Order this document
by AN1123/D

MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

AN1123

MCS3201 Floppy Disk Controller in MC68000 System

Prepared by: Ajay Matani
Motorola Semiconductor Product Sector
Anaheim, CA

INTRODUCTION

MCS3201 provides a highly integrated system solution for Floppy Disk control in the IBM/PC environment. The FDC functionality in the MCS3201 is based around an industry standard core with an array of programmable features providing interface to low cost IBM/PC compatible Floppy Disk Drives for 5.25" and 3.5" media.

Development of peripheral chips to interface with the MC68HC000 bus has been virtually ceased in favor of high integration products in the MC68300 family of controllers. Providing a low cost interface between the CMOS 68K member and intelligent peripheral subsystems like the MCS3201 FDC results in a mutual benefit to Motorola and customers who are looking for low cost solutions with high performance.

This study examines the role of the FDC in a typical system emphasizing various host interface strategies. The functionality provided by the MCS3201 in terms of the Floppy Disk Control and its merit are not considered in detail here as the said controller does have an industry standard low cost solution with a rich feature list.

The study is presented in the following format:

- Introduction to the MCS3201
- Data transfer analysis and DMA requirement in a MC68HC000 based system
- Description of the hardware interface to the MCS3201
- Analysis of the software structure
- Adapting to other M68K family members
- Summary

INTRODUCTION TO THE MCS3201

The MCS3201 is a single chip solution for the Floppy Disk Control sub-system in an IBM/PC. The "host" interface is to the so called "PC-BUS" which includes the address decoding of the I/O space on the "PC-BUS" as well as the DMA interface. The MCS3201 itself is based around a core FDC consisting of a microcoded sequencer to emulate and enhance the command set of industry standard NEC765A in an IBM environment. This microcoded design enables enhancing the functionality of the FDC sub-system as well as provides for an efficient testing environment.

The interface to the Floppy Disk Drive Electronics is implemented on board resulting in fast access time, high reliability and low cost per bit capability. Refer to the data sheet MCS3201/D for further information.

On the Host Interface end, the decode logic to map the MCS3201 at the appropriate I/O address location in the IBM/PC's 1K I/O space defined by SA[9:0] is integrated along with the $\overline{\text{IOR}}$ and $\overline{\text{IOW}}$ strobes and support for the DMA accesses by providing the AEN, DRQ.$\overline{\text{DACK}}$ and T/C signals. As far as the accessible range of the registers on the MCS3201 are concerned only SA[2:0] are needed to address the six distinct locations.

The FDC core module, as shown in the data sheet MCS3201, runs the command and data sequences with reference to the FCLK and WCLK clocks provided by the CLOCK GENERATOR that uses a 24 MHz nominal f_{OSC} crystal. There are four registers within the FDC core out of the total of seven seen in the MCS3201: Diskette Control Register—Write Only at \$3F7, Main Status Register—Read Only at \$374 and the Data Read Register and Data Write Register at \$3F5. Accesses to all of these are asynchronous and hence the Host Interface Timing is the same for all the registers on the MCS3201.

On the other hand, the events in the FDC core that require information exchange are synchronized by the core and the Host has to wait or provide long strobes while interfacing.

ACCESS CYCLE DETAILS

(Note: Refer to the AC characteristics in the MCS3201 data sheet if needed.) A simple READ operation of the register file is carried out by asserting the SA[9:0] and $\overline{\text{CS}}$ active along with AEN and $\overline{\text{DACK}}$ negated 25 ns before the $\overline{\text{IOR}}$ is asserted to meet the t_{AR} (min) parameter. For up to a 100 pF load on the data bus, the access time for the valid data on the bus is 80 ns (t_{1RD} min) and the data is guaranteed to be valid on the SD[7:0] bus for at least 10 ns (t_{DH} min).

For the write case, the addresses need to be valid as in the read case before the $\overline{\text{IOW}}$ is asserted with identical timing as t_{AW} min = t_{AR} min. The important timing is to provide valid data 60 ns (t_{DW} min) before the $\overline{\text{IOW}}$ is negated with zero hold time requirement. The address information in both the cases has to be valid until the negation of the appropriate strobe (t_{RA} min = t_{WA} min = 0 ns).

For the DMA accesses, AEN needs to be asserted to ignore the address decoder output, but $\overline{\text{DACK}}$ also needs to be asserted. Depending on the transfer direction, $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ need to be strobed with same AC timing as for the normal access, with $\overline{\text{DACK}}$ asserted before the access strobes.

During the DMA access, $\overline{\text{DACK}}$ and TC (when needed) must be wide pulses because they are synchronized by the core. The timing requirement varies, depending upon the data rate at which the floppy disk is operating. The $\overline{\text{DACK}}$ width $t_{W(AA)}$ min = 260 ns for the 500 kb/s disks and = 510 ns for the 250 kb/s medium. Note that TC width $t_{W(TC)}$ min = ($t_{W(AA)}$ min + 10)/2 i.e. = 5 ns more than half of the $\overline{\text{DACK}}$ width.

DATA TRANSFER ANALYSIS

Depending on the bit rate of data accessed from the diskette, nominal rate at which data needs to be provided by the system may be calculated as:

t_{DYC} typ = 8/$f_{Bit\ Rate}$ µs for Bit Rate specified in kb/s

Therefore, we have 16 µs, 26.67 µs, and 32 µs as the nominal cycle times. Due to delays in the FDC core and other synchronization for MFM recording at 500 kb/s, t_{DCY} min is specified as 13 µs. We will consider only this fastest case for further analysis, unless otherwise noted.

Note that this minimum cycle time would not be a sustained minimum, but would typically exist at the onset of a data transfer sequence. Once data transfer sequence within a sector begins, t_{DCY} min would tend to attain its average value with high probability. This concept is important to the solution proposed here.

The other important parameter is the service delay from a request to transfer (t_{PMRW} min), which is the response time of $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ from the DRQ request. It is specified as 12 µs for the fastest case.

In a low cost system using the MC68000, providing hardware for the DMA transfers may not be possible; therefore, the CPU needs to be involved in the data transfer task during the EXECUTION phase of a command sequence. This may be achieved by CPU responding to the transfer needs in either POOLED or INTERRUPT driven mode.

The case of the CPU polling is trivial when the MCS3201 is programmed to work in a non-DMA mode and the Request for Master Bit 7 of the Main Status Register is awaited before the new data is transferred. IRQ interrupt may be used in the same mode, but the ISR has the responsibility of distinguishing the phase of operation because the interrupt on the IRQ pin may be caused during the COMMAND and RESULT phases. A unified ISR is longer in execution and slower in response, and may fail to provide service during the EXECUTION phase of operation.

The key feature of the MC68000 architecture that comes to the rescue is the interrupt interface supporting multiple levels. The solution is based on operating the MCS3201 in a DMA mode requesting service with DRQ signal demanding an ISR to carry out data transfer at a high priority (say level 6). This distinguishes the ISR for the normal IRQ used during COMMAND and RESULT phase from the ISR during the EXECUTION phase to provide fast and efficient service.

HARDWARE INTERFACE

The interface described here between the MC68HC000 and MCS3201 as shown in the BLOCK DIAGRAM (Figure 1) is based on the Interrupt Driven DMA scheme suggested earlier, at minimal cost. Circuit level details of blocks that are generic in a 68K based system are not illustrated, but the requirements are outlined.

ADDRESS DECODER

The address decoder provides decoding for the MCS3201 access, preferably in the Absolute Short Address Space at the TOP of the MEMORY MAP along with other I/Os. It also provides $\overline{\text{DACK}}$ to the processor for the FDC access with zero wait state and hence should not require extra logic beyond an ORing with other $\overline{\text{DACK}}$s. The master decode line FDC_$\overline{\text{CS}}$ is provided to the Programmable Logic Device 16R4 which generates the strobes to interface with the FDC. Interrupt Acknowledge Cycles are also decoded here, providing master decode signal to the INTERRUPT LEVEL DECODER and the processor to indicate an autovector termination on the VPA line.

INTERRUPT LEVEL DECODER

This is actually an encoder for the various IRQ requests at each of the seven levels and a decoder if the interrupting devices need it. The MCS3201 does not provide any acknowledgement in terms of vector during the IACK bus cycle of the MC68000, so it does not need any specific signal; therefore, the FDC_IRQ_ack and FDC_DMA_ack signals shown in Figure 1 are not used as indicated. The acknowledgement of the interrupt is provided within the ISR for the each request from the FDC. If other interrupters do not require specific acknowledgement, there is no need to generate them.

The IRQ request from FDC is encoded at level 1—a low priority, while the DMA request signal DRQ is encoded at level 6—highest priority maskable interrupt. At level 7 in a 68K based system ABORT or RESTART request is found normally. Note that the requests are active high and need to be encoded accordingly. It is mandatory that DRQ have the highest priority since service latency requirement is quite stringent for the DATA TRANSFER request.

MCS3201 INTERFACE

Since the address decoder on board the FDC looks for \$3F on SA[9:4] when ADDSEL is high, these may be pulled high. The $\overline{\text{CS}}$ always enabled as accesses are controlled by isolating the read and write strobes in this implementation. Address buffer enable signal AEN, indicating a normal CPU access, is asserted when the lower nibble SA[3:0] is in a range of \$[7:0] and is disabled in the range \$[F:8]. This provides for distinguishing the DMA access of MCS3201.

The data access is provided by connecting the upper byte of the MC68000s data bus to the FDC so that the registers in the MCS3201 are mapped on alternate even bytes on the 68Ks memory space. In fact, although the data transfer is only on the even bytes here, the access decoder does not look for such restriction. Because of this, a byte access on the corresponding odd byte would result in an invalid access, since the lower data bus is not connected to the FDC. On the same token, a word access would only transfer data on the upper data bus but work perfectly otherwise. Such word transfers may be useful if data is to be transferred to the system stack within ISR.

Also note that DRQ request is fed back to the FDC on one of the general purpose inputs I7. This will enable the processor to read the DRQ signal if it needs to without disrupting the DMA sequence. Other inputs may be used to provide different configuration information to the processor.

INTERFACE LOGIC

A low cost generic PAL device 16R4-15 is used to provide most of the interface logic. The details of this interface may be found in the following source code in CUPLs programming language. The basic idea is to provide the $\overline{\text{IOR}}$ and $\overline{\text{IOW}}$ strobes that meet the AC timing requirements. The processor is considered to be working at its highest speed of 16.67 MHz here since slower speeds will work without any changes.

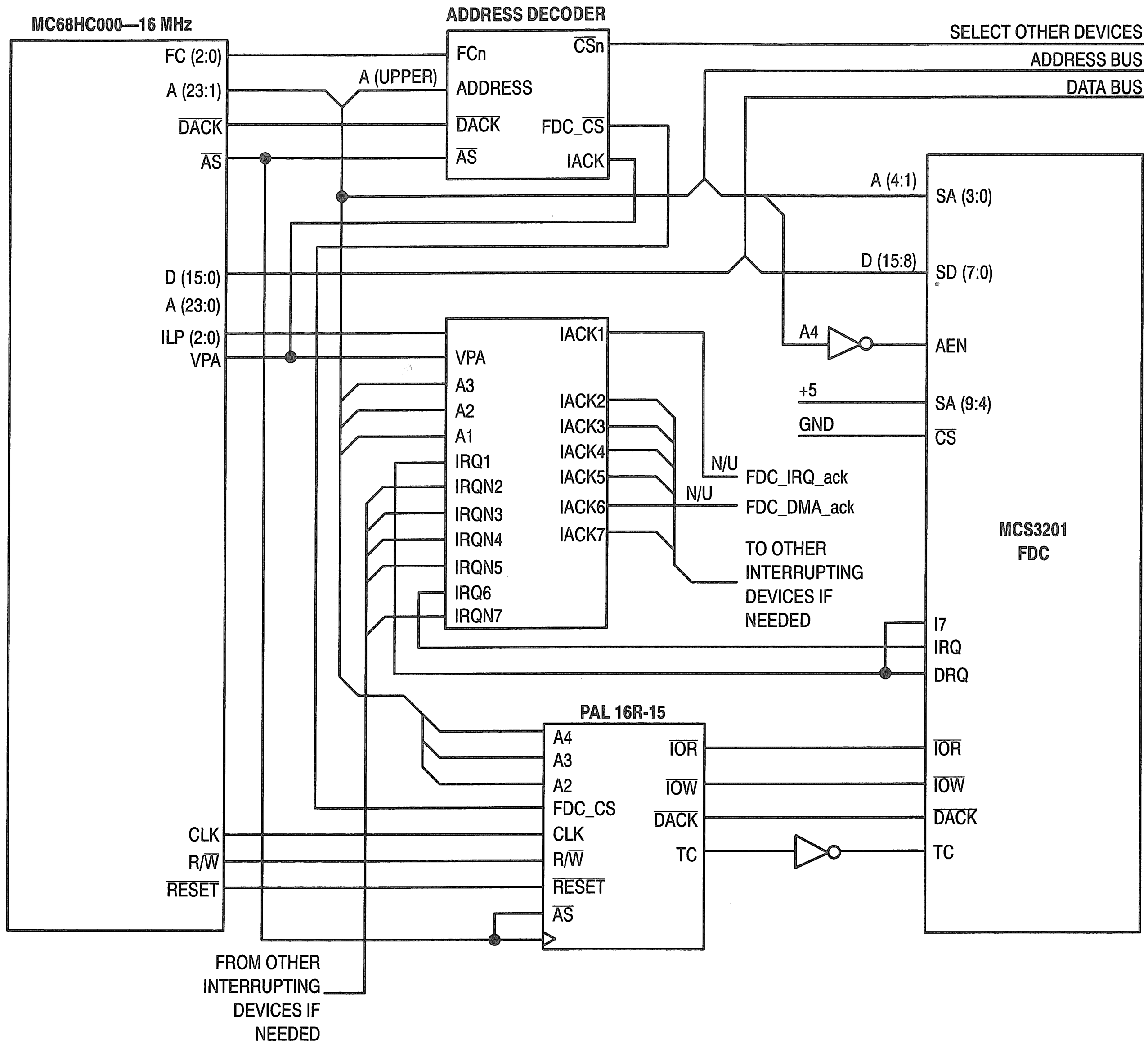

Figure 1. Block Diagram

To meet the t_{AR} min, assertion of $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ is delayed until the beginning of S3 cycle when the CLK is low. Otherwise the strobes follow the $\overline{\text{AS}}$ from the processor when FDC_$\overline{\text{CS}}$ is present. Also, during the cycles when A4 is high (i.e., DMA access), the strobes will not be asserted until the $\overline{\text{DACK}}$ is asserted.

The sequencer implemented works off the AS as clock, and changes states to a cycle by the processor at the end. To provide the extended $\overline{\text{DACK}}$ and TC pulses, if asserted during the CPU access cycle, the said signals are extended till the holding term WEND is asserted.

Depending on the address line A4 from processor, DMA or NON-DMA access is carried out. Depending on A3 during DMA access, TC is asserted indicating last transfer. A2 determines the width of the pulse for $\overline{\text{DACK}}$ and TC when asserted.

```
Name        FDCPAL  ;
Partno      MP3201P  ;
Date        12/10/90  ;
Designer    Ajay Matani  ;
Company     Motorola Semiconductor Products Inc.  ;
Assembly    None  ;
Location    None  ;
Device      p16r4  ;
Rev         01  ;

/*****************************************************************************/
/*                                                                           */
/* MCS3201 control strobe generator                                          */
/*                                                                           */
/* Generates control signals for the MCS3201 from MC68HC000 bus              */
/* running at 16.67 MHz with no wait states.                                 */
/*                                                                           */
/* Decode for various cycles                                                 */
/*                                                                           */
/* A4 A3 A2 RWN                                                              */
/*  0  x  x  1     Read cycle to access MCS3201 registers                    */
/*  0  x  x  0     Write cycle to access MCS3201 registers                   */
/*  1  0  0  1     DMA read  cycle without TC and wide pulse                 */
/*  1  0  0  0     DMA write cycle without TC and wide pulse                 */
/*  1  0  1  1     DMA read  cycle without TC and narrow pulse               */
/*  1  0  1  0     DMA write cycle without TC and narrow pulse               */
/*  1  1  0  1     DMA read  cycle with    TC and wide pulse                 */
/*  1  1  0  0     DMA write cycle with    TC and wide pulse                 */
/*  1  1  1  1     DMA read  cycle with    TC and narrow pulse               */
/*  1  1  1  0     DMA write cycle with    TC and narrow pulse               */
/*                                                                           */
/*****************************************************************************/
/* Allowable Target Device Types :  PAL16R4                                  */
/*****************************************************************************/

/** Inputs **/

Pin 1      =        CLK_ASN; /* Trailing edge of ASN is the CLOCK            */
Pin 2      =       !ASN    ; /* Address strobe from MC68HC000                */
Pin 3      =       !FDC_CS ; /* Decode for FDC access                        */
Pin 4      =        RWN    ; /* Read/WriteN strobe from MC68HC000            */
Pin 5      =        A4     ; /* A4          output from MC68HC000            */
Pin 6      =        A3     ; /* A3          output from MC68HC000            */
Pin 7      =        A2     ; /* A2          output from MC68HC000            */
Pin 8      =       !RESETN ; /* System RESET active low                      */
Pin 9      =        CLK    ; /* CLK for combinatorial logic use              */

/** Outputs **/

Pin 12     =       !IORN   ; /* Read strobe for the MCS3201                  */
Pin 13     =       !IOWN   ; /* Write strobe for the MCS3201                 */
Pin 14     =       !W0     ; /* wait sequencer bit 0                         */
Pin 15     =       !W1     ; /* wait sequencer bit 1                         */
Pin 16     =       !WEND   ; /* wait sequencer end condition                 */
Pin 17     =       !SPARE  ;                                                 */
Pin 18     =       !DACKN  ; /* DMA acknowledge strobe                       */
Pin 19     =       !TC     ; /* Terminal count on DMA transfer               */
                             /*  inverted outside                            */

/** Declarations and Intermediate Variable Definitions **/

$define _IDLE    'b'00       /* Default state, waiting for new bus cycle     */
$define _WAIT1   'b'01       /* Used to extend DACK and TC signals           */
$define _WAIT2   'b'11
$define _WAIT3   'b'10

field STATES  =  [W1..0];
```

```
Sequence STATES  {
  present _IDLE              /* Default state                                     */
    if !RESETN & DACKN &  A4 & !A2
      next _WAIT2 ;
    if !RESETN & DACKN &  A4 &  A2
      next _WAIT3 ;

  present _WAIT1             /* Unused state                                      */
      next _IDLE ;

  present _WAIT2             /* For wider DACK and TC pulses                      */
    if !RESETN
      next _WAIT3 ;

  present _WAIT3             /* For DACK and TC pulses, narrow or wide            */
    if !RESETN
      next _IDLE out WEND.d ;/*asserting WEND completes the access               */
}

/** Logic Equations **/

IORN =   !RESETN & (
          !CLK & ASN & FDC_CS & !A4 &  RWN             /* init term in S3          */
        # !CLK & ASN & FDC_CS &  A4 &  RWN & DACKN     /* init term, DACK setup    */
        #  & ASN & IORN ) ;                            /* holding term             */

IOWN =   !RESETN & (
          !CLK & ASN & FDC_CS & !A4 & !RWN             /* init term in S3          */
        # !CLK & ASN & FDC_CS &  A4 & !RWN & DACKN     /* init term, DACK setup    */
        #  & ASN & IOWN ) ;                            /* holding term             */

DACKN = !RESETN & ASN & FDC_CS &  A4                   /* init term                */
        # !RESETN & DACKN & !WEND ;                    /* holding term             */

TC =     !RESETN & ASN & FDC_CS &  A4 & A3             /* init term                */
        # !RESETN & DACKN ;                            /* holding term             */
```

ANALYSIS OF SOFTWARE STRUCTURE

The access to FDC can be restricted only in a supervisor mode, but a typical way to provide support for applications is through a SOFTWARE INTERRUPT or TRAP to the OS or BIOS (Basic Input Output System) equivalent service. The device driver for the MCS3201 would carry out low level tasks as service to the OS, in terms of synchronizing with the COMMAND, EXECUTION, and RESULT phases. A typical transaction would follow a sequence like this:

1. Application program or the Memory Manager requests a Disk Access Service (e.g., Read, Write, Format)
2. The OS queues the request for the Device Driver after validation of request.
3. Device Driver verifies the transaction request and responds with positive or negative status.
4. OS schedules CPU time to the Device Driver.
5. Device Driver interfaces with the FDC through COMMAND, EXECUTION, and RESULT phases during its allocated time slice.
   During the COMMAND and RESULT PHASE, it is recommended to poll the **Request For Master (Bit 7)** in the **Main Status Register** between the transfer of bytes to/from the FDC.
   During the EXECUTION phase, an interrupt driven Data Transfer is carried out while the Device Driver waits for an interrupt from the FDC indicating the end of the executing phase.
6. For most of the command sequences, the device driver waits between the COMMAND and the subsequent phases (remember that not all commands have an EXECUTION phase) as there is physical movement involved at the Disk Drive during that time.
   Other tasks may be carried out during such periods, and an OS supporting multi-tasking certainly would appreciate the time.
   On the other hand, during the execution phase, **Service Latency of 12 μs** and **Cycle time of 13.5 μs** for the data transfers is specified for the MCS3201.

The most time critical element of this driver is the data transfer during the EXECUTION phase. The following is the analysis of the code which provides this service in low cost systems.

SYSTEM VARIATIONS AND ASSUMPTIONS

For the hardware interface described above, the mapping of ROM, RAM, Interrupt Vector Table, and the MCS3201 within the 16M space, plays an important role on the execution speed of the Interrupt Driven Data Transfer Task and guarantees the Service Latency.

- Vector Table in ROM forces the ISR to determine the Data Transfer Direction, thereby increasing the Service Latency and the cycle time.
- RAM for storing temporary variables for the Device Driver is used if available in Absolute Short Address Space, and minimizes the time to access them.

- Mapping MCS 3201 at the top of the Map (where most I/Os in typical system are found) again will keep the accesses in the Absolute Short Address Space.
- Accesses to ROM and RAM are zero wait state, or 4 cycle is assumed because slower memory system would degrade the response time of the ISR.

The code sequence for the ISR to carry out Data Transfer is illustrated here to reflect these different scenarios.

Variable Name	Size	Location	Description
far_byte_buf	byte	absolute long address	temp holding buffer
near_byte_buf	byte	absolute short address	temp holding buffer
far_fdc_data	byte	absolute long address	FDC data reg—DMA mode
near_fdc_data	byte	absolute short address	FDC data reg—DMA mode
far_fdc_last_data	byte	absolute long address	FDC data reg—DMA mode—w/TC
near_fdc_last_data	byte	absolute short address	FDC data reg—DMA mode—w/TC
far_dma_count	word	absolute long address	DMA transfer counter
near_dma_count	word	absolute short address	DMA transfer counter
far_pointer	lword	absolute long address	DMA address pointer
near_fdc_data	lword	absolute short address	DMA address pointer

SIMPLE DEVICE DRIVER PSEUDO CODE

ENTRY_TASK

```
jsr parse_command
jsr do_set_up_dma
jsr do_fdc_command_phase
jsr wait_for_result_phase
end_task
```

EXECUTION PHASE

```
inst_in_progress                            nn
interrupt_sequence (auto vector)            50
isr_body
```

RESULT_TASK

```
jsr_do_fdc_result_phase
jsr_do_validate_completion
end_task
```

EXECUTION PHASE DETAILS

Instruction in progress must terminate before the exception processing can begin. In the most general case, this could be a very long instruction like ***movem all the registers to/from the stack with a bus error in the last transfer*** and would defeat the whole concept here. With certain restrictions in the system design, in both software and hardware the "worst case execution time—nn" can be restricted so we will keep this as a parameter "nn" that needs to be maximized in our analysis here.

Upon recognition of the interrupt, the "interrupt exception execution time" is 44 clocks with 4 cycle IACK cycle for the MC68000, but for our system as we use auto-vector we add 6 more clocks. Therefore, there is a total interrupt latency of "nn + 50 clocks" before the **Isr_body** is executed.

We consider four cases here with different system implementation flavors resulting in ISRs that provide cost savings at a performance penalty. An appropriate scheme may be used depending upon the kind of Floppy Disk Drives used that require different bit rates. To measure the performance of these four cases against the CPU's operating speed and media densities supported, let's look at the requirements in terms of CPU clock cycles to achieve proper operation:

Bit Rate	$t_{DCY}typ$	$t_{DCY}min$	$t_{PMRW}min$
500 kb/s	16.0 µs	13 µs	12 µs
300 kb/s	26.5 µs	22 µs	20 µs
250 kb/s	32.0 µs	27 µs	24 µs

For 16.67 MHz CPU clock:

Bit Rate	$k_{DCY}typ$	$k_{DCY}min$	$k_{PMRW}min$
500 kb/s	266	217	200
300 kb/s	442	367	333
250 kb/s	533	450	400

For 12 MHz CPU clock:

(Low cost solution utilizing the OSC on the MCS3201 for CPU clock generation)

Bit Rate	$k_{DCY}typ$	$k_{DCY}min$	$k_{PMRW}min$
500 kb/s	192	156	144
300 kb/s	318	264	240
250 kb/s	384	324	288

isr_body case 1

interrupt vector table in ROM i.e. not alterable
all variables are absolute long
fdc registers are absolute long

dma_count	holds the transfer count as positive integer for read case and (count–8001h) for the write case

```
isr_body
      subq.w       #1,far_dma_count                       20
      blt          write_case                              8/10
      beq          do_last_read                            8/10
      move.b       far_fdc_data,far_byte_buf              28
cont_read
      move.l       a0,-(sp)                               14
      move.l       far_pointer,a0                         20
      move.b       far_byte_buf, (a0)+                    20
      move.l       a0,far_pointer                         20
      movea        (sp)+,a0                               12
      rte                                                 20
do_last_read
      move.w       far_fdc_last_data,far_byte_buf         28
      bra          cont_read                              10

write_case
      bvs          do_last_write                           8/10
      move.b       far_byte_buf,far_fdc_data              28
cont_write
      move.l       a0,-(sp)                               14
      move.l       far_pointer,a0                         20
      move.b       (a0)+,far_byte_buf                     20
      move.l       a0,far_pointer                         20
      movea        (sp)+,a0                               12
      rte                                                 20
do_last_write
      move.w       far_byte_buf,far_fdc_last_data         28
      bra          cont_write                             10

      INTERRUPT ENTRY OVERHEAD          SUB_TOTAL          nn+50
      DMA access delay                  SUB_TOTAL          nn+50+64 = nn+114
      Rest of the ISR                   SUB_TOTAL         106
                                        TOTAL             220+nn
```

isr_body case 2

interrupt vector table in ROM i.e. not alterable
all variables are absolute short
fdc registers are absolute short

dma_count	holds the transfer count as positive integer for read case and (count–8001h) for the write case

```
isr_body
        subq.w      #1,near_dma_count                       16
        blt         write_case                               8/10
        beq         do_last_read                             8/10
        move.b      near_fdc_data,near_byte_buf             20
cont_read
        move.l      a0,-(sp)                                14
        move.l      near_pointer,a0                         16
        move.b      near_byte_buf, (a0)+                    16
        move.l      a0,near_pointer                         16
        movea       (sp)+,a0                                12
        rte                                                 20
do_last_read
        move.w      near_fdc_last_data,near_byte_buf        20
        bra         cont_read                               10

write_case
        bvs         do_last_write                            8/10
        move.b      near_byte_buf,near_fdc_data             20
cont_write
        move.l      a0,-(sp)                                14
        move.l      near_pointer,a0                         16
        move.b      (a0)+,near_byte_buf                     16
        move.l      a0,near_pointer                         16
        movea       (sp)+,a0                                12
        rte                                                 20
do_last_write
        move.w      near_byte_buf,near_fdc_last_data        20
        bra         cont_write                              10

        INTERRUPT ENTRY OVERHEAD      SUB_TOTAL             nn+50
        DMA access delay              SUB_TOTAL             nn+50+52 = nn+102
        Rest of the ISR               SUB_TOTAL             94
                                      TOTAL                196+nn
```

isr_body case 3

interrupt vector table in RAM i.e. ISR vector programmed during set_up_dma
all variables are absolute short
fdc registers are absolute short
dma_count holds the transfer count as positive integer for read or write case

```
isr_body_read
      subq.w      #1,near_dma_count                       16
      beq         do_last_read                             8/10
      move.b      near_fdc_data,near_byte_buf             20
cont_read
      move.l      a0,-(sp)                                14
      move.l      near_pointer,a0                         16
      move.b      near_byte_buf, (a0)+                    16
      move.l      a0,near_pointer                         16
      movea       (sp)+,a0                                12
      rte                                                 20
do_last_read
      move.w      near_fdc_last_data,near_byte_buf        20
      bra         cont_read                               10

isr_body_write
      subq.w      #1,near_dma_count                       16
      beq         do_last_write                            8/10
      move.b      near_fdc_data,near_byte_buf             20
cont_write
      move.l      a0,-(sp)                                14
      move.l      near_pointer,a0                         16
      move.b      (a0)+,near_byte_buf                     16
      move.l      a0,near_pointer                         16
      movea       (sp)+,a0                                12
      rte                                                 20
do_last_write
      move.w      near_byte_buf,near_fdc_last_data        20
      bra         cont_write                              10

      INTERRUPT ENTRY OVERHEAD          SUB_TOTAL          nn+50
      DMA access delay                  SUB_TOTAL          nn+50+44 = nn+94
      Rest of the ISR                   SUB_TOTAL          94
                                        TOTAL             188+nn
```

isr_body case 4

interrupt vector table in RAM i.e. ISR vector programmed during set_up_dma
all variables are absolute short
fdc registers are absolute short
dma_count holds the transfer count as positive integer for read or write case
shorter cycle time, penalty on latency of service

```
isr_body_read
      subq.w        #1,near_dma_count                        16
      beq           do_last_read                              8/10
cont_read
      move.l        a0,-(sp)                                 14
      move.l        near_pointer,a0                          16
      move.b        near_fdc_data, (a0)+                     16
      move.l        a0,near_pointer                          16
      movea         (sp)+,a0                                 12
      rte                                                    20
do_last_read
      move.w        near_fdc_last_data,near_byte_buf         20
      bra           cont_read                                10

isr_body_write
      subq.w        #1,near_dma_count                        16
      beq           do_last_write                             8/10
cont_write
      move.l        a0,-(sp)                                 14
      move.l        near_pointer,a0                          16
      move.b        (a0)+,near_fdc_data                      16
      move.l        a0,near_pointer                          16
      movea         (sp)+,a0                                 12
      rte                                                    20
do_last_write
      move.w        near_byte_buf,near_fdc_last_data         20
      bra           cont_write                               10

      INTERRUPT ENTRY OVERHEAD         SUB_TOTAL             nn+50
      DMA access delay                 SUB_TOTAL             nn+50+70 = nn+120
      Rest of the ISR                  SUB_TOTAL             48
                                       TOTAL                168+nn
```

PERFORMANCE ANALYSIS

The performance of different cases is summarized below in terms of the available clock cycles $nn_{max} = m$ that the instruction in progress may take before exception handling begins.

Case 1

	16.67 MHz			12 MHz		
Bit Rate	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}
500 kb/s	46	–3	86	–28	–64	30
300 kb/s	222	147	219	98	44	126
250 kb/s	313	230	286	167	104	174

Case 2

	16.67 MHz			12 MHz		
Bit Rate	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}
500 kb/s	70	21	98	–4	–40	42
300 kb/s	246	171	231	122	68	138
250 kb/s	337	254	298	188	128	186

Case 3

	16.67 MHz			12 MHz		
Bit Rate	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}
500 kb/s	78	29	106	4	–32	50
300 kb/s	254	179	239	130	76	146
250 kb/s	354	262	306	196	136	194

Case 4

	16.67 MHz			12 MHz		
Bit Rate	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}	m_{DCY}^{typ}	m_{DCY}^{min}	m_{PMRW}^{min}
500 kb/s	98	49	80	24	–12	24
300 kb/s	274	199	213	150	96	120
250 kb/s	365	282	280	216	156	168

OBSERVATIONS

- Because m_{DCY}typ is reasonably positive, even if the minimum value is slightly negative there is a high probability of successful operations. In the WORSE CASE, a data OVERRUN or UNDERRUN may occur, which may be taken care of by retrying the requested operation.
- Case 4 implementation differs from the rest because the critical parameter is the response time and not the cycle time.
- With Case 4 implementation for 16.67 MHz CPU operation with 500 kb/s disks, 80 clocks or 20 memory cycle worth of uninterruptable task may be supported in the worst case.
- With Case 4 implementation for 16.67 MHz CPU operation with 500 kb/s disks, Case 3 could be the best implementation since the alternate task would rarely have long uninterruptable sequence, and if it did, higher latency to service is more appropriate.
- For the 12 MHz CPU operation with 500 kb/s disks, only Case 4 implementation is feasible. In the worse case operation a new DRQ may be generated 12 clocks before the ISR to service the previous request completed. This leaves only 12 blocks for the uninterruptable task. This certainly excludes lower priority interrupts being serviced, because they take at least 50 clocks.
- For lower data rate media, the solution here would run reasonably well at 12 MHz, and would provide a very low cost solution. At 16.67 MHz one can expect that no data OVERRUN or UNDERRUN condition will occur.

INTERFACING WITH OTHER M68000 FAMILY MEMBERS

The interface approach shown here will work with MC68020, particularly with the new MC68EC020 being available. There are four distinct advantages with the MC68020 that guarantee successful operation:

1. Autovector IACK cycle is only 3 clocks, instead of 10 for the MC68000.
2. 32-bit data bus with minimum 3 clock access instead of 4 clocks in the MC68000 speeds up the ISR.
3. For critical cases, the Instruction Cache would hold the ISR because other tasks will not execute much.
4. Memory indirect addressing mode eliminates the need to save the address register as the DMA pointer may be retained in the memory and also the transfer counter may not be explicit.

For the new M68300 family this approach also would work effectively. In fact, for MC68302 and MC68340 there is a DMA controller on board and there is no need to follow this solution. For the other members like MC68330, MC68331, and MC68332 the following advantages exist:

1. Explicit IRQ requests could generate the correct vector (or autovector may be used).
2. Instruction pipelining and 2 clock minimum memory access cycle cut down ISR latency and cycle time significantly.

3. The external PAL device may be eliminated by using two different Chip Selects where the DMA access will have a number of wait states.

SUMMARY

Due to its low cost but superior architecture and memory interface, M68000 is finding homes in embedded systems that require more performance, larger memory, and were using 8-bit MCUs like the MC6809, i8085, or Z80s. Data logging equipment that requires backup support, like Word Processors, Data Entry terminals, or Docking stations for handheld instruments/scanners may want an interface to the Floppy Disk Drives and may be compatible with the PC format.

We have shown here that MCS3201 with a simple interface can provide the solution at a low cost and at a major advantage over other MCUs. This solution includes provision for a multitasking kernel that would support data entry from the keyboard or communication over the serial cable, which is desired while the FDC is being interfaced. In a typical polling implementation, the back-up task hinders operation of the end application, which result in degraded user interface.

Also note that costly DMA interfaces may be avoided in a M68000 system if the multi-level interrupt interface is utilized appropriately.

This design example also opens up a new family of low cost peripherals that are used in the IBM/PC and compatible for consideration in a M68K based system.

REFERENCES

- **MCS3201/D.** Motorola Semiconductor Technical Data on IBM PC/XT/AT Floppy Disk Formatter/Controller, Advance Information.
- **M68000UM/AD Rev 5.** M68000 User's Manual
- **MC68000/AC REV 3.** MC68000 Programming Reference Card
- Intel's Data Book on **PERIPHERALS.** Order #296467

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Center; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MOTOROLA

A30182 PRINTED IN USA 10/91 IMPERIAL LITHO 81779 18,000 MOS D-A YFAAAA