Atari ST
FD/HD Programming

Jean Louis-Guerin (DrCoolZic)
V1.2 – November 2019

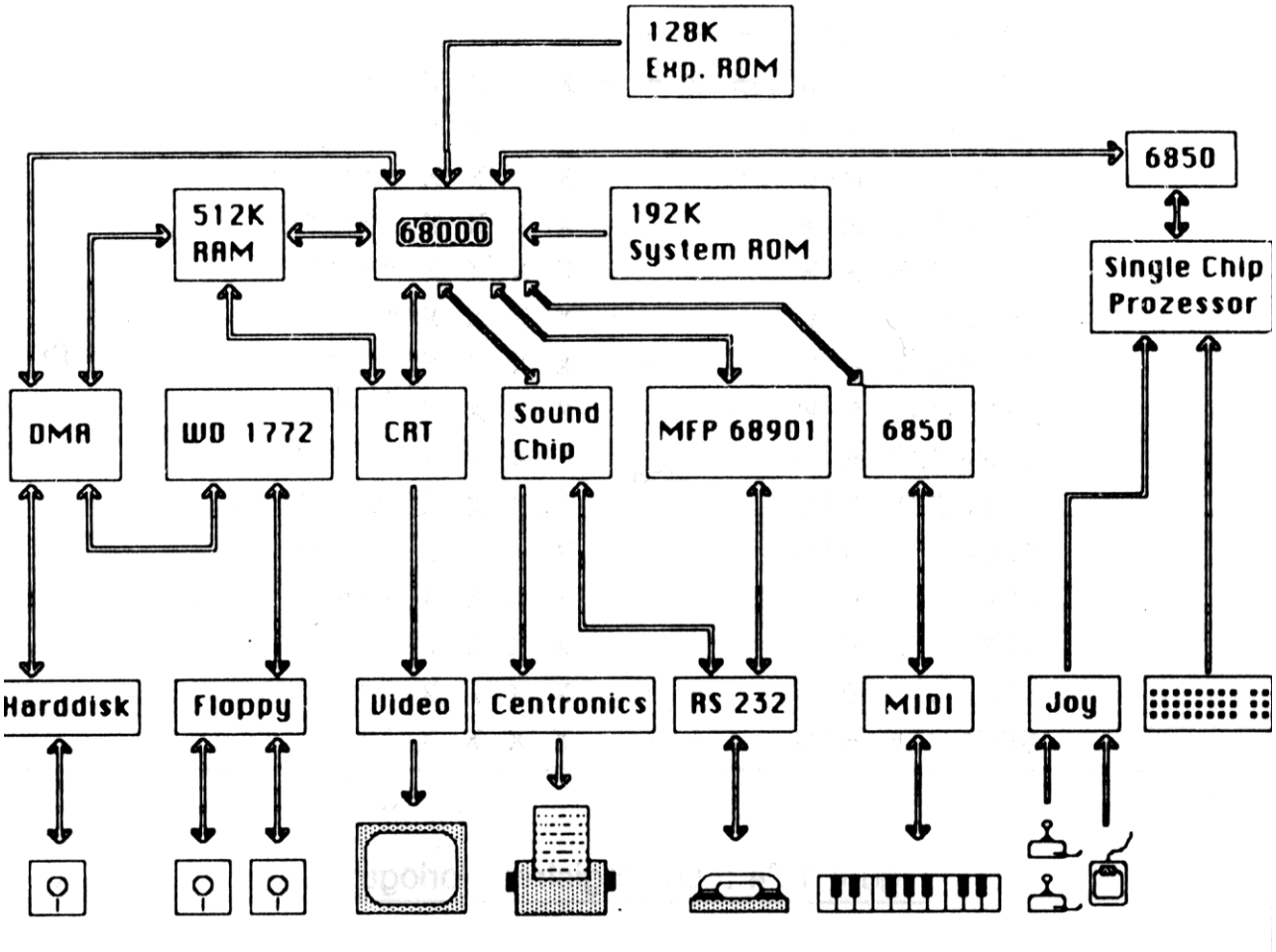
Table of Content

Introduction

The goal of this document is to provide the information necessary to write programs to access directly, at the hardware level, the Atari ST hard disk drives and floppy disk drives[1]. Therefore, I do not cover any of the BIOS/XBIOS and GEMDOS calls as I am bypassing completely the TOS environment. I first describe the Atari hardware involved in connecting disk drives, then I look at the programming of the Atari chips involved, and finally I provide a detail description of the steps required to access floppy drives and hard drives.

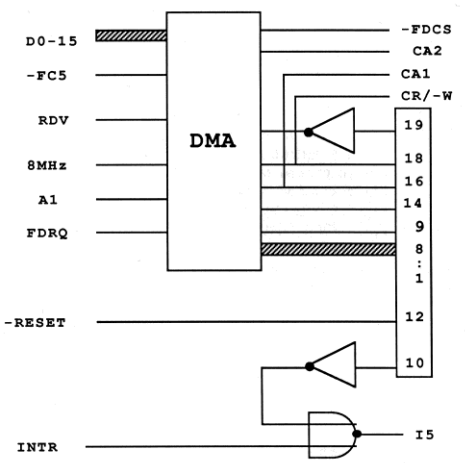
Description of the Atari Disk Drive Hardware Interface

Accessing of Atari disk drives involves many chips. Mainly: the DMA, the MMU, the Glue, the FDC, the MFP, and the PSG chips. The following diagram shows a simplified view of the Atari ST architecture:



As we can see the Atari ST uses a WD1772 FDC controller to interface the floppy drives and hard disk controllers (located outside of the Atari) to access hard drives. In the Atari system architecture the FDC/HDC data and address busses are connected to a private bus locates behind the DMA controller. In other word the DMA seats between the processor bus and the FDC/HDC. This allows the DMA controller to perform automatic transfer between the FDC, or an external Hard Disk Controller, and the memory. But this also implies that all accesses to the FDC/HDC registers have to be done through the DMA controller. The floppy drives controller is also connected to three bits of the output **Port A** of the PSG (YM2149) to control the selection of the ***drives*** and the ***side***. The hard disk controller is connected to the Atari through the DMA connector. The interrupt request of the FDC/HCD is connected to an input of the MFP (68901) general purpose I/O port (GPIO). This allows checking when an FDC/HDC command is terminated by either polling this input or by triggering an interrupt.

[1] Currently only FD is completely covered. HD information will be completed in upcoming versions.

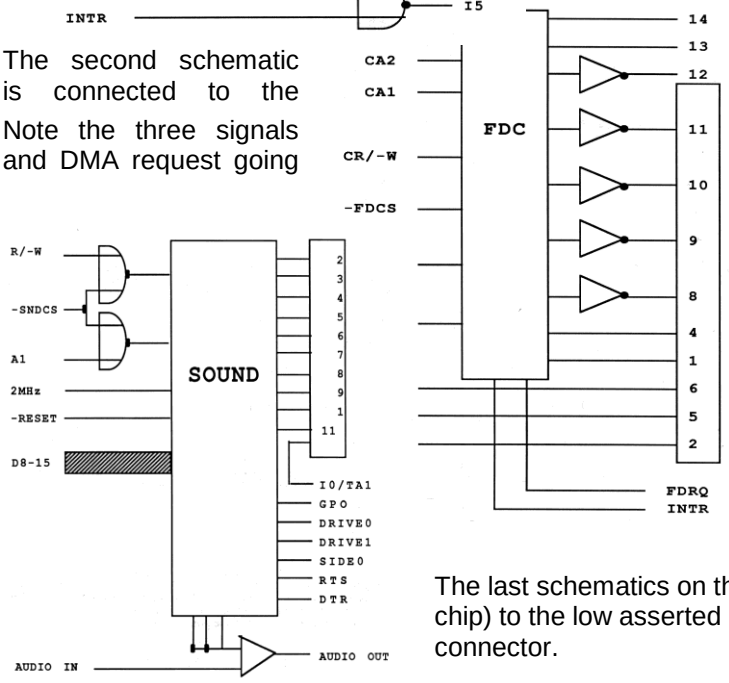


The first schematic on the left shows the DMA and its connection to the FDC and the DMA/ACSI external Hard Disk interface. We can see that the DMA private data buss (CD0-7) goes to the FDC controller as well as to the ACSI/DMA 19 pins connector. Several control signal are tranferred directly to the DMA. The FDC interrupt is “ored” with the external HD interrupt resulting to a signal I5 that is connected to bit 5 of the MFP general purpose interface.

The FDC and DMA chips both receive an 8 MHz system clock which is used by their internal micro-machines.

The second schematic on the right shows the FDC controller which is connected to the Floppy drives connector.

Note the three signals coming from the PSG chip and the interrupt and DMA request going to the MFP & DMA chips.



The last schematics on the left shows the connection of the PSG (sound chip) to the low asserted **DRIVE0***, **DRIVE1***, **SIDE0*** signals of the FDC connector.

Memory / DC Data Transfers

Read Transfers from FDC to Memory

This is a short presentation of the mechanisms involved in transferring bytes from the FDC to the memory through the DMA (e.g. when executing a read track). The following description is for the FDC but we will see later that the same principles apply to a HDC.

We first have to reset the DMA, set it to read transfer mode, fill the buffer address register, and fill the byte count register. The buffer address must point to a memory big enough to contain all the data to be read from the disk controller (e.g. about 6600 bytes in case of a read track from a floppy drive), and the count should indicate the number of 512 bytes’ chunks that the DMA will have to transfer (for a floppy read track we can use a large number like 20).

Then we have to send the necessary data to the FDC registers to start the execution of a read command (e.g. for a read track we have to fill the track register and send the ***read track*** command). During execution of the read command, as soon as an 8 bits’ byte is assembled in the FDC, a request is made to the DMA to start a fetch cycle. Internally the DMA has two 16 bytes FIFOs that are used alternatively. This feature allows the DMA to continue to receive bytes from the FDC/HDC controller while waiting for the processor to read the other FIFO. When a FIFO is full a bus request is made to the 68000 and when granted, the FIFO is transferred in 16 bits’ word to the memory. This continues until all bytes to be read have been transferred.

At the end of the command the FDC will raise an interrupt to signal the end of the command and we will have to check that the command has terminated properly.

Write Transfers from Memory to FDC

This is a short presentation of the mechanisms involved for transferring bytes from the memory to the FDC through the DMA controller (e.g. writing a track). Same principle would apply to transfer with a HDC.

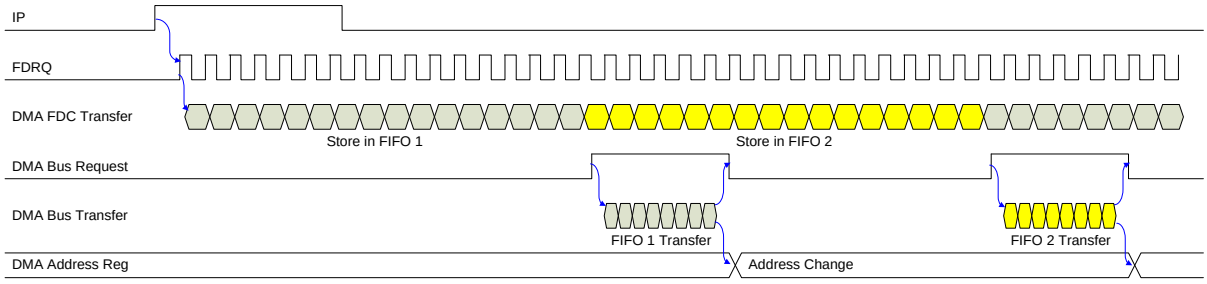
We first have to reset the DMA, set it to write transfer mode, fill the buffer address register, and fill the byte count register. The buffer address must point to the memory that contains the data to write to the FDC (e.g. about 6600 bytes in case of a write track), and the count should indicate the number of 512 bytes chunks that the DMA will have to transfer (for a write track we can use a large number like 20). Internally the DMA has two 16 bytes FIFOs that are used alternatively. This feature allows the DMA to continue to write bytes to the FDC/HDC controller from one FIFO while waiting for the processor to refill the other FIFO. When a FIFO is empty a bus request is made to the 68000 and when granted, the FIFO is filled from memory at the address pointed by the DMA address counter register. It is interesting to note that when the DMA is in write mode, the **two** internal FIFOS are filled immediately after the Count Register is written without waiting for the commands to be sent to the FDC controller.

We then have to send the appropriate data to the FDC registers to execute the command (e.g. for a write track we have to fill the track register and send the ***write track*** command). The FDC request bytes as needed to the DMA which will respond with write cycles. This continues until all bytes have been transferred.

At the end of the command the FDC will raise an interrupt to signal the end of the command and we will have to check that the command has terminated properly.

Transfer Chronogram

The following chronogram shows the sequence of events when reading a track through the DMA. Note that the events shown in diagram are not “scaled” correctly as the purpose of this diagram is just to show the sequence of events.



After the FDC receives the read track command, it starts the motor (MO signal) and waits for the drive speed to settle down. Once the drive as reached the correct speed it waits for the index pulse (IP signal) and then starts to assemble bits from the drive. After a byte has been received it raises a FDRQ to indicate that one byte is ready to be fetched. In response the DMA starts a fetch cycle to read the byte from the FDC data register on the private bus that joins the two chips. This byte is stored in one of the two DMA’s FIFOs and in response the floppy lower the FDRQ until the next byte is assembled. When the next byte is assembled a new transfer takes place and this repeats until 16 bytes has been stored in one of the DMA FIFO. At this point the DMA switch the reception of new bytes from the FDC to the other FIFO and issues a Bus request to the 68000 to indicate that it wants to perform a DMA transfer. When the Bus request is granted by the 68000 the DMA take over the control of the system bus and transfer **eight 16-bits** words from the FIFO into the memory pointed by its address register. At the end of the transfer the DMA release the bus to the processor and increments its internal address register by 16.

Reading data from a HDC would result in a chronogram very similar using slightly different control signals.

DMA Programming

This section gives information on programming the DMA (Direct Memory Access) chip. Note that we describe the DMA as one chip but in practice several chips are involved (DMA, MMU, and Glue for the STF and DMA, MCU for STE) but this is not relevant from the programming point of view (see DMA Block Diagram). When used in the context of the Atari ST and its high performance peripherals (FD, HD, CD …) DMA refers to direct transfer of data between the peripheral device and the computer memory without the direct intervention of the processor.

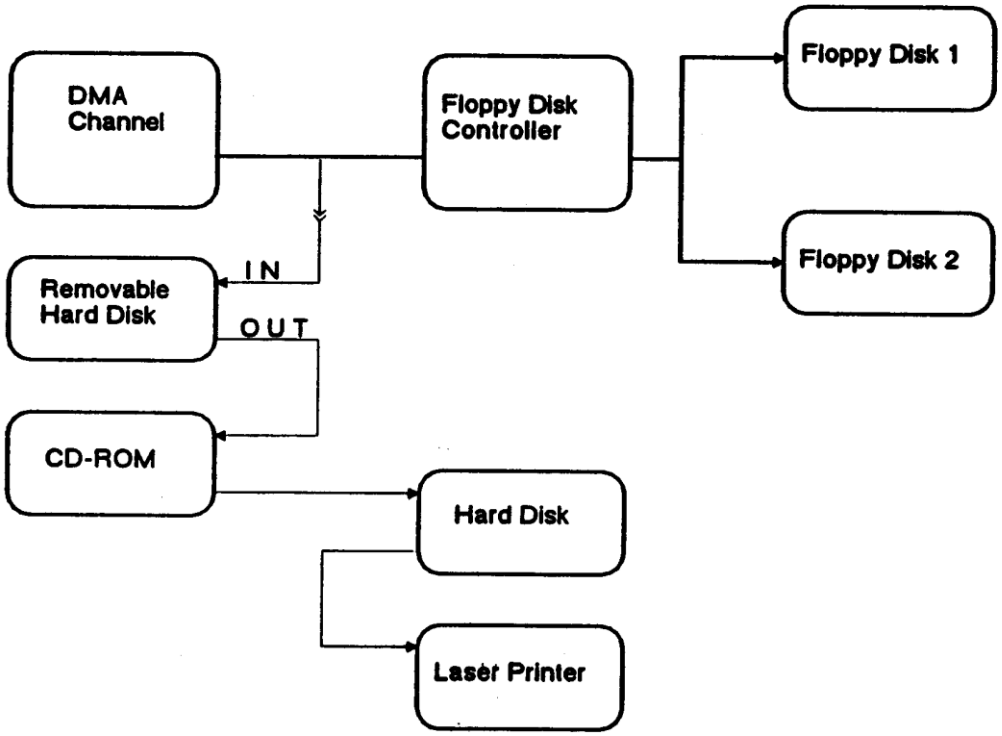
When the program running in your Atari ST computer requires data from a peripheral connected to the DMA channel (directly or through the ACSI bus) or needs to send data to the device, the program must perform the following basic steps:

1. Set a starting Atari ST Memory location for the DMA data to be sent to or received from.
2. Set a DMA count. This count is (at least) the integral number of 512 byte blocks to be sent to or received from Atari ST memory.
3. Set the direction of data flow. This is accomplished by setting the DMA Read/Write bit to 0 for a read from a peripheral device and 1 for a write to a device.
4. Write the command to be performed to the peripheral device and wait for command completion.
5. Select the DMA source (external or internal). DMA takes place here, with the peripheral indicating completion.
6. Check the device status for an error. If no error occurred the data is now correctly placed in the peripheral or Atari ST Memory (depending on whether you were sending it to, or receiving it from, the peripheral).

General Atari ST DMA Connection Block Diagram

Following is a non-limitative list of Atari ST peripheral devices that use the DMA channel including the Atari ST floppy disk controller (which is connected directly to the DMA channel): the Laser printer through its APPC (Atari Page Printer Controller), the Hard Disk through its AHDI (Atari Hard Disk Interface), the CDAR Audio/ROM CD Unit, and the Removable Hard Disk.

More recent peripheral include Satan or UltraSatan Disk interface.



DMA Registers Address Map

The DMA registers are mapped in memory of the Atari ST at the following address:

R/W Byte	$FF860D	DMA Address Counter Low
R/W Byte	$FF860B	DMA Address Counter Medium
R/W Byte	$FF8609	DMA Address Counter High
R/W Word	$FF8606	DMA Mode(W) / Status(R)
R/W Word	$FF8604	FDC Access / Sector Count
	$FF8602	Reserved
	$FF8600	Reserved

DMA registers address map

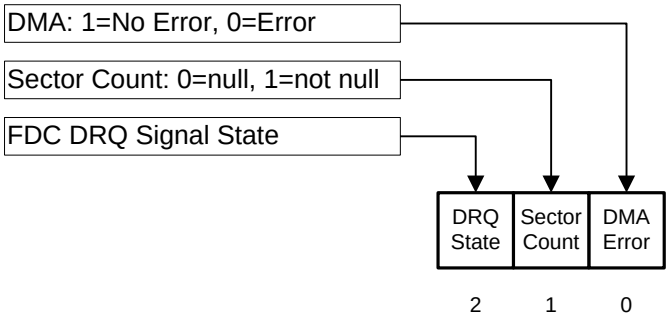
DMA Registers detail

$FF8604 R/W (16 bits) ***FDC Registers Access / DMA Sector Count Register:***

- When bit 4 of the DMA mode register is zero a read or write access to this word causes a read or write cycle on the DMA bus (referred later as a DMA bus cycle). If bit 3 of the mode register is set the Hard Disk is selected (HDCS*) otherwise, the Floppy Disk is selected (FDCS*). The CR/W* signal is set according to the type of the CPU access (R/W) and CA1; CA2 signals are set according to bit 1 & 2 of the mode register. The DMA interface only uses 8 bits when writing and therefore the upper byte is ignored and when reading the 8 upper bits consistently reads 1.
- When bit 4 of the DMA mode register is one the internal sector count register is selected (***write only*** - trying to read this register returns unpredictable value). This register stores the upper limit on the number of 512-bytes blocks that can be transferred in a single DMA operation. This sector count register is decrement by one each time 512 bytes' block has been transferred and when the count reaches zero the DMA will stop to transfer data. **Only the lower 8 bits are used** (value 1 to 255). Therefore, up to 255*512 (130560) bytes can be transferred in one operation.

$FF8606 R (16 bits) ***DMA Status word:***

Bit 0: DMA error status: 1 = no error; 0 = error
Bit 1: Sector count status: 0 if count has reached zero; 1 otherwise.
Bit 2: DQR Status: 1 if the DRQ signal is active; 0 otherwise.
Other bits are reserved and should be ignored.



Reading the DMA status register (Word)

$FF8606 W (16 bits) ***DMA Mode Control register:***

Bit 0: not used, must be set to 0.
Bit 1: Control the CA1 output of the DMA chip during a DMA bus cycle. CA1 is connected to A0 of FDC and CA1 of the HD Interface port.

- ► For the HD interface this signal is used to signal the start of a new command block. The start of a new command block happens when this bit is clear (0) and a CS (Chip Select) occurs. For the rest of the command this bit must be set (1).

Bit 2: Control the CA2 output during a DMA bus cycle. CA2 is connected to A1 of FDC and *is not used by the HD interface*.

Bit 3: Select witch of the HDCS*/FDCS* chip-select outputs is low asserted during the DMA bus cycle;
- ► 1 = the HDCS* chip-select will be asserted,
- ► 0 = the FDCS* chip-select will be asserted.

Bit 4: Sector count / Register select: decides whether the DMA internal sector count register of the DMA or the HDC/HDC external registers are accessed when reading or writing at address $FF8604.
- ► 1 = the DMA internal sector count register is selected (write only). The sector count register sets the upper limit of 512 bytes blocks that can be transferred at one time.
- ► 0 = the HDC-FDC external controller registers are accessed through a read or write DMA bus cycle.

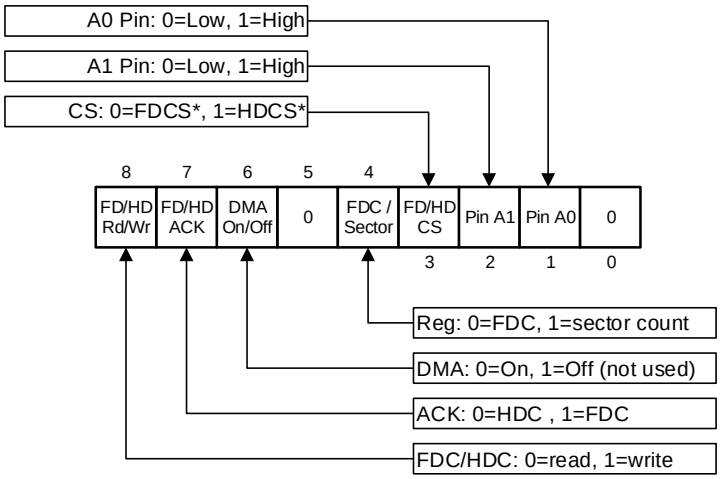
Bit 5: Reserved; must be set to zero.

Bit 6: Supposed to Enable/Disable DMA. When 1 the DMA is disabled; when 0 DMA is enabled. In fact, this bit is not used by the DMA and can take any value but it is a good practice to set it to zero.

Bit 7: FDC/HDC transfers acknowledge;
- ► 1 =the DRQ from the FDC is acknowledged.
- ► 0 = the DRQ from the HDC is acknowledged.

Bit 8: Write/Read DMA transfer direction;
- ► 1 = data are transferred from memory to controller (Write direction);
- ► 0 = data are transferred from controller to memory (Read direction).

When this bit is toggled the DMA is reset. Resetting the controller flushes the internal FIFO and clears the Sector Count Register. This must be done before each DMA operation.



Writing the DMA mode register (Word)

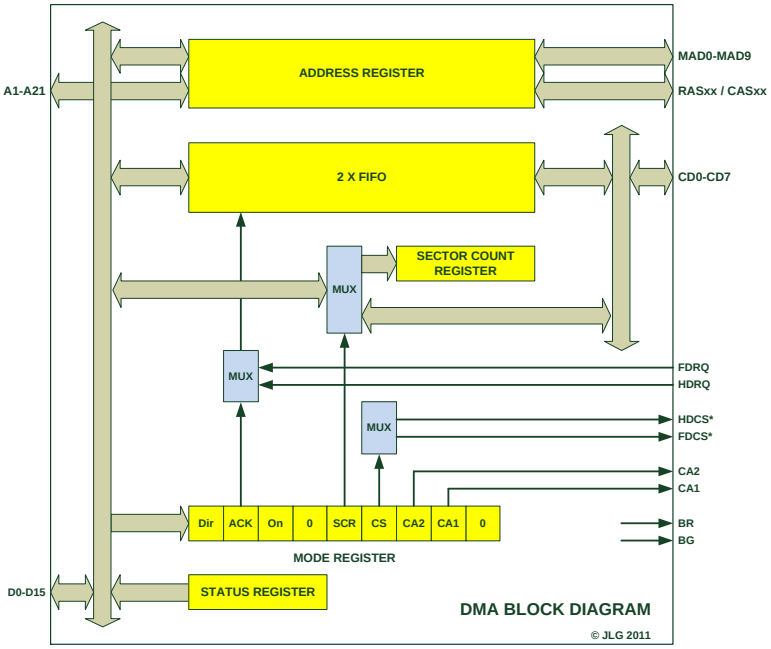
$FF8609	R/W (8 bits)	***DMA Address Counter High byte:*** The High byte of the DMA internal 24 bits address register.
$FF860b	R/W (8 bits)	***DMA Address Counter Middle byte:*** The Middle byte of the DMA internal 24 bits address register.
$FF860d	R/W (8 bits)	***DMA Address Counter Low byte:*** The Lower byte of the DMA internal 24 bits address register.

DMA Block Diagram

Atari did not provide a block diagram of the DMA. Therefore, I have created this extremely simplified block diagram of the DMA chip. The registers/FIFOs are displayed in yellow and the different multiplexers are shown in blue.

Note that the DMA Address register is in fact located in the ***MMU chip*** and some control signals are located in the ***Glue chip*** but they should be considered logically as part of the DMA device.

Obviously this diagram does not show the correct and full control logic of the DMA chip (for example CA1-CA2) but it should help to understand how the different flags of the control register are used.



DMA Mode Control Register Values for the FDC

When programming the DMA mode register for use with the floppy disk controller the following values should be used:

- For accessing the FDC: 0000 000D X000 0$A_1A_0$0
- For accessing the Sector count register: 0000 000D X001 XX X 0

Where D[2] sets the direction of the transfer (1=W, 0=R), A_1 A_0 are the two addresses bit used by the FDC controller, and Xs are don't care (it is usual to set them to 0). In practice writing the following values in the mode register at address $FF8606 will result in accessing the following registers at address $FF8604:

DMA Mode register	**DMA in Read mode**	**DMA in Write Mode**
FDC Register Control (W) / Status (R)	$0080	$0180
FDC Track Register (R/W)	$0082	$0182
FDC Sector Register (R/W)	$0084	$0184
FDC Data Register (R/W)	$0086	$0186
DMA Count Register (W)[3]	$0090	$0190

- For FDC DMA Transfer: 0000 000D 100X 0XX0

DMA Mode Control Register Values for the HDC

When programming the DMA mode register for use with the external hard disk controller the following values should be used:

- For accessing the HDC: 0000 000D X000 1X$A_1$0

[2] The direction bit should not be toggled between accesses; otherwise it will reset the DMA.

[3] DMA count register is Write Only

- For accessing the Sector count register: 0000 000D X001 XXX0
- For HDC DMA Transfer: 0000 000D 000X XXX0

Where D sets the direction transfer mode (1=W, 0=R), A_1 is the CA1 address bit used by HDC controller, and Xs are don't care (it is usual to set them to 0).

DMA Programming Tips and idiosyncrasies:

- The DMA Address Counter register must be loaded (written) in a Low, Mid, High order. Be sure to write the Address Counter register **before** writing the Sector Count register as this will trigger the DMA immediately.
- The DMA Address Counter register must be read in a High, Mid, Low order.
- The DMA address register is 24 bits long but in fact only 22 bits are used on STF/STE. Therefore, writing $41 at address $FF8609 is equivalent to writing $01.
- There are two eight-word FIFOs in the DMA chip which serves as read/write buffers.
  - ► In read mode, when one of the FIFO is full (i.e. when 16 bytes has been transferred from the FDC or HDC) the DMA chip performs a bus request to the 68000 and in return the processor will grant the control of the bus to the DMA, the transfer to the memory is then done with 8 cycles then the bus is released to the system. As the processor takes time to grant the bus and as transferring data from FIFO to memory also takes time, the other FIFO is used for continuing the data transfer with the FD/HD controllers. The FIFOs are ***not flushed automatically*** at the end of a transfer, and therefore it is only possible to transfer data in ***multiples of 16 bytes***. Be aware of this behavior, for command that does not transfer data in multiple of 16 bytes like the ***read address*** command which only transfer 6 bytes.
  - ► In write mode, when one of the FIFO is empty (i.e. when 16 bytes has been transferred to the FDC or HDC) the DMA chip performs a bus request to the 68000 and in return the processor will grant the control of the bus to the DMA, the transfer from the memory is then done with 8 cycles then the bus is released to the system. As the processor takes time to grant the bus and as transferring data from memory to FIFO also takes time, the other FIFO is used for continuing the data transfer with the FD/HD controllers. It is important to know that after the DMA has been set to ***write mode***, the transfer of data to the two FIFOs is triggered by writing a value to the sector count register. In other word **immediately** after writing the DMA Sector Count Register the first 32 bytes from memory are written into the internal FIFOs in preparation for transfer to the FDC on demand.
- Toggling the Read/Write transfer direction bit (bit 8) of the mode register clears the DMA controller status register, flushes the two internal FIFOs, and clears the Sector Count Register. Therefore, when accessing DMA/FDC registers, be careful not to toggle the transfer direction bit (the bit-8) otherwise this would reset the DMA. You **should** reset the DMA controller before each DMA operation[4].
- The DMA chip has no interrupt capability. Therefore, the end-of-transfer interrupts are generated by the controllers (the FDC & HDC interrupt outputs are logically OR'ed). These interrupts are connected to the General Purpose I/O Port, bit 5 and are masked and vectored by the 68901 MFP chip, on interrupt level 7. If you prefer to poll the status of the interrupt request line you can test the bit 5 of the MFP GPIP data register (this is what is done by the current TOS software). For that matter just read a byte at address $FFFA01 and mask it with $20; if the result is zero there is no interrupt.
- Turn off the floppy VBL check routine **_flopvbl** while using the FDC/DMA by setting the floppy lock variable (**flock** at $43e) to $FF. This prevents the VBL routine to screw up the transfer by accessing the DMA chip registers periodically. When the transfer is finished you have to reset the **flock** variable to 0 this will cause the **_flopvbl** routine to automatically deselect the drive for you once the FDC has shut off the motor.
- If the DMA status word is polled during a DMA operation *the transfer might be disrupted*. Therefore, polls the Floppy Disk Controller interrupt using the MFP General Purpose I/O register to detect the completion of a WD1772 FDC command. Do **not** poll the FDC Busy or DMA Sector Count zero status bits of the DMA status register.
- Make sure you select the Sector Count Register mode (setting DMA mode register bit 4 = 1) before setting the register count. Make sure you select the Controller Access mode (setting the DMA mode register bit 4 = 0) before setting or reading any of the FDC registers.
- The DMA count register set the upper limit of 512 bytes blocks that can be transferred in a single DMA operation. Therefore, up to 127.5 Kbytes (255 * 512) can be transferred in a single DMA operation. The

[4] The only exception is when you want to repeat a read operation to get more than 16 bytes of data to flush the content of the FIFO.

lower limit for the sector count is 1 for any transfer of 512 bytes or less. Setting it to zero result in a DMA error.

- The Atari Hardware documentation indicates that it is necessary to select the DMA status count register before testing the DMA status however ***this is not necessary***. Note that this is always done in the Atari source code I have seen and it does not hurt…
- The sector count register is write only. Reading this register return unpredictable values.
- Writing the sector count register triggers the DMA operation. For that reason, you must follow this sequence: first load the address counter register, then reset the DMA, then set the transfer mode with the control register, and finally trigger the DMA operation by writing to the sector counter register.
- When accessing DMA/FDC registers be careful not to toggle the transfer direction bit (the bit-8) otherwise this resets the DMA. For example, if you are in write mode use the value $180 to access the FDC status register instead of the value $080 in read mode.
- Bit 6 of the DMA Mode Control register is supposed to enable/disable the DMA. However, on STF/STe this bit is ignored (i.e. writing $90 is equivalent to writing $D0).
- The Atari documentation mentions that it is necessary to write the DMA control register immediately after you write the DMA data register. Otherwise it is possible to get a double strobe from the DMA chip. In practice not following this recommendation seems to work fine. This might be necessary when using the so-called Atari bad DMA chip? If you want to be safe you may want to follow this rule.

PSG Programming

This section gives information on programming the PSG (YM2149) chip in the context of accessing the floppy drives through the FDC controller. The signals **SELECT0***, **SELECT1***, and **SIDE0*** of the FD interface are connected to bits 0-2 of the PSG I/O port A.

PSG Registers Address Map

The PSG registers are mapped in memory of the Atari ST at the following address:

R/W Byte	$FF8802	Write Data
R/W Byte	$FF8800	Register Select / Read data
	Reg 14	Port A

PSG Registers detail

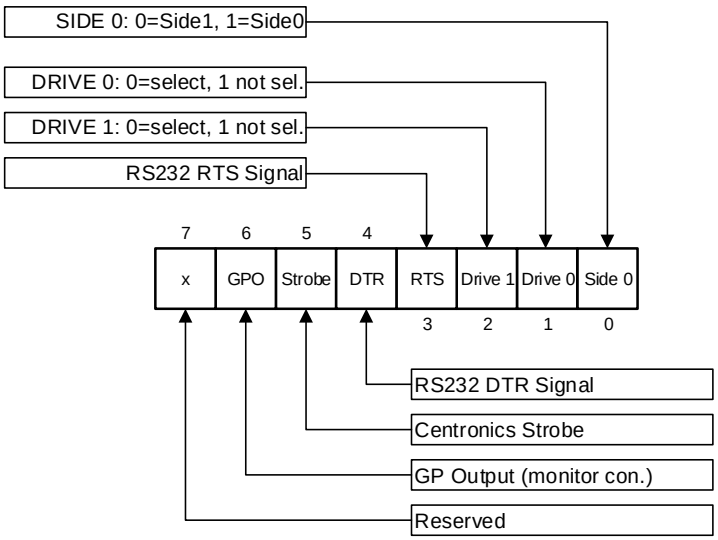
$FF8800 W (8 bits) ***PSG Register Select:***
The YM2149 has 16 internal registers. The number of the register you want to access needs to be loaded in the PSG select register. The data read and write will access this register until a new value is written to the select register. Note that only the bottom 4 bits are used to select one of the 16 internal registers. For accessing the I/O port A, the **value 14** must be loaded.

$FF8800 R (8 bits) ***PSG Read Data:***
Read a byte from the selected register

$FF8802 W (8 bits) ***PSG Write Data***:
Write a byte to the selected register.

The 3 lower bits of the register 14 (I/O Port A) are used by the floppy interface:

Bit 0: Side selection: 1 selects side 0, and 0 selects side 1 of a double sided floppy diskette.
Bit 1: Drive 0 selection. When 0 drive 0 is selected.
Bit 2: Drive 1 selection. When 0 drive 1 is selected.



PSG Port A bits

PSG Programming Tips:

- During access to the PSG data register (read or write), **all the 68000 interrupts** must be disabled. Otherwise the floppy interrupt routine will deselect the drive again. If you are programming in C

language it is practical to use the `Giaccess()` routine (takes care of disabling interrupt) instead of directly accessing the PSG register.

- Only **one drive** must be selected!
- Never leaves a drive selected when not used anymore. This might be harmful to your floppies.
- You should be careful before deselecting a drive. The FDC will automatically turn off the motor after the 10 index pulse (10 * 200ms) if no command has been received during this period. However the drive **needs to be selected** in order to the index pulse to be conveyed to the FDC. This implies that you **must** wait for the motor to stop before deselecting the drive.
- As the I/O Port A is used for other things make sure you do not change the state of bits 3-7. For that matter you have to read the current state of the Port A and only modify the three lower bits.

MFP 68901 Programming

This section gives information on programming the MFP (68901) chip in the context of accessing the floppy drives through the FDC controller.

MFP Registers Address Map

The MFP registers are mapped in memory of the Atari ST at the following address

R/W Byte	$FFFA01	General Purpose I/O Int Port
R/W Byte	$FFFA03	Active Edge Register
R/W Byte	$FFFA05	Data Direction
R/W Byte	$FFFA07	Interrupt Enable A
R/W Byte	$FFFA09	Interrupt Enable B
R/W Byte	$FFFA0B	Interrupt Pending A
R/W Byte	$FFFA0D	Interrupt Pending B
R/W Byte	$FFFA0F	Interrupt In Service A
R/W Byte	$FFFA11	Interrupt In Service B
R/W Byte	$FFFA13	Interrupt Mask A
R/W Byte	$FFFA15	Interrupt Mask B
R/W Byte	$FFFA17	Vector Register
R/W Byte	$FFFA19	Timer A Control
R/W Byte	$FFFA1B	Timer B Control
R/W Byte	$FFFA1D	Timer C&D Control
R/W Byte	$FFFA1F	Timer A Data
R/W Byte	$FFFA21	Timer B Data
R/W Byte	$FFFA23	Timer C Data
R/W Byte	$FFFA25	Timer D Data
R/W Byte	$FFFA27	Sync Character
R/W Byte	$FFFA29	USART Control
R/W Byte	$FFFA2B	Receiver Status
R/W Byte	$FFFA2D	Transmitter Status
R/W Byte	$FFFA2F	USART Data

The MFP is initialized by the TOS. The main usage of the MFP for floppy drive programming is to detect the INTRQ from the FDC. This signal is input on **bit 5** of the GPIO register.

By enabling the bit 7 of the Interrupt Enable B register it is possible to generate an interrupt (vector $11C). But in most case it is easier just to poll the bit 5 of the GPIO to find out if an interrupt has been generated by the FDC (bit 5 = 0)

Another possible usage of the MFP in the context of floppy programming is to use one of the 4 timers to measure precise timing. This can be particularly useful to measure the exact time it takes to read a sector. This information is useful to check for specific protection mechanism (i.e. Copylock, Macrodos, …). I recommend using the timer A for that matter[5]. The timer must be programmed in delay mode with an appropriate pre-scale (dividing by 10 is suggested) and the running value can be polled. It is also possible to program the MFP so that an overflow of the timer generates an interrupt on vector $134 (bit 5 of the Interrupt Enable A register).

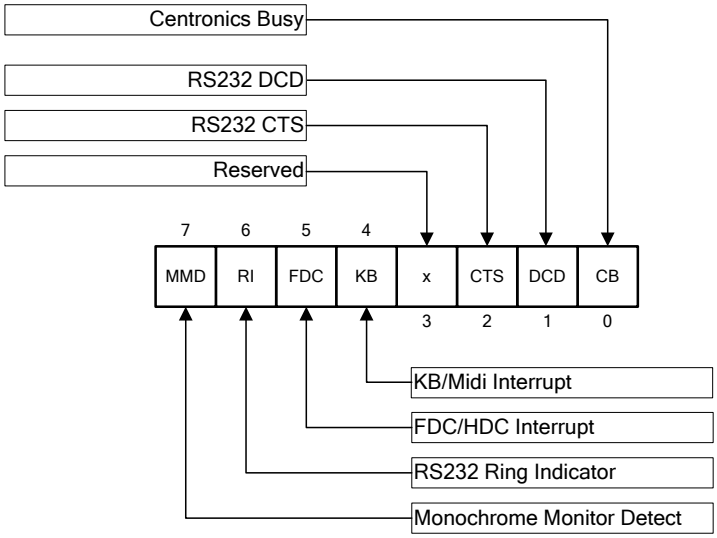
MFP Registers detail

Here we will only look at the registers useful for the FDC programming.

[5] Timer A is used in STE for the sound, but can be momentarily used. Remember to set the Timer A control register back to the state it was before usage (save and restore).

$FFFA01 R/W (8 bits) ***GPIO General Purpose I/O Port***

This is the data register for the 8 bit ports, where the data from the port bits are send or read.



MFP GPIO bits

$FFFA19 R/W (8 bits) ***Timer A Control Register***

Timer A is fed with the Centronics Busy signal. Timer A is normally not used and can therefore be used to measure time (see Measurement of FDC bytes width). The last 5 bits of the control register are used to determine the operating mode of the timer. The bit 4 reset the timer and should be kept at zero and the three low bits are programmed as follow:

Bits 3 - 0	Function
0000	Timer stop
0001	Delay Mode divide by 4
0010	Delay Mode divide by 10
0011	Delay Mode divide by 16
0100	Delay Mode divide by 50
0101	Delay Mode divide by 64
0110	Delay Mode divide by 100
0111	Delay Mode divide by 200
1000	Event count mode

$FFFA1F R/W (8 bits) ***Timer A Data Register***

Read/Write the timer counter.

FDC WD1772 Programming

This section gives information on low level programming of the FDC (WD1772).

Accessing FDC Registers

As mentioned the FDC is not directly mapped in the 68000 address space, but is accessed through the DMA chip (see DMA Registers Address Map). Like for the PSG this is therefore a two steps operation: first the desired register is selected through the DMA, then the reads or writes transfer takes place. Reading or writing to the selected register is done at address $FF8604.

Selecting one of the 4 FDC registers is done using the bit 1 and 2 of the DMA mode register (see DMA Register detail for more information). The bit 4 of the DMA mode register must be set to 0, and the bit 8 (DMA transfer direction) **must not** be toggled (otherwise the DMA is reset). This leads to the following values of the DMA mode register for accessing the FDC registers:

DMA Mode register	DMA in Read mode	DMA in Write Mode
FDC Control / Status Register	$080	$180
FDC Track Register	$082	$182
FDC Sector Register	$084	$184
FDC Data Register	$086	$186

FDC Registers detail

Data Shift Register - This 8-bit register assembles serial data from the Read Data input (RD) during Read operations and transfers serial data to the Write Data output during Write operations.

Data Register - This 8-bit register is used as a holding register during Disk Read and Write operations. In disk Read operations, the assembled data byte is transferred in parallel to the Data Register from the Data Shift Register. In Disk Write operations, information is transferred in parallel from the Data Register to the Data Shift Register.

When executing the Seek Command, the Data Register holds the address of the desired Track position. This register is loaded from the Data bus and gated onto the Data bus under processor control.

Track Register - This 8-bit register holds the track number of the current Read/Write head position. It is incremented by one every time the head is stepped in and decremented by one when the head is stepped out (towards track 00). The content of the register is compared with the recorded track number in the ID field during disk Read, Write, and Verify operations. The Track Register can be loaded from or transferred to the Data bus. This Register is not loaded when the device is busy.

Sector Register (SR) - This 8-bit register holds the address of the desired sector position. The contents of the register are compared with the recorded sector number in the ID field during disk Read or Write operations. The Sector Register contents can be loaded from or transferred to the Data bus. This register is not loaded when the device is busy.

Command Register (CR) - This 8-bit register holds the command presently being executed. This register is not loaded when the device is busy unless the new command is a force interrupt. The Command Register is loaded from the Data bus, but not read onto the Data bus.

Status Register (STR) - This 8-bit register holds device Status information. The meaning of the Status bits is a function of the type of command previously executed. This register is read onto the Data bus, but not loaded from the Data bus.

FDC General Disk Read Operations

Sector lengths of 1281 256, 512 or 1024 are obtainable in either FM or MFM formats. For FM, DDEN* is placed to logical 1. For MFM formats, DDEN* is placed to a logical 0. Sector lengths are determined at format time by the fourth byte in the ID field.

SECTOR LENGTH TABLE	
SECTOR LENGTH FIELD (HEX)	**NUMBER OF BYTES IN SECTOR (DEC)**
00	128
01	256
02	512
03	1024

The number of sectors per track for the WD1772 is from 1 to 240. The number of tracks for the WD1772 is 0 to 240.

FDC General Disk Write Operation

When writing on the diskette the WG output is activated, allowing current to flow into the Read/Write head. As a precaution to erroneous writing the first data byte is loaded into the Data Register in response to a Data Request from the device before the WG is activated.

Writing is inhibited when the WPRT* input is asserted, in which case any Write Command is immediately terminated, an interrupt is generated and the Write Protect Status bit is set.

For Write operations, the WD1772 provides WG to enable a Write condition, and WD which consists of a series of active high pulses. These pulses contain both Clock and Data information in FM and MFM. WD provides the unique missing clock patterns for recording Address Marks.

On the WD1772, the Precomp Enable bit in Write Commands allows automatic Write pre-compensation to take place. The outgoing Write Data stream is delayed or advanced from nominal by 125 nsec according to the following table:

PATTERN				**MFM**	**FM**
X	1	1	0	Early	N/A
X	0	1	I	Late	N/A
0	0	0	1	Early	N/A
1	0	0	0	Late	N/A
previous bit sent		current bit sending	next bit to be sent		

Pre-compensation is typically enabled on the inner most tracks where bit shifts usually occur and bit density is at its maximum. READY is true for read/write operations (all Type II and III Command executions).

FDC Command Summary

The WD1772 accepts 11 commands[6]. Command words are only loaded in the Command Register when the Busy Status bit is off (Status bit 0). The one exception is the Force Interrupt Command. Whenever a command is being executed, the Busy Status bit is set. When a command is completed, an interrupt is generated and the Busy Status bit is reset. The Status Register indicates whether the completed command encountered an error .or was fault free. Commands are divided into four types and are summarized in the following sections.

		BITS							
TYPE	**COMMAND**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
I	Restore	0	0	0	0	h	v	r1	r0
I	Seek	0	0	0	1	h	v	r1	r0
II	Read Sector	1	0	0	m	h	e	0	0
II	Write Sector	1	0	1	m	h	e	p	a0
III	Read Address	1	1	0	0	h	e	0	0
III	Read Track	1	1	0	1	h	e	0	0
III	Write Track	1	1	1	1	h	e	p	0
IV	Force Interrupt	1	1	0	1	i3	i2	i1	i0

[6] Only 8 of the 11 commands are presented here. The step / Step-in, and Step-out commands are not presented.

Flag Summary[7]

h = Motor On Flag (Bit 3)		**e = 15ms Settling Delay (Bit 2)**	
0	Enable Spin-up Sequence	0	No Delay
1	Disable Spin-up Sequence	1	Add 15ms Delay
m = Multiple Sector Flag (Bit 4)		**p = Write Pre-compensation (Bit 1)**	
0	Single Sector	0	Unable Write Pre-comp
1	Multiple Sector	1	Disable Write Pre-comp
v = Verify Flag (Bit 2)		**a0 = Data Address Mark (Bit 0)**	
0	No Verify	0	Write Normal Data Mark
1	Verify on Destination Track	1	Write Deleted Data Mark
r1,r0 = Stepping Rate (Bits 1,0)		**i3,i2,i1,i0 Interrupt condition (Bits 3-0)**	
0,0	6 ms	1,0,0,0	Immediate Interrupt
0,1	12 ms	0,1,0,0	Interrupt on Index Pulse
1,0	2 ms	0,0,0,0	Terminate without interrupt
1,1	3 ms		

In the Atari the MO output is directly connected to the drive and it is therefore mandatory to always enable the spin-up sequence (**h = 0**). The settling delay option should not be used (**e = 0**), and the write pre-compensation should always be used (**p = 0**). The stepping rate should normally be set to 3 ms (**r0, r1 = 1, 1**) but it is possible to set it to 2 ms (r0, r1 = 1, 0) however this gives less reliable results.

[7] The u flag is not presented as it is only used by the step commands not presented here.

FDC Type I Commands

The Type I Commands Include the Restore and Seek Commands. Each of the Type I Commands contains a rate field (r0, r1), which determines the stepping motor rate. As already mentioned in the Atari the stepping rate should normally be set to 3 ms (**r0, r1 = 1, 1**). A 4 µsec (MFM) or 8 µsec (FM) pulse is provided as an output to the drive. For every step pulse issued, the drive moves one track location in a direction determined by the direction output. The chip steps the drive in the same direction it last stepped unless the command changes the direction

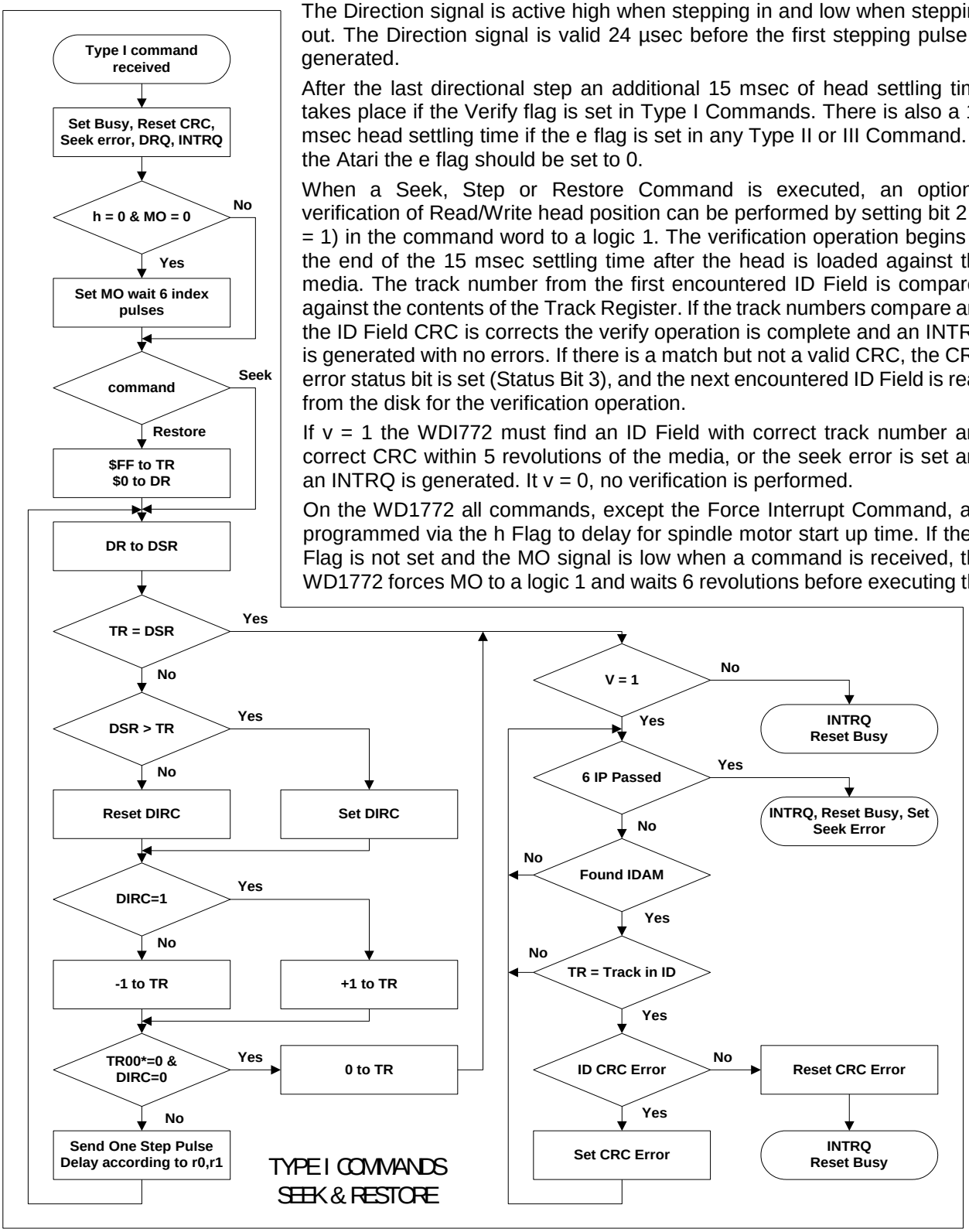
The Direction signal is active high when stepping in and low when stepping out. The Direction signal is valid 24 µsec before the first stepping pulse is generated.

After the last directional step an additional 15 msec of head settling time takes place if the Verify flag is set in Type I Commands. There is also a 15 msec head settling time if the e flag is set in any Type II or III Command. In the Atari the e flag should be set to 0.

When a Seek, Step or Restore Command is executed, an optional verification of Read/Write head position can be performed by setting bit 2 (v = 1) in the command word to a logic 1. The verification operation begins at the end of the 15 msec settling time after the head is loaded against the media. The track number from the first encountered ID Field is compared against the contents of the Track Register. If the track numbers compare and the ID Field CRC is corrects the verify operation is complete and an INTRQ is generated with no errors. If there is a match but not a valid CRC, the CRC error status bit is set (Status Bit 3), and the next encountered ID Field is read from the disk for the verification operation.

If v = 1 the WDI772 must find an ID Field with correct track number and correct CRC within 5 revolutions of the media, or the seek error is set and an INTRQ is generated. It v = 0, no verification is performed.

On the WD1772 all commands, except the Force Interrupt Command, are programmed via the h Flag to delay for spindle motor start up time. If the h Flag is not set and the MO signal is low when a command is received, the WD1772 forces MO to a logic 1 and waits 6 revolutions before executing the



command. At 300 RPM, this guarantees a one second spindle start up time. If after finishing the command, the device remains idle for 9 revolutions, the MO signal goes back to a logic 0. If a command is issued while MO is high, the command executes immediately, defeating the 6 revolution start up. This feature allows consecutive Read or Write commands without waiting for motor start up each time; the WD1772 assumes the spindle motor is up to speed. In the Atari the h flag must always be set to 0.

Restore (Seek Track 0)

Upon receipt of this command, the Track 00 (TR00*) input is sampled. If TR00* is active low indicating the Read/Write head is positioned over track 0, the Track Register is loaded with zeroes and an interrupt is generated. If TR00* is not active low, stepping pulses at a rate specified by the r1, r0 field are issued until the TR00* input is activated.

At this time, the Track Register is loaded with zeroes and an interrupt is generated. If the TR00* input does not go active low after 255 stepping pulses, the WD1772 terminates operation, interrupts, and sets the Seek Error status bit, providing the v flag is set.

A verification operation also lakes place if the v flag is set. The h bit allows the Motor On option at the start of a command.

Seek

This command assumes that the Track Register contains the track number of the current position of the Read/Write head and the Data Register contains the desired track number. The WD1772 updates the Track Register and issues stepping pulses in the appropriate direction until the contents of the Track Register are equal to the contents of the Data Register (the desired track location). A verification operation takes place if the v flag is on. The h bit allows the Motor On option at the start of the command. An interrupt is generated at the completion of the command. Note: When using multiple drives, the Track Register must be updated for the drive selected before seeks are issued.

FDC Type II Commands

The Type II Commands are the Read Sector and Write Sector commands. Prior to loading the Type II command into the Command Register, the computer loads the Sector Register with the desired sector number. Upon receipt of the Type II command, the Busy Status bit is set. If the e flag = 1 the command executes after a 15 msec delay.

When an ID field is located on the disk, the WD1772 compares the Track Number on the ID field with the Track Register. If there is not a match, the next encountered ID field is read and a comparison is again made. If there is a match, the Sector Number of the ID field is compared with the Sector Register. If there is no Sector match, the next encountered ID field is read off the disk and comparisons again made. If the ID field CRC is correct, the data field is located and is either written into, or read from, depending upon the command. The WD1772 finds an ID field with a Track number, Sector number, and CRC within four revolutions of the disk, or, the Record Not Found Status bit is set (Status Bit 4) and the command is terminated with an INTRQ.

Each of the Type II Commands contains an m flag which determines II multiple records (sectors) are read or written, depending upon the command. If m = 0, a single sector is read or written and an Interrupt is generated at the completion of the command. If m = 1, multiple records are read or written with the Sector Register Internally updated so that an address verification occurs on the next record. The WD1772 continues to read or write multiple records and updates the Sector Register in numerical ascending sequence until the Sector Register exceeds the number of sectors on the track or until the Force interrupt Command is loaded into the Command Register, which terminates the command and generates an interrupt.

For example: Witte WD1772 is instructed to read sector 10 and there are only 9 on the track, the Sector Register exceeds the number available. The WD1772 searches for 5 disk revolutions, interrupts out, resets Busy, and sets the Record Not Found Status Bit.

Read Sector

Upon receipt of the Read Sector Command, the Busy Status Bit is set, then when an ID field is encountered that has the correct track number, correct sector number, and correct CRC, the data field is presented to the computer. The Data Address Mark of the data field is found with 30 bytes in single density and 43 bytes in double density of the last ID field CRC byte. If not, the ID field is searched for and verified again followed by the Data Address Mark search. If, after five revolutions the DAM is not found, the Record Not Found Status Bit is set and the operation is terminated. When the first character or byte of the data field is shifted through the DSR, it is transferred to the DR, and DRQ is generated. When the next byte is accumulated in the DSR, it

is transferred to the DR and another DRQ is generated. If the computer has not read the previous contents or the DR before a new character is transferred that character is lost and the Lost Data Status Bit is set. This sequence continues until the complete data field is inputted to the computer. If there is a CRC error at the end of the data field, the CRC Error Status bit is set, and the command is terminated (even if it is a multiple record command).

At the end of the Read operation, the type of Data Address Mark encountered in the data field is recorded In the Status Register (Bit 5) as shown:

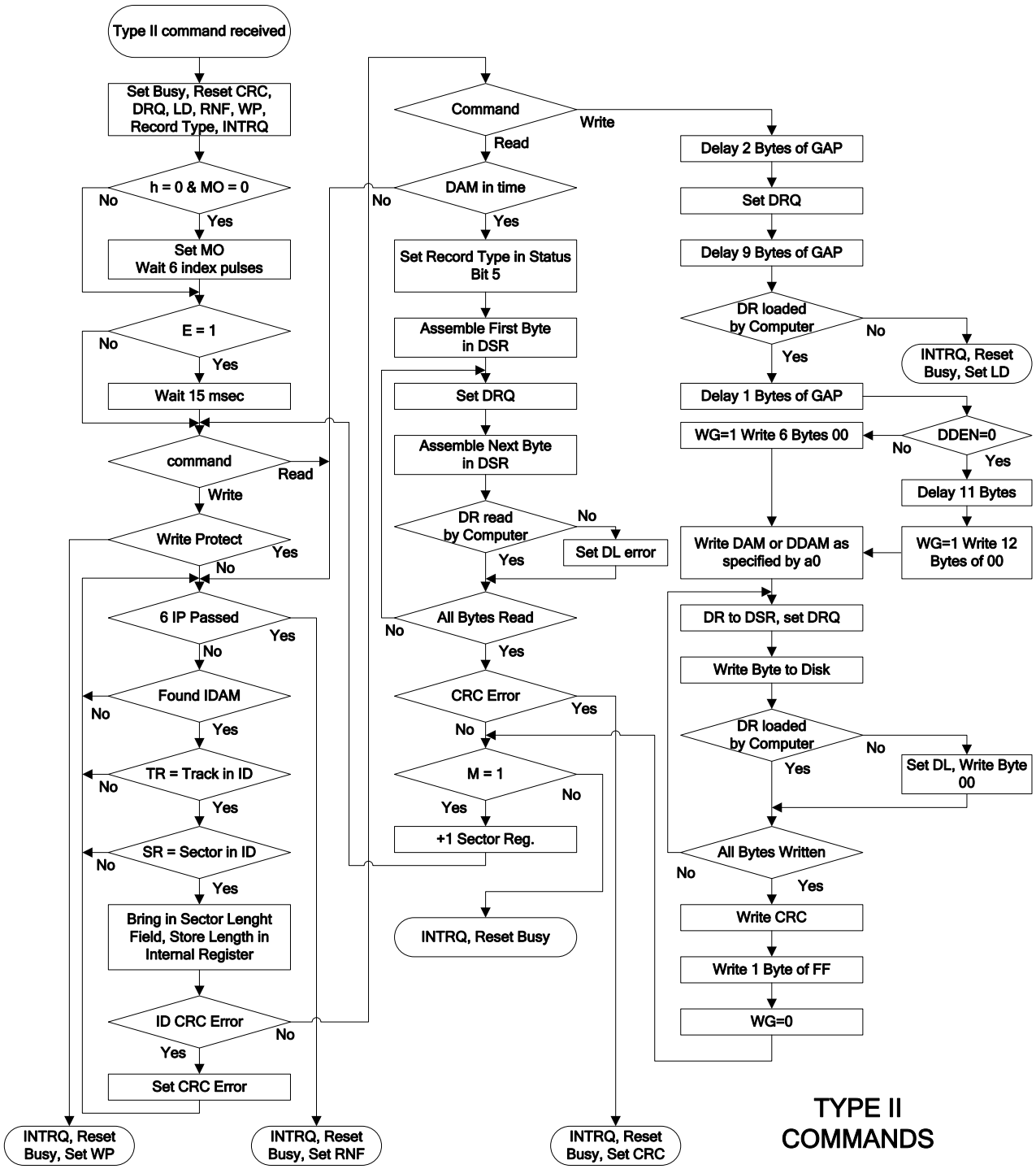
Status Bit 5	
1	Deleted Data Mark
0	Data Mark

Write Sector

Upon receipt of the Write Sector-Command, the Busy Status Bit is set. When an ID field is encountered that has the correct track number, correct sector number, and correct CRC, a DRQ is generated. The WD1772 counts off 11 bytes in single density and 22 bytes in double density from the CRC field and the WG output is made active. If the DRQ is serviced (i.e., the DR is loaded by the computer). If DRQ is not serviced, the command is terminated and the Lost Data Status Bit is set. If the DRQ is serviced, the WG is made active and six bytes of zeroes in single density and 12 bytes in double density are written on the disk. The Data Address Mark is then written on the disk as determined by the a0 field of the command as shown:

a0 Bit 0	**Data Address Mark**
1	Deleted Data Mark
0	Data Mark

The WD1772 writes the data field and generates DRQ's to the computer. If the DRQ is not serviced in time for continuous writing the Lost Data Status Bit is set and a byte of zeroes is written on the disk. The command is not terminated. After the last data byte is written on the disk, the two-byte CRC is computed internally and written on the disk followed by one byte of logic ones in FM or in MFM. The WG output is then deactivated. INTRQ sets 24 µsec (MFM) after the last CRC byte is written. For partial sector writing, the proper method is to write data and fill the balance with zeroes.


Type II command received
Set Busy, Reset CRC, DRQ, LD, RNF, WP, Record Type, INTRQ
h = 0 & MO = 0
No
Yes
Set MO
Wait 6 index pulses
E = 1
No
Yes
Wait 15 msec
command
Read
Write
Write Protect
Yes
No
6 IP Passed
Yes
No
Found IDAM
No
Yes
TR = Track in ID
No
Yes
SR = Sector in ID
No
Yes
Bring in Sector Lenght Field, Store Length in Internal Register
ID CRC Error
No
Yes
Set CRC Error
INTRQ, Reset Busy, Set WP
INTRQ, Reset Busy, Set RNF
Command
Write
Read
DAM in time
No
Yes
Set Record Type in Status Bit 5
Assemble First Byte in DSR
Set DRQ
Assemble Next Byte in DSR
DR read by Computer
No
Yes
Set DL error
All Bytes Read
No
Yes
CRC Error
Yes
No
M = 1
No
Yes
+1 Sector Reg.
INTRQ, Reset Busy
INTRQ, Reset Busy, Set CRC
Delay 2 Bytes of GAP
Set DRQ
Delay 9 Bytes of GAP
DR loaded by Computer
No
Yes
INTRQ, Reset Busy, Set LD
Delay 1 Bytes of GAP
WG=1 Write 6 Bytes 00
DDEN=0
No
Yes
Delay 11 Bytes
WG=1 Write 12 Bytes of 00
Write DAM or DDAM as specified by a0
DR to DSR, set DRQ
Write Byte to Disk
DR loaded by Computer
No
Yes
Set DL, Write Byte 00
All Bytes Written
No
Yes
Write CRC
Write 1 Byte of FF
WG=0
TYPE II COMMANDS

FDC Type III Commands

Read Address

Upon receipt of the Read Address Command, the Busy Status Bit is set. The next encountered ID field is then read in from the disk, and six data bytes of the ID field are assembled and transferred to the DR, and a DRQ is generated for each byte. The six bytes of the ID field are shown:

TRACK ADDR	SIDE NUMBER	SECTOR ADDR	SECTOR LENGTH	CRC 1	CRC 2
1	2	3	4	5	6

Although the CRC characters are transferred to the computer, the WD1772 checks for validity and the CRC error status bit is set if there is a CRC error. The Track Address of the ID field is written into the sector register so that a comparison can be made by the user. At the end of the operation an interrupt is generated and the Busy Status is reset. Remember that in the Atari the bytes are read through the DMA FIFOs and therefore only multiple of 16 bytes are written to memory. It is therefore mandatory to repeat the Read Address command several times to pass the FIFO.

Read Track

Upon receipt of the Read Track Command, the head is loaded and the Busy Status bit is set. Reading starts with the leading edge of the first encountered index pulse and continues until the next index pulse. All Gap, Header, and data bytes are assembled and transferred to the data register and DRQ's are generated for each byte. The accumulation of bytes is synchronized to each address mark encountered. An interrupt is generated at the completion of the command.

This command has several characteristics which make it suitable for diagnostic purposes. They are: no CRC checking is performed; gap information is included in the data stream; and the Address Mark Detector **is on for the duration of the command**. Because the AM detector is always on, write-splices may cause the chip to look for an AM inside address or data blocks.

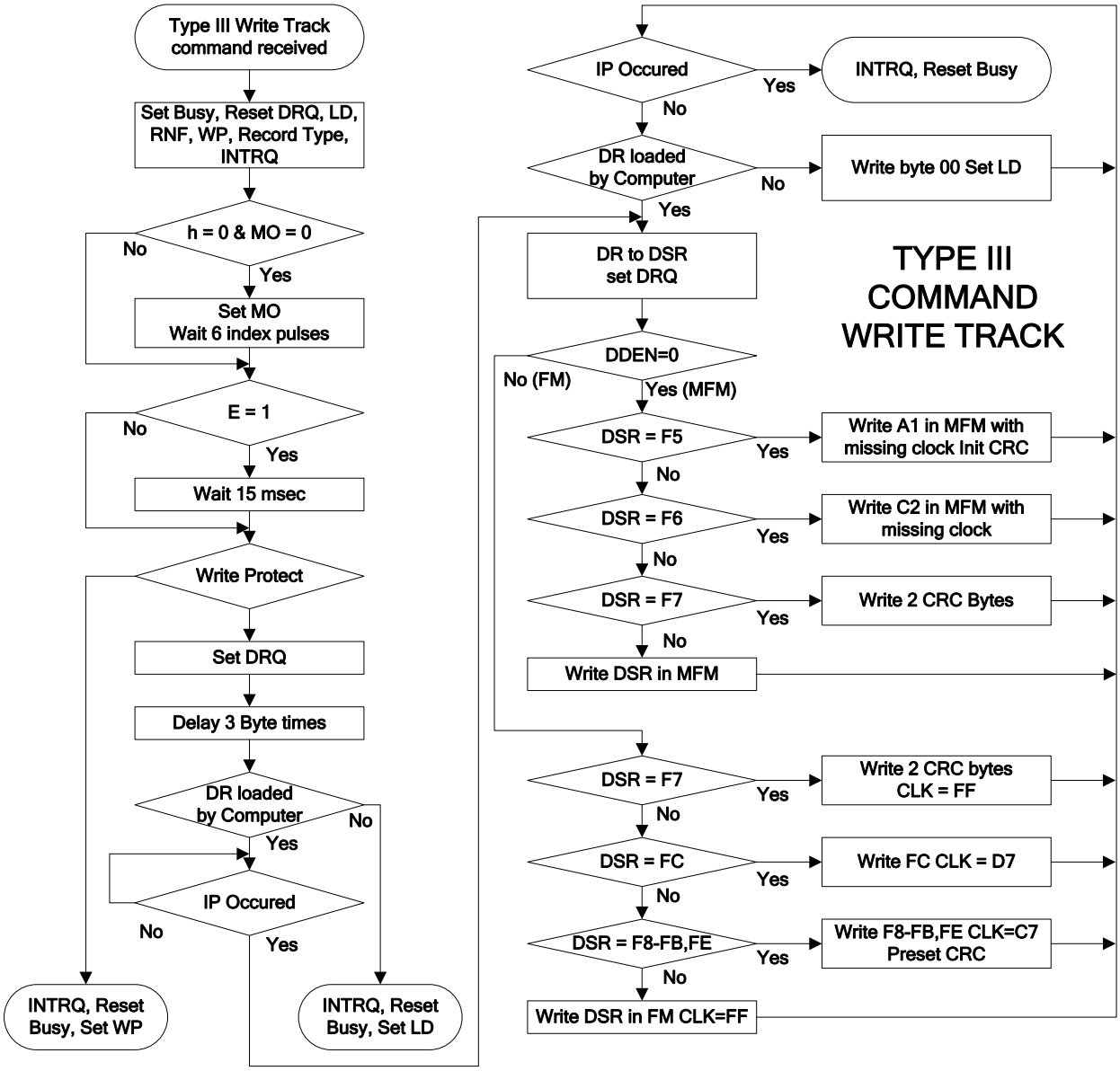
The IDAM, ID field, ID CRC bytes, DAM, Data, and Data CRC Bytes for each sector are correct. The Gap Bytes may be read incorrectly during write-splice time because of synchronization.

Write Track Formatting the Disk

Data and gap information are provided at the computer interface. Formatting the disk is accomplished by positioning the R/W head over the desired track number and issuing the Write Track Command. Upon receipt of the Write Track Command, the Busy Status Bit is set. Writing starts with the leading edge of the first encountered Index Pulse and continues until the next index Pulse, at which time the interrupt is activated. The Data Request is activated immediately upon receiving the command, but writing does not start until after the first byte is loaded into the Data Register. If the DR is not loaded within three byte times, the operation is terminated making the device Not Busy, the Lost Data Status Bit is set, and the interrupt is activated. If a byte is not present in the DR when needed, a byte of zeroes is substituted.

This sequence continues from one Index Pulse to the next. Normally whatever data pattern appears in the Data Register is written on the disk with a normal clock pattern. However, if the WD1772 detects a data pattern of F5 through FE in the Data Register, this is interpreted as Data Address Marks with missing clocks or CRC generation.

The CRC generator is initialized when any data byte from F8 to FE is transferred from the DR to the DSR in FM or by receipt of F5 in MFM. An F7 pattern generates two CRC characters in FM or MFM. As a consequence, the patterns F5 through FE do not appear in the gaps, data field, or ID fields. Also, CRC's are generated by an F7 pattern.



Note that, in MFM, for the marks characters (between $F8 and $FF) the least significant bit is always ignored and therefore: $F8=$F9, …, $FE = $FF.

Usually disks are formatted using IBM 3740 or System 34 formats with sector lengths of 128, 256, 512, or 1024 bytes.

Data Pattern In DR (HEX)	In FM (DDEN* = 1)	In MFM (DDEN* = 0)
00 thru F4	Write 00 thru F4 with CLK = FF	Write 00 thru F4, in MFM
F5	Not Allowed	Write Al[8] in MFM, Preset CRC
F6	Not Allowed	Write C2[9] in MFM
F7	Generate 2 CRC bytes	Generate 2 CRC bytes
F8 thru FB	Write F8 thru FB, CLK = C7 Preset CRC	Write F8 thru FB, in MFM
FC	Write FC with CLK = D7	Write FC in MFM
FD	Write FD with CLK = FF	Write FD in MFM
FE	Write FE, CLK = C7, Preset CRC	Write FE in MFM
FF	'Write FF with CLK = FF	Write FF in MFM

FDC Type IV Commands

The Forced Interrupt Command is used to terminate a multiple sector read or write command or to insure Type I status in the Status Register. This command is loaded into the Command Register at any time. If there is a current command under execution (Busy Status Bit set) the command is terminated and the Busy Status Bit reset. The lower four bits of the command determine the conditional interrupt as follows:

- i0,i1 Not used with the WD1772
- i2 Every Index Pulse
- i3 Immediate Interrupt

The conditional interrupt is enabled when the corresponding bit positions of the command (i3-i0) are set to a 1. When the condition for interrupt is met the INTRQ line goes high signifying that the condition specified has occurred. If i3-i0 are all set to zero (Hex $D0), no interrupt occurs but any command presently under execution is immediately terminated. When using the immediate interrupt condition (i3 = 1) an interrupt is immediately generated and the current command is terminated. Reading the status or writing to the Command Register does not automatically clear the interrupt. The Hex $D0 is the only command that enables the immediate interrupt (Hex $D8) to clear on a subsequent load Command Register or Read Status Register operation. Always follow a Hex $D8 with a $D0 command.

Wait 16 µsec (double density) or 32 µsec (single density) before issuing a new command after issuing a forced interrupt. Loading a new command sooner than this nullifies the forced interrupt.

Forced interrupt stops any command at the end of an internal micro-instruction and generates INTRQ when the specified condition is met. Forced interrupt waits until ALU operations in progress are complete (CRC calculations, compares, etc.).

[8] Missing clock transition between bits 4 and 5.

[9] Missing clock transition between bits 3 and 4.

Status Register

Upon receipt of any command, except the Force Interrupt Command, the Busy Status Bit is set and the rest of the status bits are updated or cleared for the new command. If the Force Interrupt Command is received when there is a current command under execution, the Busy Status Bit is reset, and the rest of the status bits are unchanged. If the Force Interrupt Command is received when there is not a current command under execution, the Busy Status Bit is reset and the rest of the status bits are updated or cleared. In this case, Status reflects the Type I commands.

The user has the option of reading the Status Register through program control or using the DRQ line with DMA or interrupt methods. When the Data Register is read the DRQ bit in the Status Register and the DRQ line are automatically reset. A write to the Data Register also causes both DRQ's to reset.

The Busy Bit in the status may be monitored with a user program to determine when a command is complete, in lieu of using the INTRQ line. When using the INTRQ, a Busy Status check is not recommended because a read of the Status Register to determine the condition of busy resets the INTRQ line.

The format of the Status Register is shown below:

BITS							
7	6	5	4	3	2	1	0
S7	S6	S5	S4	S3	S2	S1	S0

Because of internal sync cycles, certain time delays are observed when operating under programmed I/O, as shown.

Operation	Next Operation	Delay Req'd.	
		FM	MFM
Write to Command Reg.	Read Busy Bit (Status Bit 0)	48µsec	24µsec
Write to Command Reg.	Read Status Bits 1-7	64µsec	32µsec
Write Register	Read Same Register	32µsec	16µsec

Status Register Description

BIT NAME	MEANING
S7 Motor On	This bit reflects the status of the Motor On output
S6 Write Protect	- On Read: Not Used. - On any Write: It indicates a Write Protect. This bit is reset when updated
S5 Record Type / Spin-up	- On Type I commands: When set, this bit indicates that the Motor Spin-Up sequence has completed (6 revolutions). - On Type II & III commands: this bit indicates record Type. 0 =Data Mark. 1 = Deleted Data Mark.
S4 Record Not Found (RNF) / Seek Error	- On Type I commands: When set the desired track was not verified - On Type II & III commands: When sets it indicates that the desired track, sector, or side were not fount. This bit is reset when updated.
S3 CRC Error	If S4 is set, an error is found in one of more ID fields; otherwise it indicates error in the data field. This bit is reset when updated.
S2 Lost Data Byte / TR00	- When set, it indicates the computer did not respond to DRQ in one byte time. This bit is reset to zero when updated. - On Type I commands, this bit reflects the status of the TR00 signal.
S1 Data Request / Index Pulse	- This bit is a copy of the DRQ output. When set, it indicates the DR is full on a Read Operation or the DR is empty on a Write operation. This bit is reset to zero when updated. - On Type I commands, this bit indicates the status of the IP signal
S0 Busy	When set, command is under execution. When reset, no command is under execution

Status Register Summary

Type	Command	S7	S6	S5	S4	S3	S2	S1	S0
I	All Type I	MO	X	SU	SE	CRC	T0	IP	BSY
II	Read Sector	MO	X	RT	RNF	CRC	LD	DRQ	BSY
II	Write Sector	MO	WP	RT	RNF	CRC	LD	DRQ	BSY
III	Read Address	MO	X	X	RNF[10]	CRC	LD	DRQ	BSY
III	Read Track	MO	X	X	X	X	LD	DRQ	BSY
III	Write Track	MO	WP	X	X	X	LD	DRQ	BSY
IV	Interrupt while busy	-	-	-	-	-	-	-	0
IV	Interrupt while idle	MO	X	X	X	X	T0	IP	0
	Idle	MO	WP[11]	X	X	X	T0	IP	0

Where:

- 0 = always 0
- X = undefined
- - = retains previous value
- MO = motor on (1 = motor on)
- WP = write protect (1 = write protected)
- SE = Seek Error (1 track not verified)
- SU = spin up (1 = spin-up completed)
- RT = record type (1 = deleted data)
- RNF = record not found (1 = record not found)
- CRC = CRC (1 = CRC error ➔ if rnf=1 error in ID field, if rnf=0 error in data field)
- T0 = TRK00* (1 = head at track 0)
- LD = lost data (1 = lost data)
- IP = index pulse (1 = disk at index pulse)
- DRQ = data request (1 = data register requires service)
- BSY = busy (1 = controller busy)

[10] Not documented but when performing a ***read address*** to a track without addresses the RNF bit is set.

[11] Not documented but after execution of a Type I command the WP status bit is continuously updated and can be polled like MO and T0, but this is **not true** after a Type II or III command

Floppy Disk Programming

This section indicates how to program the FDC and related chips (DMA, PSG, and MFP) to perform FD operations.

General information / Tips

- All the FDC commands are executed by sending the command to the FDC through the DMA chip. First the drive must be selected by setting the proper outputs of the PSG chip, and then the command is sent to the FDC. Most FDC commands are normally terminated when the INTRQ is raised. This can be tested by polling the proper input of the MFP or it can generate an interrupt. The only two exceptions are:
- Force Interrupt commands: The only two useful force interrupt commands are the *force interrupt with no interrupt command* ($D0 command), and the *force interrupt every index pulse command* ($D4 command).
- Read/Write multiple sectors command: in this case the command is terminated when all the bytes required are transferred by sending a force interrupt command.
- For any command sent to the FDC it is recommended to setup a watchdog. If the command does not execute correctly after some time, send a $D0 to force interrupting the command.
- At completion of a FDC command you have to read the status to find out information on how the command executed. Most of the status bits are not changed until a new command is received by the FDC. Exception are for bits S7 (MO), bit S1 (IP), and sometimes bit S2 (TR00*) signals which are continuously updated outside of the execution of a command.
- Wait until the motor is turned off by the FDC (by checking status bit S7) before deselecting it. If the drive is deselected before the FDC has automatically turned off the MO, the FDC will not receive the IP and the motor will stay on forever (not good idea for your floppies!).
- FDC Bug: The ***read-track*** command should read a number of bytes of less than 6500. However, it seems that the WD1772 has a bug and from time to time this command fails and returns a huge number of bytes (up to 21000 bytes!). In this case you should retry.
- Not documented in the FDC documentation, after execution of a Type I command the WP (bit S6) is set in the status register. If you want to poll the WR status bit (for example to check if a non-protected disk has been ejected, you have to leave the WD1772 to a Type I “idle mode”. This is done by following Type II or type III commands by a Type I command (like a seek to current track command). Note that this is not necessary after Type III commands that always returns Type I status after (this is documented).
- Remember that the reading of bytes from the FDC is done through the DMA FIFOs in multiple of 16 bytes. So up to 15 bytes might get stuck in the FIFOs. Therefore, you should take this in account for the read address and read track commands.

Typical Floppy Disk Operations

This section of the document describes typical functions that should be provided in a Floppy Disk Library.

Enter Supervisor mode

Remember that access directly to the Hardware can only be done in supervisor mode.

Drive Select

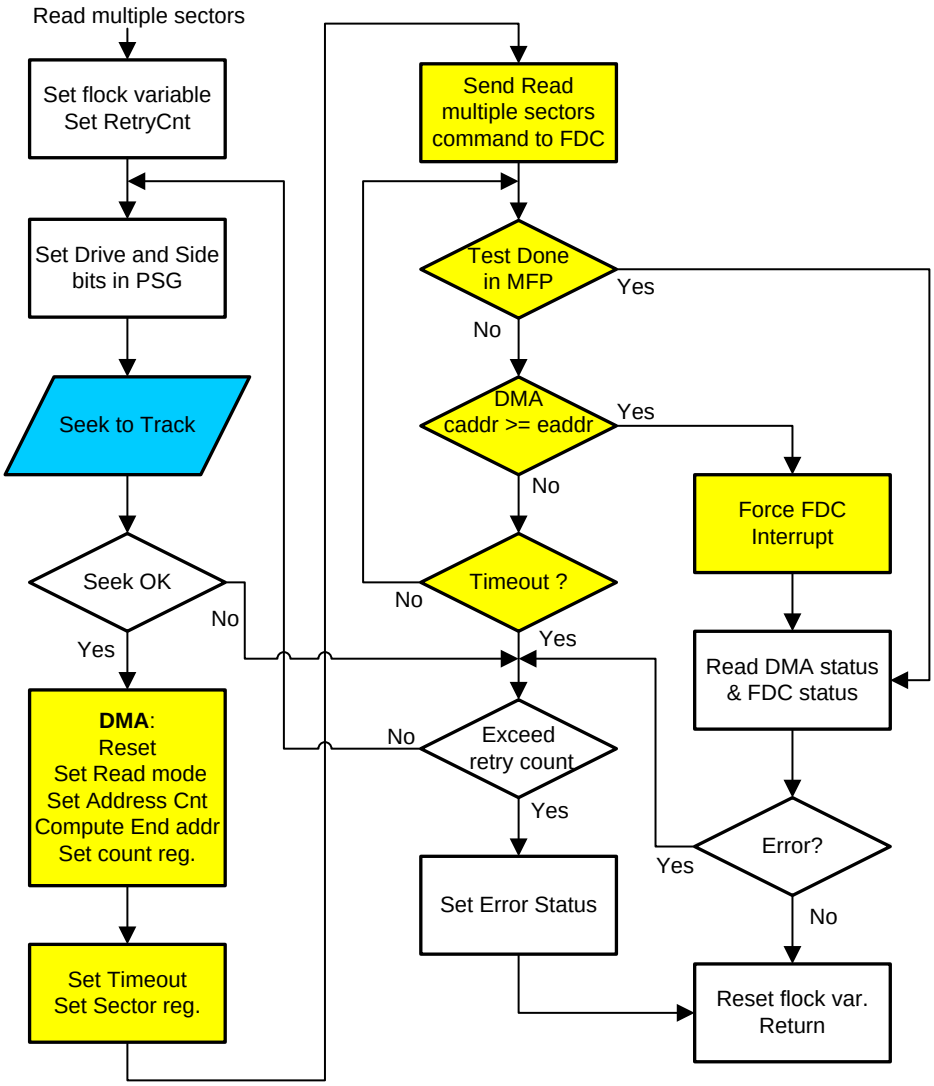
Before sending any command to the drive it is necessary to select it. This is done by using the low level access to the PSG as described in the PSG Programming section.

Seek to Track

If the drive has already been used and the Track Register contains the correct value, it is only necessary to send a seek command to the FDC. If the position and/or the value of the Track Register is unknown it is necessary to send a *restore command* before the *seek command* to reset the track register to zero and position the head accordingly.

Multiple sectors read

The following diagram shows a typical read sequence for multiple sectors. Notice that this sequence uses the “seek to track” sequence previously described.



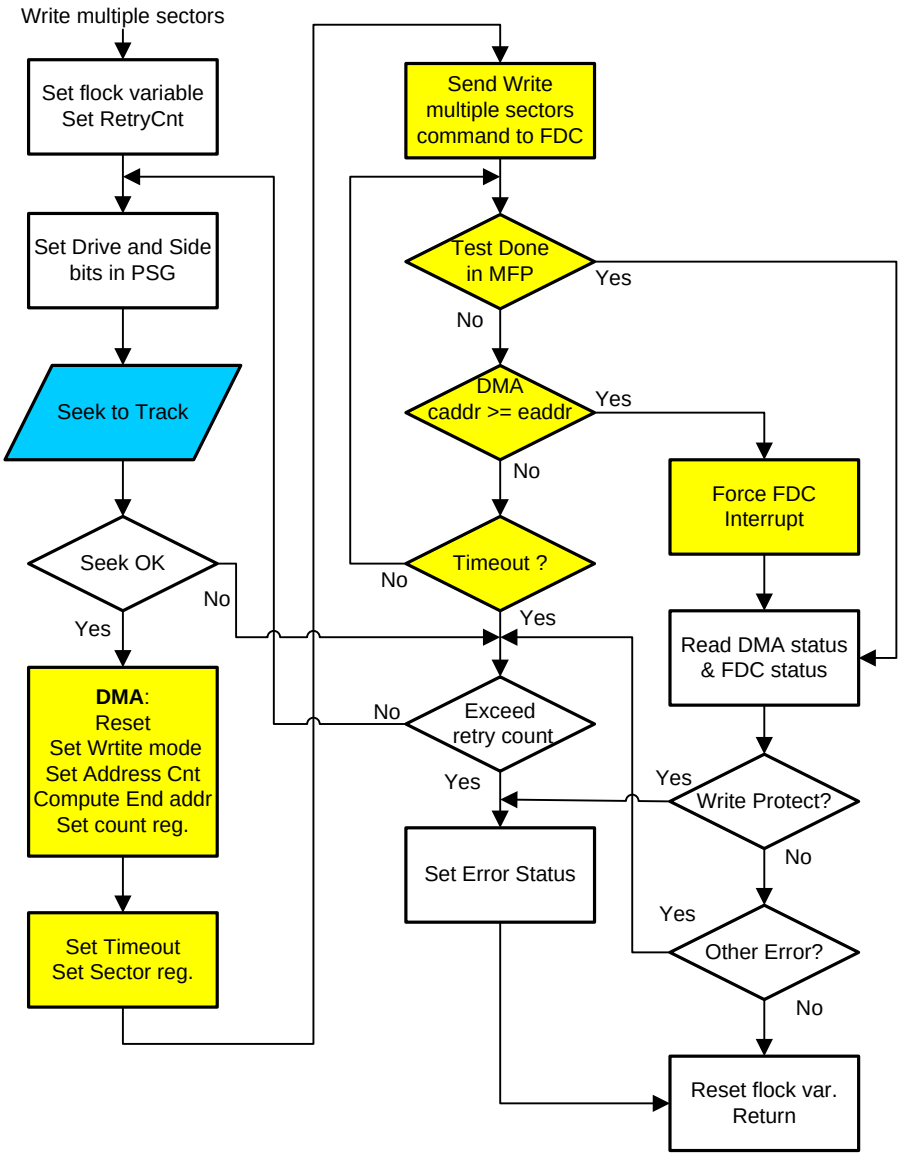
Read Multiple Sectors Sequence

Few things to note:

- If read does not succeed the first time it is recommended to try several times (e.g. 4 times). You have to know that on an Atari FD retries happen more often that you may think!
- When 2 consecutive read fails, it is recommended to “shake” the head back and forth (using restore / seek) to eventually remove dust particles under the head (this is not shown in the diagram). This is sometimes referred as the shoe-shine technique.
- For multiple sectors read it is necessary to send a force interrupt command when the end address has been reached. Therefore, in multiple sectors mode you have to read the current address in the DMA address register.
- The flock variable must be set during the FDC operation to be sure that the DMA is not accessed by the VBL routine.

Multiple sectors write

The following diagram shows a typical write sequence. Notice that this sequence uses the “seek to track” sequence previously described.

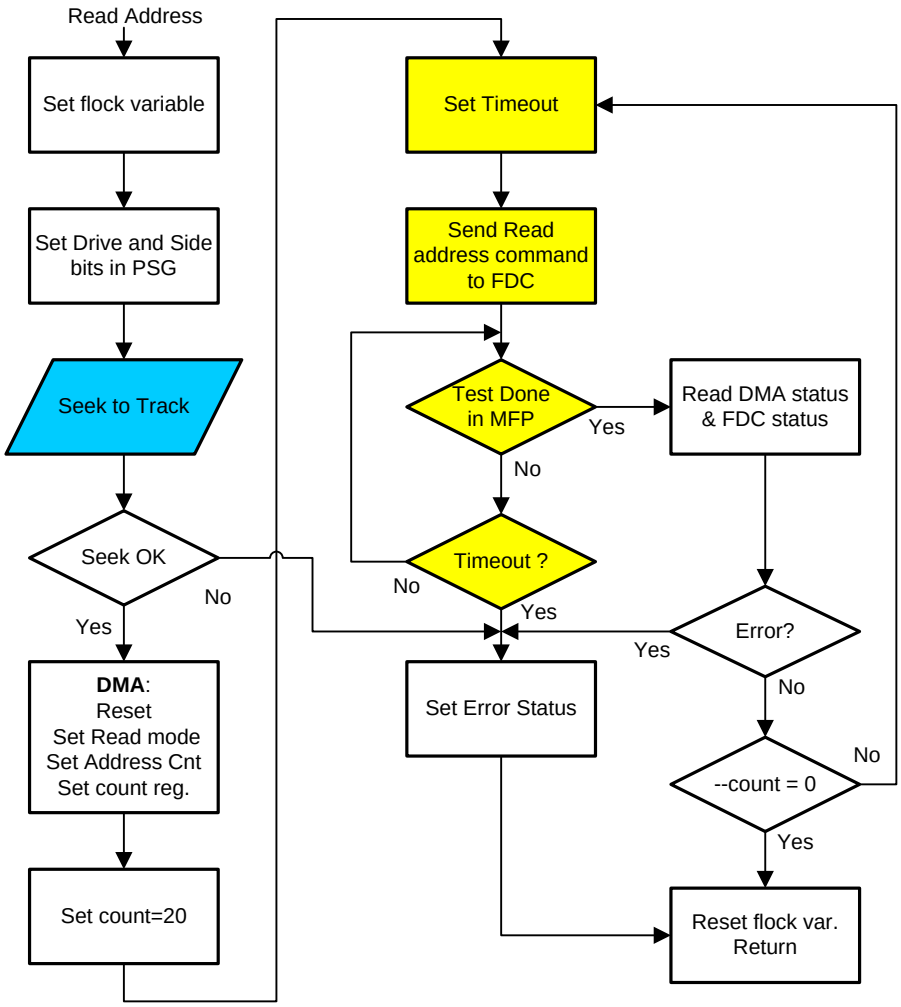


Few things to note:

- If write does not succeed it is recommended to try several times (e.g. 4 times).
- When 2 consecutive write fails, it is recommended to shake the head back and forth (using restore / seek) to eventually remove dust particles under the head.
- For multiple sectors write it is necessary to send a force interrupt command when the end address has been reached. Of course the current address is read from the DMA.
- The flock variable must be set during the FDC operation to be sure that the DMA is not accessed by the VBL routine.

Read Address

The following diagram shows a typical write sequence. Notice that this sequence uses the “seek to track” sequence previously described.

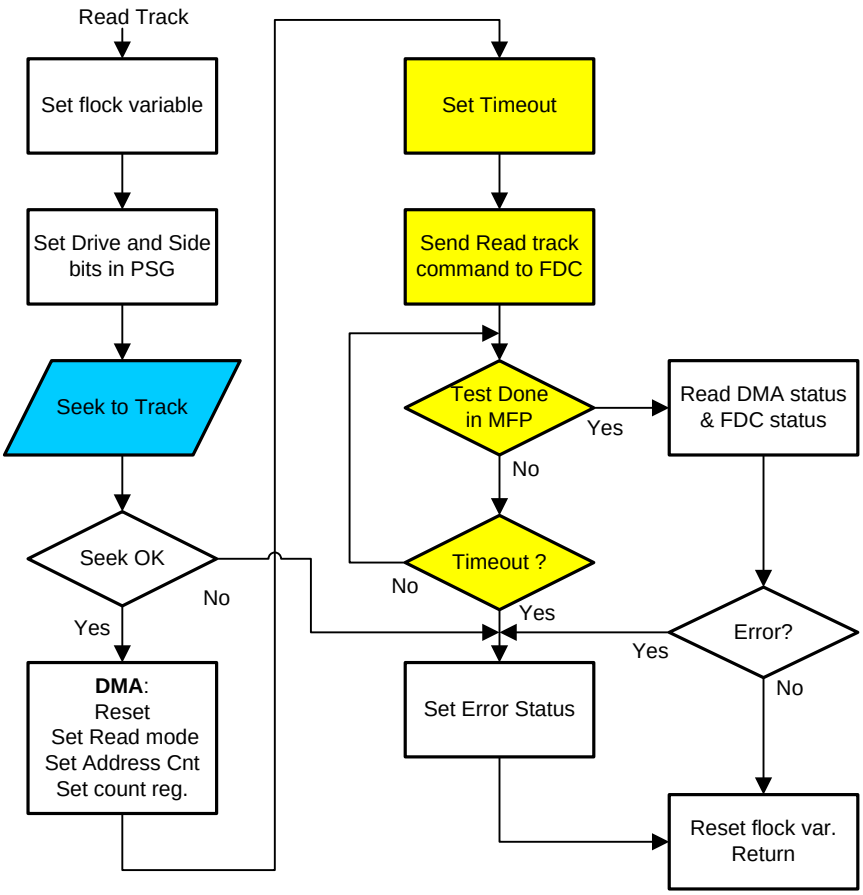


Few things to note:

- Beware that a read address command only transfers 6 bytes. If you only issue one read address command, there are not enough bytes in the FIFO to start a bus request/grant and therefore nothing is transferred to memory.
- It is possible to read all the addresses in a track starting from the Index pulse. For that matter you should send a Force interrupt command (with every index pulse set), poll the FDC interrupt bit in the MFP, and when set send several (20 is a good number) read address commands with only one DMA operation. Now you just need to interpret the content of the buffer.

Read Track

The following diagram shows a read track sequence. Note that no retry is performed.



Few things to note:

- Beware that a read track command transfers an unknown number of bytes that is not necessary a multiple of 16. If you really want to get all the bytes (usually not very important) you need to send extra command that reads enough data to flush the FIFO. As a maximum of 15 bytes may be stuck in the FIFO it is possible to send 3 extra read address command to force a flush of the FIFO and to process the buffer to remove possible extra data in the buffer.

Write Track

The write track is similar to the read track.

Measurement of FDC bytes time-width

It is sometimes useful to measure the time it takes for a byte to be assembled by the FD. A normal byte is assembled by the FDC every 32µs (8 bits of 4µs) resulting in a DMA burst transfer request every 512 µs (16 bytes). Some protection mechanisms apply various possible variations on the bit width and it is therefore useful to be able to measure these variations.

For that matter we are going to use one of the 68901 MFP timers. Usually timer A is a good choice as it is only used for sound on Atari STe. The MFP is connected to a 2.4576 MHz crystal and the timer A offers several pre-scaling. The best choice is to use a pre-scaling of 10 (this will become obvious later). This pre-scaling results in a frequency of 245.76 KHz and a therefore a period of 4.0690104167 µs (you can use 4069 ns as a good **integer** approximation). As the timer register is 8 bits wide, an overflow will happen every 256 ticks or about every 1042 µs. Therefore, unless we use interrupts, we must ensure that we poll the value of the timer in less than 1 ms in order not to miss any overflow.

When a read sector command (or for that matter a read track command) executes we need to set up a loop that waits for the INTRQ to be raised by the FDC indicating the end of command. This is done by checking the bit 5 of the MFP GPIO.

Inside the same loop we also need to check if the DMA address register has been increased. This change happens every 16 received characters or, at nominal rate, every 512 µs.

Therefore, the pseudo code for measuring timing looks like this:

```
Prepare the FDC (select, seek, etc)
Prepare the DMA (read mode, buffer address, count)
Prepare the timer (reset, set pre-scale to 10, start)
Loop {
      Read DMA address
      Has dma address changed ?
            Yes Get & store timer time, store new address
      Has the FDC raised the INTRQ ?
            Yes break
}
```

As we can see the main actions in this loop is:

- Read the DMA address, to check if it has changed, and
- Check the MFP GPIO register to see if the command has terminated.

The loop must take less than 512µs in order not to miss an address change and this should not be a problem.

But the precision of the measurement is directly related to the execution time of the loop (time when both tests fail). For example, if the loop takes 100 µs the precision will be of 100/512 or about 20% for 16 bytes or about 1.2% for a byte which is not acceptable. It is therefore important to optimize as much as possible this loop. For example, I have optimized this loop to less than 15 µs and this represents a worst case precision of 15/512 (about 3%) on a chunk of 16 bytes or about 0.2% per byte. When shortening the loop do not forget that you still have to handle the timer overflow (but if you are smart it should not affect the loop time).

As mentioned above a pre-scale of 10, resulting in a change every 4µs, is a good choice. If you remember that an address change in the DMA is occurring every 512 µs this corresponds to about 128 timer ticks with this pre-scale. The value 128 happens to be just the median value ($80) of an unsigned byte. This is very convenient to store, in array of byte, the variation of each 16 bytes' chunks transferred. Note also that 4µs provides a precision which is in line with the loop time.

Without taking any special care in the above loop you will notice that on regular basis the values measured/stored are completely off. For example, you will get one value largely bigger than normal and the next one largely shorter to compensate. It should not take you too much time to figure out that the problem comes from the fact that the processor is interrupted. You can leave with this problem by post processing the values: you correct any pair of wrong values by replacing both values with the mean of the two. But obviously a much better solution is to enter a critical section at the beginning of the loop above by turning off all the interrupts.

Note that you do not get the timing for the first 16 bytes as there is no reference point on when the transfer effectively starts. I have tried to come with a way to measure the timing for this first chunk which implies to know when the first byte is transferred. The DMA has a status register that reflect the state of the FDC DRQ signal in bit 3. In the Atari HW documentation it is explicitly said that it is a bad idea (i.e. don't do it) to query the status register during DMA transfer. This makes sense as the DMA has two sides: one toward the FDC to transfer bytes, and one toward the 68000 to read and writes DMA registers (not to mention DMA transfer).

Consequently, it is probably too much load for the DMA to transfer a byte to/from the FDC while the 68000 try to read/write some internal register. However, it is possible to get the time of the first DRQ by reading the DMA status. As a matter of fact, as the transfer has not yet started we are not creating too much perturbation to the DMA and it works fine. This mean that just before the loop already described we need to add another tight loop that just check for the first DRQ (just after the IP), and at the end of the loop we store the current time which correspond to the time of transfer of the first byte. The idea seems interesting but unfortunately it does not work for reasons that would be too long to explain here.

References

- Engineering Hardware Specification of the Atari ST Computer System The Atari Corporation - Sunnyvale, California - 7 January 1986
- *How To Use The Atari St Hard Disk Port* Doug Collinge - School of Music, University of Victoria
- Atari ASCI/DMA Integration Guide – June 28, 1991
- Atari ST/STe/MSTe/TT/F030 Hardware Register Listing Dan Hollis - 1/22/94
- Atari ST Internals - Abacus Software 1985
- Le livre du Développeurs sur Atari ST - Micro Application -1989
- WD177X-00 Floppy Disk Controller/Formatter - Western Digital Corporation
- WD1772 Spécifications V1.2 – Jean Louis-Guerin 2014
- Probing the FDC – Davis Small (Start Vol. 1 no. 2 fall 1986)
- Flop.s Floppy Disk Driver – Atari corporation 1985
- Hitchhiker's guide to the BIOS – Atari corporation 1985
- Atari ST/Mega – Edition Weka 1991
- Floppy disk library – Jean Louis-Guerin 2008
- Email from Nicolas Pomarède <npomarede> 2013

Revision

- V1.2 Corrected an error on PSG side selection graphic on page 12. November 2019
- V1.1 Added many information about Hard Disk Programming in the DMA section and added lots of new information. The goal is to provide in future a document that includes FD and HD programming. September 2013
- V1.0 Initial published revision – October 2008