
IMPLEMENTATION OF 3D BIQUADRATIC
SECTIONS ON THE MB8764

Feature	Parallel	Cascade
min sample period	(11+7n).1μs	(11+10n).1μs
input to output delay	.7μs	(4n+5)>1μs
IROM locations used	25+21n	17+26n
ARAM locations used	3n+3	3n+3
BRAM locations used	4n+4	5n
note: for .1μs machine cycle, n=# of 3D sections		

Scaling is necessary in the design of a digital filter to prevent overflow within fixed-point machines such as the MB8764. The design of the MB8764 also helps to prevent overflow during intermediate calculations in the arithmetic and logic block. Internal ALU operations and the D register provide twice the dynamic range of the ALU input registers A and B. Thus the result of an intermediate operation which overflows in a 16-bit register of the MB8764 can remain valid in the D register, allowing subsequent operations without overflow. If an overflow should occur, the MB8764 can minimize the error through the use of the CLP flag.

THE MB8764 DEVELOPMENT SYSTEM

Introduction

Once a program has been designed for a digital device it is important that it be fully tested. This especially true for the MB8764 program that is to be input into the internal mask ROM of the chip, as there is no adjustment possible once the mask is produced. Any mistakes in the mask ROM design must be accepted or the design must be corrected and a new chip produced. Fujitsu MB8764 programs can be tested on the MB8764 itself with the use of the MB87902 software development tool kit. The tool kit supplies a 16 MHz clock to the MB8764 giving it a machine cycle of 0.125 μsec or 25% slower than the minimum specified MB8764 machine cycle of 0.1 μsec. The slower clock rate is required for the MB8764 to make data transfer between external RAM and the chip.

This chapter first gives a brief description of the development system for the MB8764 and then follows through the testing of a fourth-order Butterworth filter program. The information on the MB8764 development system found in this chapter is derived from references (6) and (7) and from experiences the author had when using the development system.

Description of the MB8764 Development System

The development system for the MB8764 can be divided into two primary parts, a Fujitsu FM-16S microprocessor and the Fujitsu FDSP KIT-8764 evaluation board. The microprocessor is a standard Fujitsu model equipped with the following hardware:

1) 10 mega-bit internal drive,
2) One 5¼" floppy disk,
3) A CP/M86 board and expansion RAM card,
4) CRT and printer.

Software provided includes: 1) Wordstar, a word processing program used to create code and data files; 2) the MB8764 assembler (ASM64) which assembles the wordstar code files into the ROM executable code; and 3) the MON64 program which is actually two programs used to control the FDSP KIT-8764 evaluation board.

The FDSP KIT-8764 evaluation board is primarily a standard MB8764 with support hardware to interface it with the Fujitsu FM-16S microprocessor. It also provides the designer with three sockets for EPROM programming and testing. The support hardware includes:

1) A 1024-word instruction RAM, accessed by the MB8764 through the MB8764's external instruction port,
2) A 1024-word expansion RAM, which operates as a standard MB8764 expansion RAM,

3) Two 512-word data RAMs, one for storing data to be input into the MB8764 and the other for storing the MB8764 output data,

4) An analog interface, which provides 12-bit ADC and DAC for analog input/output, and

5) An interface circuit, to enable the FM-16S microprocessor to control the board.

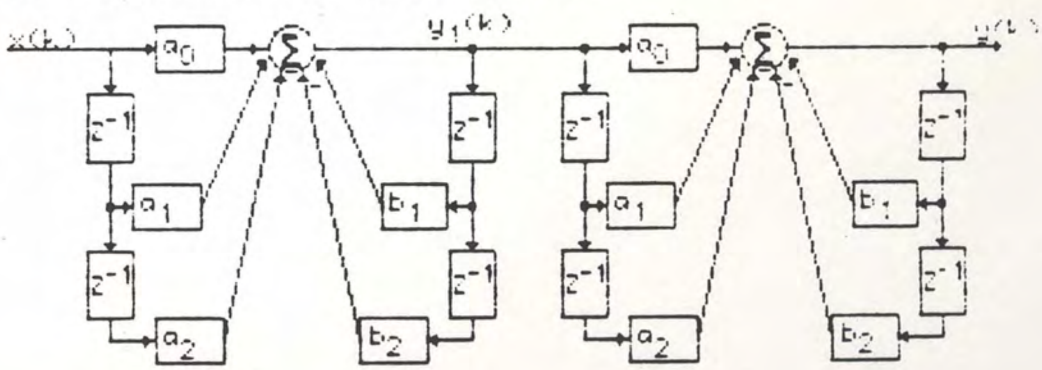
With the development system, a designer may choose the MB8764 input to be an analog signal a digital signal from data RAM, or a digital signal from a user supplied device. The same choices apply to the MB8764 output. If the output is directed to data RAM then 512 words of output data may be accessed and viewed on the CRT. Program execution can be stopped by the microprocessor at almost any point in the program. While paused the D, A, X, Y and CO registers can be viewed as well as any addresses in the instruction RAM, internal RAM, or external RAM. Any of the addresses or registers that can be viewed may also be changed to another value. If instruction code is altered, the new program can be loaded back from the instruction RAM to a disk file in the microprocessor. When a program passes all tests, an EPROM is made or a floppy disk created with the tested program on it. Fujitsu will use this EPROM or floppy disk to create a custom MB8764 chip with an IROM loaded with the program sent. If a mask IROM is not required EPROMs can be manufactured by the development system and used as external IROM for the MB8764.

Testing a Program

A fourth-order low pass Butterworth digital filter was designed with the following specifications:

1) Cutoff frequency - 50 KHZ
2) Max loss in passband - 3 DB
3) Sample frequency - 250 KHZ

Conversion from analog to digital was made via the bilinear transform. The filter was implemented as a cascade of two biquadratic sections. The figure below shows the model and the equation is represents.



$$H(z) = \frac{.1839+(.3678)z^{-1}+(.1839)z^{-2}}{1-(.3290)z^{-1}+(.0646)z^{-2}} \times \frac{.2532+(.5065)z^{-1}+(.2532)z^{-2}}{1-(.4532)z^{-1}+(.4663)z^{-2}}$$

Figure 17. Butterworth Filter Model and Equation.

The step response and the frequency response of the model was calculated on an HP 85 computer, and it verified the model to be valid. The calculated step response data was saved to compare to the output data from the real time execution of the model on the MB8764.

As assembly level program was written for the filter and edited in wordstar on the Fujitsu FM-16S microcomputer. The file created by wordstar was checked for errors and assembled when all errors had been corrected. Error codes from the assembler were adequate but documentation of program format requirements were inadequate with many errors solved by trial and error. The assembler permits some use of address labels and variables in the assembly level program and converts them to proper values before converting the program to machine code. Along with a statement list the assembler provides the designer with a dictionary list and a symbol list. These provide documentation on the variables and labels used in the assembly level program.

Machine code, created upon assembly of the filter program without error, was stored in a .DEB file. The .DEB file was loaded into the instruction RAM of the FDSP KIT-8764 using the DEBG1 program. The DEBG1 program can also be used to read and write programs between EPROM and instruction RAM or from instruction RAM back to the FM-16S microprocessor.

With program instructions loaded into the instruction file program DEBG1 was exited, and the DEBG2 program loaded. The filter program in the instruction RAM was now able to run on the MB8764 under the control of the DEBG2 program.

Because the program called for the step response of the filter, no input was generated within the program. The following functions were accomplished through the use of DEBG2 commands. Output was specified to be placed into the output data RAM. Program execution was begun and then paused to check output data RAM contents, register contents and IRAM contents. Corrections were made to program code until output data results were correct, and the program was operating properly. A special note is made that attempts to store data in address FF of IRAM were not successful, however, when storage was changed to register FC the program ran correctly. A listing of the filter program executed is shown in Figure 18. Figure 19 is a comparison of calculated step response and MB8764 program step response.

Summary

The MB8764 digital signal processing chip is well supported by the MB8764 Support Tool development system. Its ability to run programs at 80% of the maximum internal clock rate of the MB8764 and to use the MB8764 chip instead of a software simulation of the chip gives the designer a chance to evaluate program results in real time. Documentation of assembly language formatting requirements is inadequate. Including formatting examples would greatly improve the documentation.

```
PRG      BWFIL          *required by assembler
ORG      CREECH,$10     *assembler required sets code location
CLR      X:Y:D          *clears X,Y,Z reg

LDI:NOP #$0BC5          *
MOV:NOP A,$80           *
LDI:NOP #$178A          *
MOV:NOP A,$81           *     This section
LDI:NOP #$0BC5          *
MOV:NOP A,$82           *     loads the
LDI:NOP #$150E          *
MOV:NOP A,$83           *     equation coefficients.
LDI:NOP #$0422          *
MOV:NOP A,$84           *
LDI:NOP #$1035          *NOTE:FORMAT REQUIREMENTS ARE STRICT
MOV:NOP A,$85           *A SPACE AFTER A COMMA OR A COLON
LDI:NOP #$206B          *CAUSES ASSEMBLY ERROR.
MOV:NOP A,$86           *
LDI:NOP #$1035          *
MOV:NOP A,$87           *
LDI:NOP #$1D01          *
MOV:NOP A,$88           *
LDI:NOP #$1DD7          *
MOV:NOP A,$89           *
LDI:NOP #$0000          *     This section
MOV:NOP A,$01           *
MOV:NOP A,$02           *     sets initial
MOV:NOP A,$04           *
```

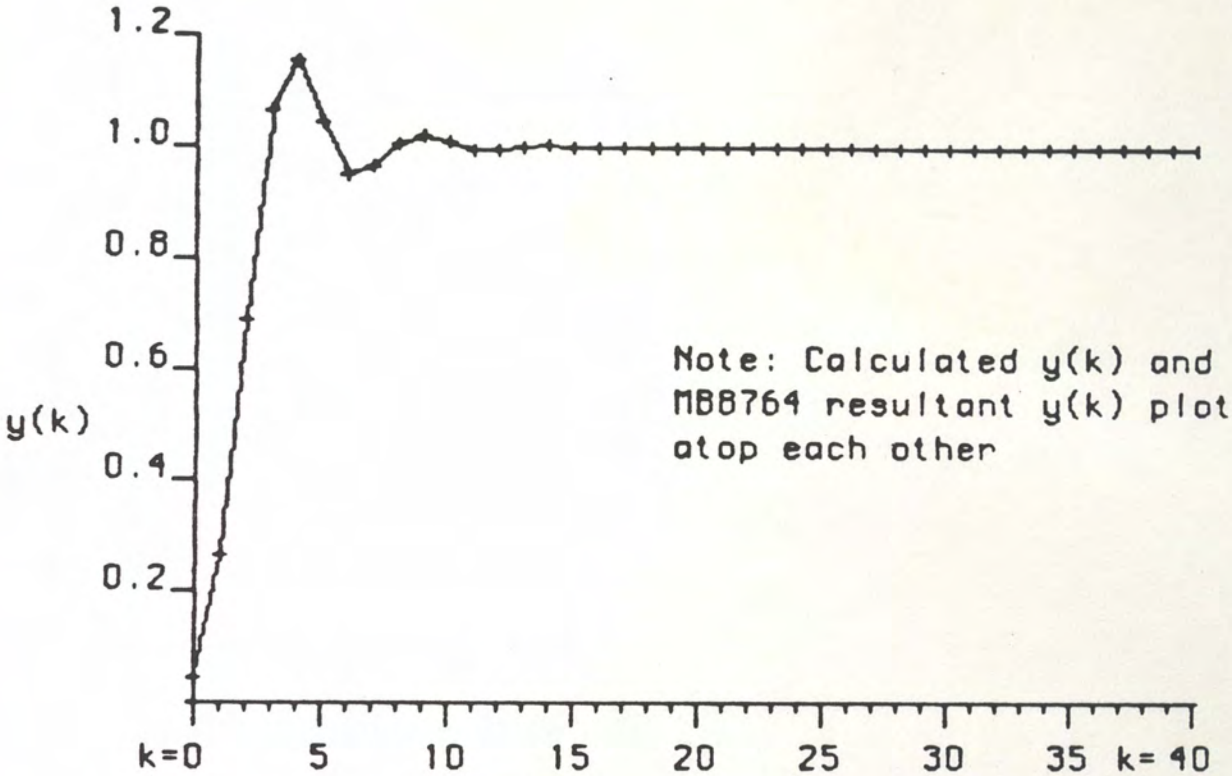
Figure 18. Butterworth Filter Program Ready for Assembly.

```
    MOV:NOP A,$05        *     conditions to zero.
    MOV:NOP A,$07        *
    MOV:NOP A,$08        *______________________________________
L1  LAB:NOP $01(X),$01   *calculation loop begins
    LAB:NOP $02(X),$02   *   calculate          |
    LAB:MLT $04(X),$03   *   preliminary        |
    LAB:MSM $05(X),$04   *   y1(k)              |---------------
    LAB:MSM $04(X),$06   *                      |calculate
    LAB:MRD $05(X),$07   *                      |preliminary
    MOV:MLT D,$FC        *prelim y1(k) to BRAM  |y2(k)
    LAB:MSM $07(X),$08   *----------------------|
    LAB:NOP $08(X),$09   *                      |
    NOP:MSM              *                      |
    NOP:MRD              *                      |
    MOV:NOP D,$FE        *                      |---------------
    MOV     #$800,EA     *set output mode
    LDI:NOP #$4000       *simulates receipt of an input of 1
    MOV:NOP A,$FD        *
    MBA:NOP $00(X),$FD   *
    LAB:NOP $00(X),$00   *x(k) to A and a0 to B
    MOV:NOP $FC,D        *
    NOP:MSM              * x(k) x a0 + (preliminary y1(k))
    MOV:NOP D,$FC        *
    MBA:NOP $03(X),$FC   *
    LAB:NOP $03(X),$05   *
    MOV:NOP $FE,D        *
    MXY:MSM #$7F,#$00    *y2(k) calculated X index shifted back 1
```

Figure 18. Continued.

```
MOV:NOP D,EO:$FE          *y2(k) output
MBA:NOP $07(X),$FE        *y2(k) stored in 06(x) of ARAM
NOP                       *NOTE:THIS PROGRAM HAS EXTRA LINES OF
NOP                       *CODE IN IT TO GIVE IT A SAMPLE RATE
NOP                       *OF 250 KHZ.
NOP                       *
JMP:NOP L1                *returns to start of program loop
END                       *required by assembler
```

Figure 18. Continued.



k	calc y(k)	MB8764 y(k)
0	0.046585	0.046508
1	0.269358	0.269165
2	0.691428	0.691162
3	1.064878	1.064636
4	1.155829	1.155640
5	1.043399	1.043274
6	0.949128	0.949036
7	0.957215	0.957092
8	1.004364	1.004150
9	1.021906	1.020691
10	1.007881	1.007690
11	0.993354	0.993164
12	0.993313	0.993103
13	1.000069	1.002991
14	1.003150	1.001221
15	1.001395	0.998962

|

Figure 19. Impulse Response of Butterworth Filter.

COMPUTING THE DISCRETE FOURIER TRANSFORM ON THE MB8764

Introduction

The discrete Fourier transform (DFT) can be represented by the equation:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad , \; k=0,1,2,\ldots N-1$$

The DFT can be computed directly from the equation above or can be computed using the fast Fourier transform (FFT) algorithm. Implementing a DFT with an FFT algorithm greatly reduces calculations necessary to perform the DFT. This reduction, from approximately N^2 complex multiplication and adds, to $N\log_2 N$ complex multiplications and adds, enables a computer to perform the transform in much less time. The MB8764 which offers a 0.1 μsec multiply and add is a good candidate for performing real time DFTs. This chapter will briefly discuss how the MB8764 can be used to perform the DFT directly and via the FFT algorithm.

Implementing the DFT

A program which performs the primary computation loop of a 64-point DFT of complex inputs is shown in Figure 20. Inputs are assumed to be stored in BRAM. The first loop for k=0 in the DFT equation is a just summation of the complex inputs because the transform coefficients equal one. The remaining loops use complex coefficients which are stored in table ROM. The program can be expanded to perform up to a 512-point DFT but requires input data to be stored in ERAM and additional lines of code to page through the table ROM and RAM. The limit of 512 complex points is set by the ERAM expansion limit of 1024 words. Paging of the ROM is a very complex operation because of the order in which the transform coefficients are accessed in the DFT equation.

Performing the FFT

The FFT algorithm is developed from the DFT by decomposing the DFT of N samples into N/2 DFTs of two samples each. In the process of decomposition, the symmetry and the periodicity of the DFT is taken advantage of in order to reduce the number of calculations necessary to compute the DFT. The required calculations are sometimes referred to as butterfly computations. The equations that must be implemented by each butterfly are:

$$X_{m+1}(p) = X_m(p) + (W^r{}_N)(X_m(q))$$

$$X_{m+1}(p) = X_m(p) - (W^r{}_N)(X_m(q))$$

Where r is determined by the location of the butterfly and $W^r{}_N = e^{-j(2\pi/N)r} = \cos(2\pi/N)r - j\sin(2\pi/N)r$. Given the number of sample points N, values for $\cos(2\pi/N)r$ and $\sin(2\pi/N)r$ r=0 to N/2 can be solved for a stored in ROM as a table for use by the program (see reference 8).

A program to implement the FFT algorithm would consist of the following sections:

1) Data input. Data is input into the MB8764 after being reverse bit shuffled.
2) Calculation. Calculating the results would require calucation of $(N/2)x\log_2 N$ butterflies a routine for the calculation of a butterfly is shown in Figure 21. Twenty-six machine cycles are necessary to execute the butterfly routine. Additional machine cycles are required for loop commands and indexing. The total number of machine cycles for the calculation of a 64- or 128-point FFT is approximately $30\ x(N/2)x\log_2 N$.
3) Data output. The inplace FFT algorithm would provide results to the same registers as the inputs were received. Output in A+jB form would require no additional cycles because it can be performed in the calculation loop. If output is desired in another form additional program steps may be required.

Paging is not necessary if the number of registers in table ROM is less than 128. Thus for more than a 128-point FFT the designer must devise a method to perform the table ROM paging. The 1024 word limit on ERAM expansion allows the MB8764 to compute up to a 512-point FFT. A 64-point FFT can be computed with no need for external expansion. For more than 64 points external expansion is required.

Summary

The MB8764 will perform both the DFT and the FFT algorithm very efficiently for 64 points and requires no external expansion. The DFT is not easily expanded up to the 512 points because it accesses the Table ROM in a complex manner. For the 512-point DFT external expansion of ROM to 2048 words is required. The FFT may also be expanded to 512 points and requires no external ROM, but will require some additional programming steps to provide RAM and ROM paging. With a 0.1 μsec instruction cycle a 64-point DFT can be performed in less than 9.0 msec and a 64-point FFT can be computed in less than 600 μsec.

```
        -----THIS CALCULATES FOR THE SECOND THROUGH N-1 LOOP---
        -----INITIALIZE---- $03=1,set PGT, $00=0,$01=0
        MOV:NOP *$3F,C0           *loop counter for k initialized
L2      MOV:NOP C0,$04            *k=1 to N-1 loop;C0(k) saved
        MOV:NOP *$40,C0           *loop counter for n initialized
L1      LTB:NOP $00(X),$00(Y)     *n=0 to N-1 loop
        MOV:NOP $00,D             * This section
        LTB:MSM $01(X),$00(Y)     * calculates the
        MOV:NOP $00(Y),B          * real and imaginary subtotals
        NOP:MRD                   * and puts
        MOV:MLT D,$00             * real result in address $00
        LTB:NOP $00(X),$01(Y)     * imaginary result in $01
        MOV:NOP $01,A             *
        NOP:MSM                   *
        NOP:SUM                   *
        MOV:NOP D,$01             *---------------------------
        MOV:NOP X,A               *Updates the
        MOV:NOP $03,B             *X and Y index
        MXY:ADD *$00,*$01         *registers for each new n
        MOV:NOP D,X               *---------------------------
        JC0:NOP L1                *jump to L1 63 times then continue
        MOV:NOP $00,E0            *output real part X(k)
        CLR:NOP Y                 *clear Y
        LDI:NOP *$0002            *Compute new
        MOV:ADD $04,C0            *value for
        MOV:NOP D,$03             *X index
        MOV:NOP $03,X             *
        MOV:NOP $01,E0            *output imaginary part X(k)
        LDI:NOP *$0000            *initializing
        MOV:NOP A,$00             *address $00 and
        MOV:NOP A,$01             *address $01
        JC0:NOP L2                *loop back to L2 for 62 times
```

Figure 20. 64-Point DFT Program.

```
L1    MOV:NOP Y,$04              * store Y index
      LTB:NOP $00(X),$00(Y)      *calculate real and imag.
      LTB:NOP $01(X),$01(Y)      *
      LTB:MLT $00(X),$01(Y)      *parts of Xm(q) x WN^r
      LTB:MRD $01(X),$00(Y)      *
      MOV:MLT D,$00:A            *real part to ARAM and A
      MOV:MSM $05,Y              *change y index
      MOV:NOP D,$01              *imag part to ARAM
      MOV:NOP $00(Y),B           *real part Xm(p) to B
      MXY:ADD #$00,#$02          *incrementing $05 Y index
      MOV:SUB D,$7E(Y)           *real part Xm+1(p) to BRAM
      MOV:NOP D,$02              *real part Xm+1(q) to ARAM
      MOV:NOP $01,A              *imag part Xm(q) x WN^r to A
      MOV:NOP $01,B              *imag part Xm(p) to B
      MOV:ADD Y,$05              *
      MOV:SUB D,$7F(Y)           *imag part Xm+1(p) to BRAM
      MOV:NOP D,$03              *imag part Xm+1(q) to ARAM
      MOV:NOP $04,Y              *change back Y index
      MAB:NOP $02,$00(Y)         *real part Xm+1(q) to BRAM
      MAB:NOP $03,$01(Y)         *imag part Xm+1(q) to BRAM
      -------------------
      -------------------        *INDEXING AND LOOP COMMANDS
      -------------------
```

Figure 21. FFT Butterfly Routine for 64-point FFT.

CONCLUSIONS

The Fujitsu MB8764 digital signal processor was found to be a powerful processor capable of performing very fast multiply and sum routines. This speed enables it to solve a second-order binomial equation in 1.6 μsec, a 64-point FFT in .6 μsec, and a 64-point DFT in 9.0 msec. An eighth-order digital IIR filter implemented in a parallel form can operate with a sample rate of 149.25 KHZ. The weakness in the Fujitsu chip lies in its internal precision. With only 16 bits internal precision, sample rates greater than five times the maximum signal frequency may be too great for the internal precision of the Fujitsu. Increasing the chips internal precision to 24 bits is possible by using two words for internal data transfer and coefficient storage, and by shifting the D register so that the lower-order bits can be transferred out. This procedure is cumbersome and would slow down processing by at least a factor of ten. Double precision operations are not possible because the D register carries only 26 bits.

The MB8764 allows for external expansion of ROM and RAM. When ERAM is used either the instruction cycle must be 1.25 μsec or less, or the ERAM speed switching option must be utilized. This option, selected by an external pin, allows ERAM to be accessed at half the rate of the instruction cycle. A DFT, FFT or digital filter program which uses ERAM will run faster with the 1.25 μsec machine cycle than with a 0.1 μsec instruction cycle and the ERAM show speed option selected. RAM and ROM are divided into pages with the RAM having 256 words per page. This paging causes problems in any program that works with more than a page of data or coefficients. DFT calculations for more than 64 points, although possible on the MB8764, are difficult to program and slow to operate because of this paging problem.

The input/output features on the MB8764 can be used to govern the sample rate of a digital filter. This is done by using a jump instruction that prevents program execution from continuing until an input is received. The address attached input mode allows specific coefficients of a digital filter to be changed during program execution. Thus a designer can produce a digital filter that reacts to various parameters and compensates its transfer function to accommodate the parameter changes.

Instructions are designed to take advantage of the separate sides of RAM and their indexes. This makes programming on the MB8764 most efficient when ARAM and BRAM or table ROM and BRAM can be used independently. When this separation cannot be used by an application the MB8764 becomes awkward in its internal data transfer. Thus the

MB8764 is not a general purpose microprocessor but is specifically designed for digital signal processing or similar arithmetic operations.

The MB8764 helps to prevent overflow in preliminary operations from occuring by providing two bits to the left of the decimal point in the D register. The data format in the input/output and storage registers allows for one bit to the left of the decimal point. If input signals are restricted to +/- one, scaling of the input signal is unnecessary.

Specifications of the MB8764 claim it can implement a second-order filter in 0.7 μsec. It should be noted that the second-order filter to which this specification applies is a second-order FIR filter.

REFERENCES

1. *MB8764 Programming Manual*. Tokyo, Japan: Fujitsu Limited.

2. *MB8764 Hardware Manual*. Tokyo, Japan: Fujitsu Limited.

3. Booth, Andrew D. "A Signal Binary Multiplication Technique," Quarterly J. Mechanical Application Math., Vol. 4, pp. 236-240. Reprinted by Earl E. Swartzkander, Jr., ed. *Benchmark Papers in Engineering and Computer Science/21 Computer Arithmetic*. Stroudsburg, Pennsylvania: Dowden, Hutchinson and Ross, 1980.

4. Canright, Robert Eldon, Jr. "Digital Filtering with the iAPX 86/20." Research Paper, University of Central Florida, 1983.

5. Phillips, Charles L., and Nagle, H. Troy, Jr. *Digital Control System Analysis and Design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1984.

6. *MB87902 The Fujitsu MB8764 Support Tool Outline*. Tokyo, Japan: Fujitsu Limited.

7. *MB87902 Software Development Tool Kit for MB8764 Digital Signal Processor Detailed Description*. Tokyo, Japan: Fujitsu Limited, 1984.

8. Oppenheim, Alan V., and Schafer, Ronald W. *Digital Signal Processing*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975.