X3T10/95-143Rø

X3T10/94- r 1.1
January 26, 1995 12:37 am

Proposal for ATA-3 Features

ATA-3 & ATAPI

Devon Worrell
January 26, 1995

WESTERN DIGITAL

1651

1.0 Introduction

1.1 Purpose

The purpose of this document is to describe a proposal from Western Digital. This proposal will add functionality to the ATA-3 Specifications.

The issue is that more people in the world are now aware that more is to be gained by enhancing ATA than by promoting SCSI into the volume PC space. The obvious opportunity is to overlap the ATA channel, i.e. send one command per device. The motivation for overlapped ATA is multi-tasking OS. The next obvious opportunity is to add queuing, i.e. more than one command per device. This would allow for the DMA channels be multi-threaded.

We must realize that the existing single threaded dual channel DMA, single threaded dual channel ATA solution hasn't shipped and we need to get market focus behind it....not focus all the interest on something that isn't available...ie. SWAT...Sell What is Available Today. The multi-threaded thing is at best a 2H95 product, more likely a 1H96 product.

1.2 Scope

This document will cover the key architectural points and other design issues. It is intended that this document be the basis for industry discussion.

1.3 Audience

This document is intended for use by Western Digital Engineers and Managers and attendees of ATA and SFF.

1.4 Technical Terms

There are a number of technical terms being used today to describe the capabilities this document proposes be added to the ATA interface. This of course is natural (Given that Marketing is involved), but could cause confusion when reading this proposal.

Multi-thread, Multi-threaded, Multi-threading	This must be the most misused term used today. It is used to describe everything from operating systems to lines at the bank. For the purposes of this document this term will be used to describe a very general capability of the ATA interface only. When used to describe the ATA Interface then it will mean that the interface can be used to transmit new commands to device(s) on the cable before the completion of any previous (pre-existing) command(s). This term will NOT be used to describe any capability of the Host or Devices attached to the ATA Cable.
Single Thread, Threaded or Threading	This will be used to describe the opposite of Multi-thread and will be used to describe a general capability of the ATA Interface only.
Overlap, Overlapped, Overlapping	This will be used to describe a capability of Device(s) attached to the ATA Cable.
Arbitration	When used within the context of this proposal, this will refer to the sharing of common signals or other resources (Registers)used

	in conjunction with the ATA interface, Host computers or Devices using the ATA interface.
Command Queuing	Simply put, allowing more than one command at a time to be accepted by the ATA Device.
Tagging	No not showing the world that you exist. This is synonymous with Command Queuing and is used interchangeably.
Controller	The electronics on a peripheral, that provides the ATA functionality.
IDE	Integrated Drive Electronics. The industry knows the AT Attachment (ATA) interface by this name. ATA, ATA-2 and ATA-3 are used by the Standards bodies as a pseudonym for the ubiquitous IDE Interface.
Semaphore	Used in the context of this proposal, a semaphore is any mechanism that is used to lock access to a common resource.
Task File, Task File Registers	This refers to the Registers that are used to control an ATA device. For the purposes of this proposal only those registers and signals that can be altered by either the Host or the Device are considered part of the Task File Registers. e.g. BSY, DRDY, DREQ etc. are NOT considered part of the Task File.
Control Block	Same as Task File Registers.
Post	That part of the Power-up or initialization sequence that is used to setup the hardware characteristics. Normally this operation is performed by the System BIOS (for x86 based systems.)
Simple Queued Commands	Simple Queued Commands is the capability in the Device to accept multiple commands. Each of these commands can then be reordered by the device. The device has complete freedom to reorder the queued commands.

2.0 Goals

It is the intent of this proposal to provide some basic improvements to ATA. The most basic improvement will be to allow commands to be sent to each of the two devices on one ATA Cable, independently. That is allowing commands to be processed in both devices at the same time. A further improvement will be to allow multiple commands to be sent to each of the devices on the ATA Cable. Although this may seem simple, the compatibility issues are extremely complicated.

2.1 Generic Goals

Keep it as simple as possible.
Allow Commands to be sent to each Device on the ATA cable independently.
Allow mixing of Legacy Devices and Newer capability Devices and still use the new capabilities.
Use DMA and independent commands as basis for greatest performance improvement.
Reduce the number of Interrupts to further improve performance.
Report Capabilities and Default to Legacy Mode.
Enable Capabilities that are backward compatible in Post, using Set Features.
Enable Capabilities that are not backward compatible on a command by command basis.
Single solution for ATA and ATAPI peripherals.
Use only the existing task file register space
Maintain compatibility with current silicon.
Support both PIO and DMA modes.
Support existing PCI Bus Master DMA
Support transfers of more than 512 bytes on any given DRQ Interrupt.
Allow error reporting after transfer on reads.
Allow for a minimum queue depth of 64.
Allow the drive to release the Task File Registers before command completion.
Support only simple queued commands.
Minimize Host polling requirements.

2.2 Other Possible Areas for Improvement

There are some other areas that should be investigated for ATA-3:

One Interrupt for both Primary and Secondary Channels.
Parity on Data Transfers.
Reduced Command Set.
3.3v Interface.

Note that this proposal goes no further than to point out the need for the above features.

3.0 Marketing considerations

Some would say that all these changes are not needed, after all there is the SCSI interface. Some would also say that these changes are producing an ATA-3 interface that is no different in complexity from SCSI.

We all should realize the mistakes were made with SCSI. You must also realize that ATA runs to a different model. We would not use the comparison Simple versus Complex, instead we would use: More Control and Cheap versus Less Control and not so Cheap. Clearly ATA has always been and always will be much closer to an implementable standard due to it being a register based interface, whereas SCSI is a protocol based interface.

3.1 History

If you take a step back, you will realize that any interface is comprised of only four elements:

Physical:	Connectors, Jumpers, Form Factor
Electrical:	Signals, Voltages, Timings
Protocol:	Sequence, Bits, Registers
Commands:	Operations

If you apply this model against the 10 years since the original PC/AT hard disk controller shipped and then transitioned to ATA, you will see that progress had occurred where the PC market needed it to.

3.1.1 Physical

ATA has 3 physical interfaces:

- 3.5": 40 pin I/O connector, 4 pin power
- 2.5": 44 pin I/O that has 40 pins of I/O and 4 of power
- 1.8": 68 pin PCMCIA

3.1.2 Electrical

ATA has had a relatively constant electrical interface with the following areas of growth:

- PDIAG-/DASP to address compatibility during transition from controller/ST-506 drives to ATA drives.
- DMARQ and DMACK to support DMA transfers.
- Faster PIO Data Register Timing.
  Mode 0: 600 ns per word - 3.33 Mbyte/sec.
  Mode 1: 383 ns per word - 5.22 Mbyte/sec.
  Mode 2: 240 ns per word - 8.33 Mbyte/sec.
  Mode 3: 180 ns per word - 11.1 Mbyte/sec.
  Mode 4: 120 ns per word - 16.66 Mbyte/sec.
- Faster Multiword DMA Data Register Timing.
  Mode 0: 480 ns per word: 4.16 Mbyte/sec.
  Mode 1: 150 ns per word: 13.33 Mbyte/sec.
  Mode 2: 120 ns per word: 16.66 Mbyte/sec.
- CSEL to ease installation and eliminate jumpers

3.1.3 Protocol

Until ATAPI, ATA has only added more advanced data transfer protocols and had relied upon the Command Register write, BSY bit, DRQ bit, and IRQ protocol for almost everything:

- Multi-sector PIO Read/Write.
- DMA Read/Write.
- Packet Interface. (ATAPI)

3.1.4 Command Set

Since the AT controller first shipped in 1984, command functionality has been added in the areas of:

- IDENTIFY to report capabilities.
- SET FEATURES to enable advanced capabilities.
- Multi-Sector PIO Read/Write.
- DMA Read/Write.
- Power management.
- Download microcode.
- Removability.
- Packet Interface. (ATAPI)
  - CD-ROM command sets.
  - Tape command sets.

3.2 Issues with SCSI

The major issues with SCSI within the PC market were:

- No effort was made to standardize the hardware registers used to integrate it into the PC. What the PC OEMs want are standard register sets, standard command sets, and embedded operating system support. This forced either:
  - Vendor supplied drivers, or
  - Layered device driver model with vendor supplied hardware driver (aka mini-port)
- A command set that was too complex and subject to too much interpretation.
- A protocol that was too complex for single-user machines.
- A cost point that was/is too high.

It should come as no surprise that the industry is trying to take the ATA cost benefits and extend the capabilities. The beauty of ATA is that it defines a register set, a protocol, a command set, an electrical interface, and a physical interface all within one document. PCI Bus Master DMA is obvious, as is moving from the domain of a single threaded channel to a multi-threaded channel to a queuing device.

3.3 Model for enhancing ATA

The obvious model is:

- Power up to Legacy/Compatibility mode.
- Report advanced capabilities via Identify Drive.

- Have the OS driver enable the advanced capabilities via Set Features command or use arguments to individual enhanced capability commands.
- Have the OS driver communicate with the ATA device via the advanced capabilities protocol/commands.

3.3.1 Timing Control

We all must admit that none of us like the BIOS controlled local bus ATA hardware, but a standard register set would have been impossible due to these devices running at different clock frequencies and the requirement that they be programmed to support fixed Mode 0/1/2/3/4 PIO ATA timings. Fortunately, the PCI DMA model is 100% register standard, thus this will be easier for all.

3.4 ATAPI Disk

As long as ATA retains a single register set, it makes embedded OS integration much easier. With the advent of ATAPI compliant ATA CD-ROM, ATAPI compliant ATA Tape, and ATA compliant ATA disk, it seems natural for the following to occur:

- Definition of ATAPI compliant ATA disk
- Move ATA support into the layered device driver model, with ATA/ATAPI handled via a mini-port. This allows the OS to support Disk, Tape, and CD-ROM via a single TSD for SCSI or ATA.

3.5 Focus for this Proposal

Western Digital's impression is that PCI has killed VL. Intel worked to get SFF to accept a PCI Bus Master DMA ATA register level specification. The goal is a common register set in order to facilitate embedded operating system (i.e. MS) support. Why? If $200 CPU uses 10% of bandwidth for ATA PIO data transfer, then this cost is $20. Multimedia performance is shown to be directly related to amount of CPU bandwidth that can be allocated to it. DMA frees the CPU to do multimedia better.

PCI chipsets today are high pin-count, thus the die sizes are pad limited. This gives them lots of logic to work with as long as the function doesn't add pins. Lets just say that Intel is setting the PCI chipset feature bar and that their 486 and Pentium PCI chipsets have embedded EIDE in the CPU to PCI bridge. The design multiplexes PCI and ATA onto the same pins, thus it costs no pins, and uses up the excess die size...it is free! Unfortunately, SCSI can't be done the same way (i.e more gates, more pins) and I don't see it ever getting integrated into the chipset for this reason and the reason of embedded software support (i.e. Chicken/Egg situation... chipset vendor would want embedded OS support. Which means they would have to license architecture/design from existing SCSI LSI supplier. The Intel Pentium chipset includes the PCI Bus Master DMA ATA capability.

Additionally, the world is trying to add ATA capabilities to issue one I/O per ATA device, not one I/O per ATA channel. The key issues have to do with philosophy (some now, some later versus all later) and where to arbitrate (host hardware/software versus peripheral side) for control of the ATA task file, the ATA IRQ and the ATA DMARQ/DMACK. This topic has started within a working group of ATA (ATA-3). Basically there is a WD proposal that takes a phased approach and has host arbitration and a Quantum/Conner approach which takes a Big Bang approach and has peripheral arbitration.

Some basic points to remember:

- ATA is getting more capable and staying cheap.
- SCSI has become cheaper and easier, but it isn't going to get anywhere near as cheap as ATA since it isn't in the chipsets.
- ATAPI CD-ROM is now 2X and 4X. There are requests for ATAPI HDCD, ATAPI CD-R, and ATAPI changer specs.
- ATAPI Tape is now starting to happen.

- The future direction for high speed ATA data transfer is DMA. PIO transfers will become a thing of the past. ATA will only be increasing the data transfer rates when using DMA in the future.
- A push is starting for Overlapped Commands and Queued Commands.
- PCI Bus Master ATA is starting to happen (one command/channel). This same capability can be leveraged by simple Driver modifications to support interleaving of data transfers from each device on the cable.

4.0 Overview

Rather than delve into the complexities of this proposal, a general overview of the Reasons, Concepts and Specific Proposals is discussed in this section.

4.1 Key Points and General Approach

- Phase in the Changes. Although the changes to the interface should be phased in, the actual definition of them shall occur before any implementations are attempted.

  Start in 1995.
  More advanced capabilities in 1996.
- Provide solutions for the Customers of ANSI and SFF.

  Host Silicon providers (e.g. Intel PCI and Bus Master DMA).
  BIOS (e.g. Phoenix).
  Drivers (e.g. Microsoft, IBM, Apple).
  System Vendors.
- Focus on Overlapped operation.
- Allow but de-emphasize more advanced Server and Array capabilities.
- Obtain OS and Hardware solutions in a timely manner.

  Support in Win 95 Release 2.
  Support in the PC'95 and/or PC'96 Hardware Design Guide.

4.2 Assumptions / Constraints used in the generation of this proposal

- Don't alter the key value that ATA currently has (Cost, Performance, True Register Set Standard).
- Phase in the improvements.
- Allow first implementations without any hardware changes.
- Always be backward compatible.
- ATAPI & ATA Features should be implemented identically where possible.
- Improve Performance only where complexity and cost are not increased.
- Overlapped operations will be mainstream, Queuing will not be within the 1995-96 timeframe.
- Reducing number of host interrupts will be beneficial.
- System suppliers, Motherboard manufacturers, BIOS and OS suppliers will decide what is acceptable and what is not.

4.3 Phased approach

The proposed features should be phased in over one to two years. There are five distinct capabilities. Each of these can be used when all of the previous capabilities are implemented.

- Overlap in a mixed legacy and overlap capable environment (Legacy Overlap).

  Allow CD-ROM and Tape Drives to release the Task File Registers and commands to be sent to legacy device (e.g. the faster ATA Disk Drives).
- Overlap in a homogeneous overlapping environment (Homogeneous Overlap). Note that the only distinction between the mixed mode and homogeneous environment is that the Driver must control the overlap for the Legacy Devices, the Overlap capability will be implemented only one way. There will be the possibility for the Overlap capable Device to be either Microprocessor driven or Accelerated.

  Add Overlapped command capability to all the devices.
  Add Proxy Interrupt support.
  Use existing PIO and DMA.

- New Overlapped PCI DMA Capability (Reduced Host Interrupts).
  Allow each Drive on the ATA cable to use the DMA Controller without any Host (Driver) intervention.
- Command Queuing and Tagging.
  Multiple commands for each Drive on the ATA cable.
- Advanced DMA for Servers and Arrays.
  Allow Tag information to be processed by the DMA Controller.

4.4 Basic Building Blocks

- Arbitration of the Task File Registers.
  - Release of Task File Registers. A new wrinkle in the protocol will allow the peripherals to clear the BSY signal before the completion of the command, thus allowing the registers to be used for sending command(s) to another device.
  - Selection Command (A2h). This allows the registers to be given back to the Device when needed.
- Arbitration of Interrupt Request (INTRQ) line.
  - Proxy Interrupt.
  - Service Status bit for detecting who interrupted.
- Arbitration of DMA Control (DMARQ, DMACK) signals.
  - Overlapped DMA uses INTRQ to signal ready to transfer of data just like the PIO protocol.
  - Overlap PCI DMA Bridge Logic reads & writes Task File Registers directly.
  - Interlock function in PCI DMA Bridge Logic controls access to ATA Registers.
- Overlap Capable PCI DMA Drive.
  - DMA Ready Status.
- Command Queuing
  - Communication of Tags (ATA Feature Register / ATAPI Tag Register.)
- Advanced or Descriptor Based PCI DMA.
  - Pointer to memory block used to select DMA control information for each Tagged command.

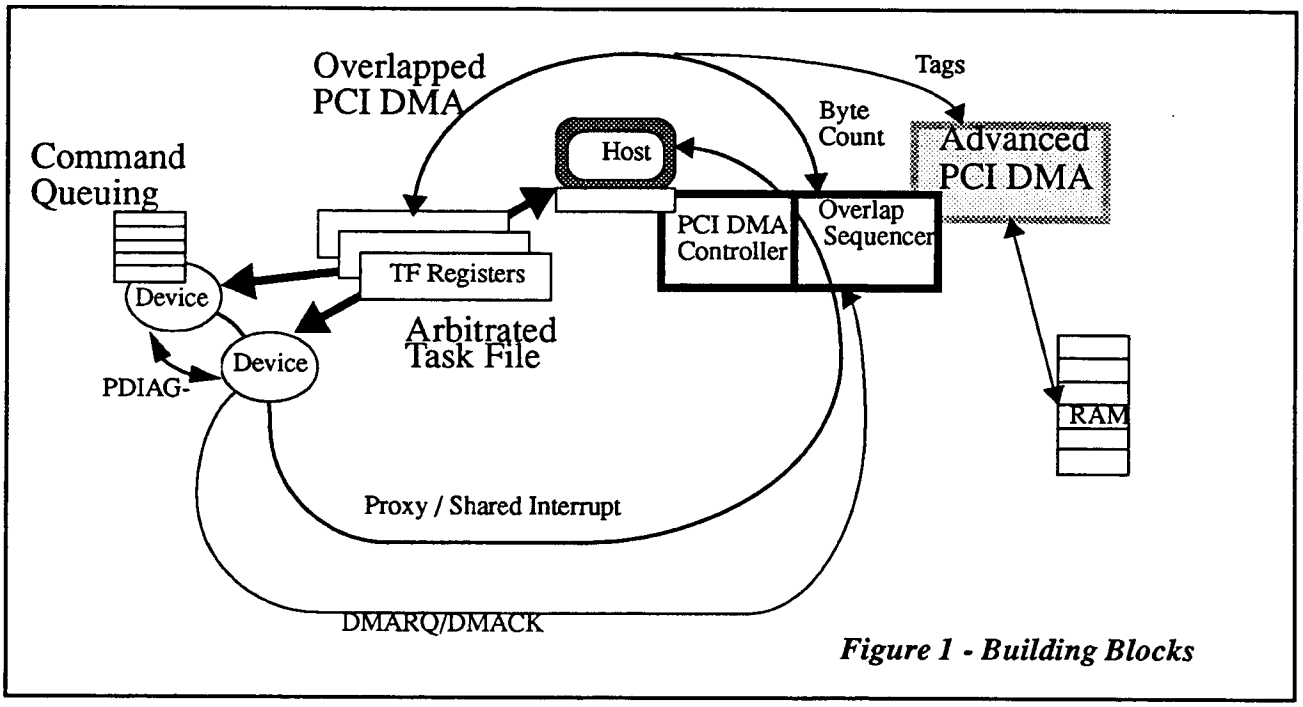

Figure 1 - Building Blocks

4.5 Capabilities

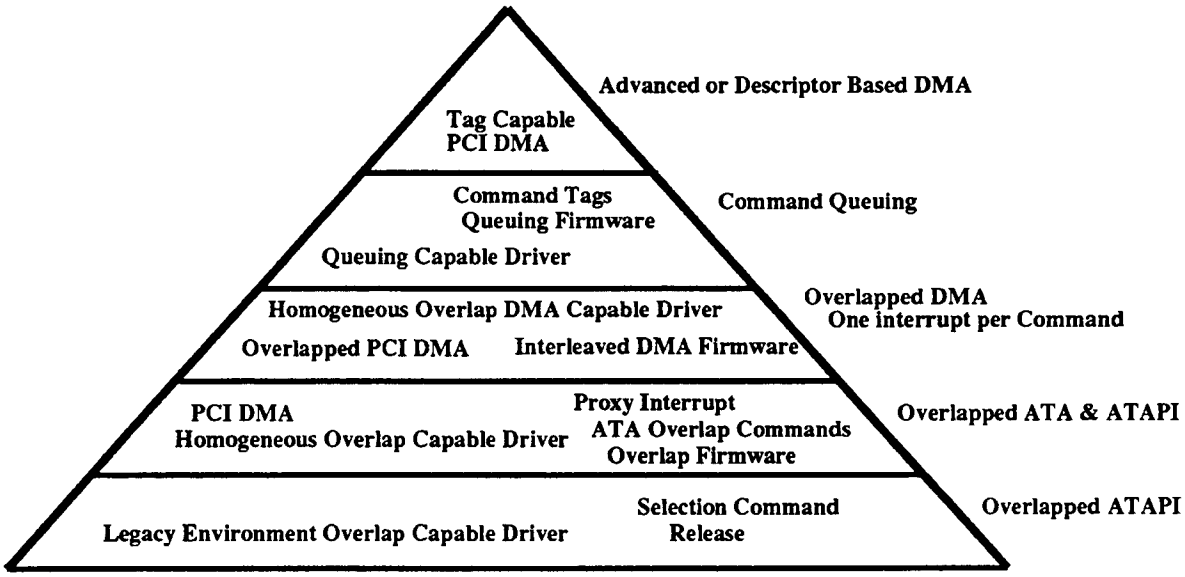

Figure 2 - Pyramid of Capabilities

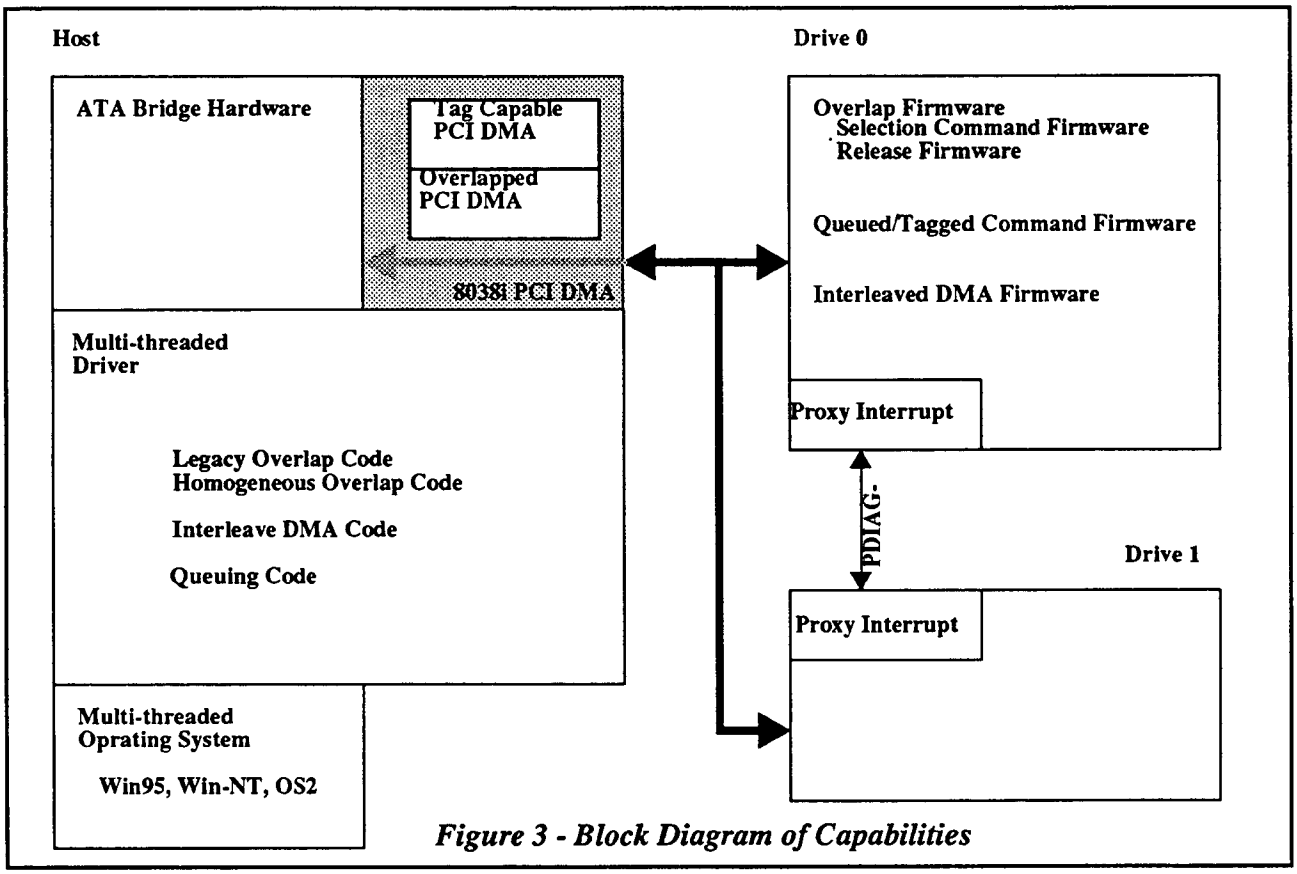

Figure 3 - Block Diagram of Capabilities

4.5.1 Mixed Legacy Environment Overlap

- Allows a CD-ROM Drive to be attached to Primary Cable with no Performance Penalty.
- Uses existing Host & Drive (ATAPI) Hardware (No hardware changes needed.)
- ATAPI Drive Releases the Task File Ownership after acceptance of an ATAPI command.
- Overlap Mode is enabled on each command via the ATAPI Features Register.
- Overlapped Commands are issued to an ATA (Legacy) Drive while an ATAPI Command is still processing.
- Interrupts are generated from the selected device only. Thus the Driver must always select the Overlap capable device when there is no active command to a Legacy Device.
- Device uses Interrupt & SERVICE Status to gain Host's attention.
- SERVICE Status set when any service is needed by the Device.
- Driver uses the A2h (Select) Command to give control of the Task File Registers back to the Device after an Interrupt and Sensing the SERVICE status bit.
- The Interrupt Reason RELEASE Status bit is used to indicate a Release Interrupt.
- Use existing PCI DMA.
- When DMA used, the Drive Interrupts when DMA is complete, only when there is still more data to be transferred (That command still has more data to transfer).
- When DMA used, Host (Driver) uses Byte Count from the Selection Command (A2h) results, to program the DMA Controller.

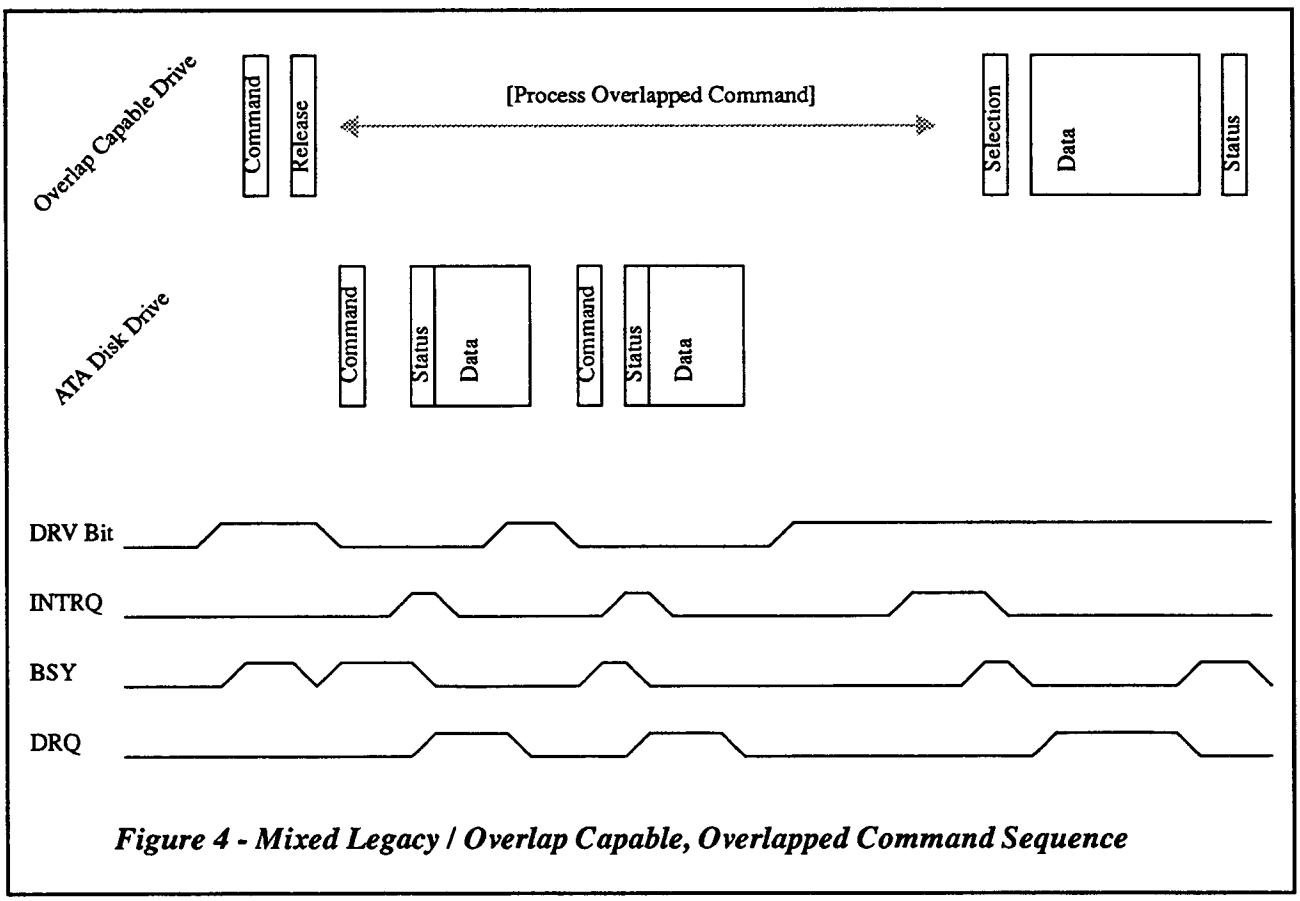

Figure 4 - Mixed Legacy / Overlap Capable, Overlapped Command Sequence

4.5.2 Homogeneous Overlap

- Two new ATA commands for Read and Write Overlapped or Queued.
- DMA or PIO specified in the ATA Feature Register.
- Device Releases Task File after acceptance of an Overlapped command.
- Task File arbitration performed by the Host (Driver).
- Proxy Interrupts used to signal Service to Host.
- Device prevents interrupts while BSY or DRQ is set.
- Both Devices on the Cable must support the Proxy Interrupt for it to be used.
- Limited Overlap can still be used without Proxy Interrupts.
- Proxy Interrupt function is enabled by the Driver via the SET FEATURES Command.

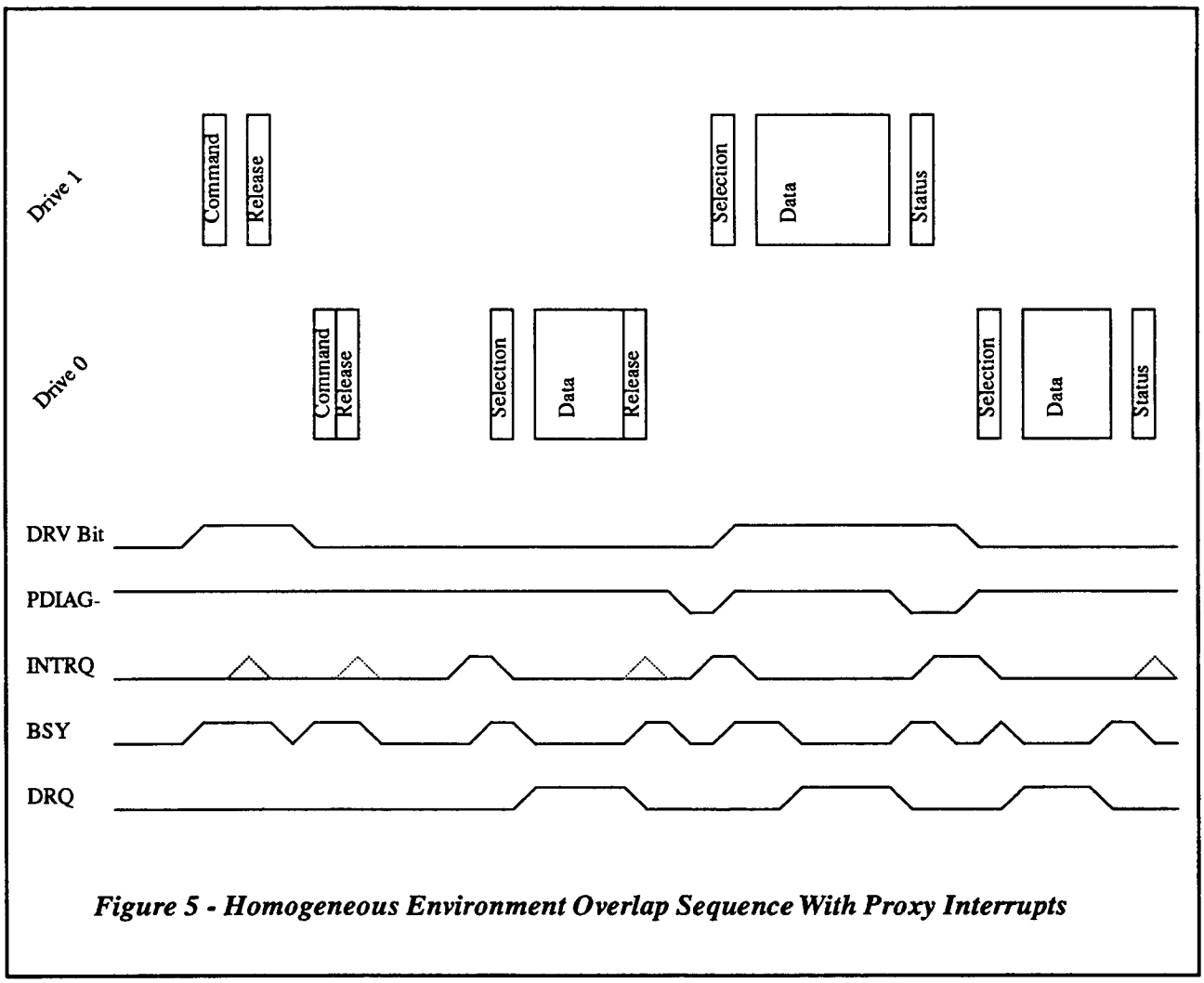

Figure 5 - Homogeneous Environment Overlap Sequence With Proxy Interrupts

4.5.3 PCI DMA Overlap Capability

- Intercepts the Interrupt from the ATA/ATAPI Device.
- New Status bit in the ATA Status Register to indicate DMA ready.
- Sequencer selects each Drive, senses DMA Ready & Service status bits.
- Arbitrates and Selects a Drive by issuing A2 command.
- Uses an Interlock to prevent Host (Driver) and Sequencer collisions.
- Blocks Interrupts while processing INTRQ or DMA transfers.
- Interrupts Host for unknown interrupt reasons.
- Host (Driver) Performs same function on systems that do not have the Hardware support for the PCI DMA Overlap.
- No change in the existing Drive DMA or DMARQ/DMACK logic.
- Bridge logic must include control registers for each of the two devices.
- Sequencer in bridge logic reads the Task File Status, Byte Count and Tag registers.

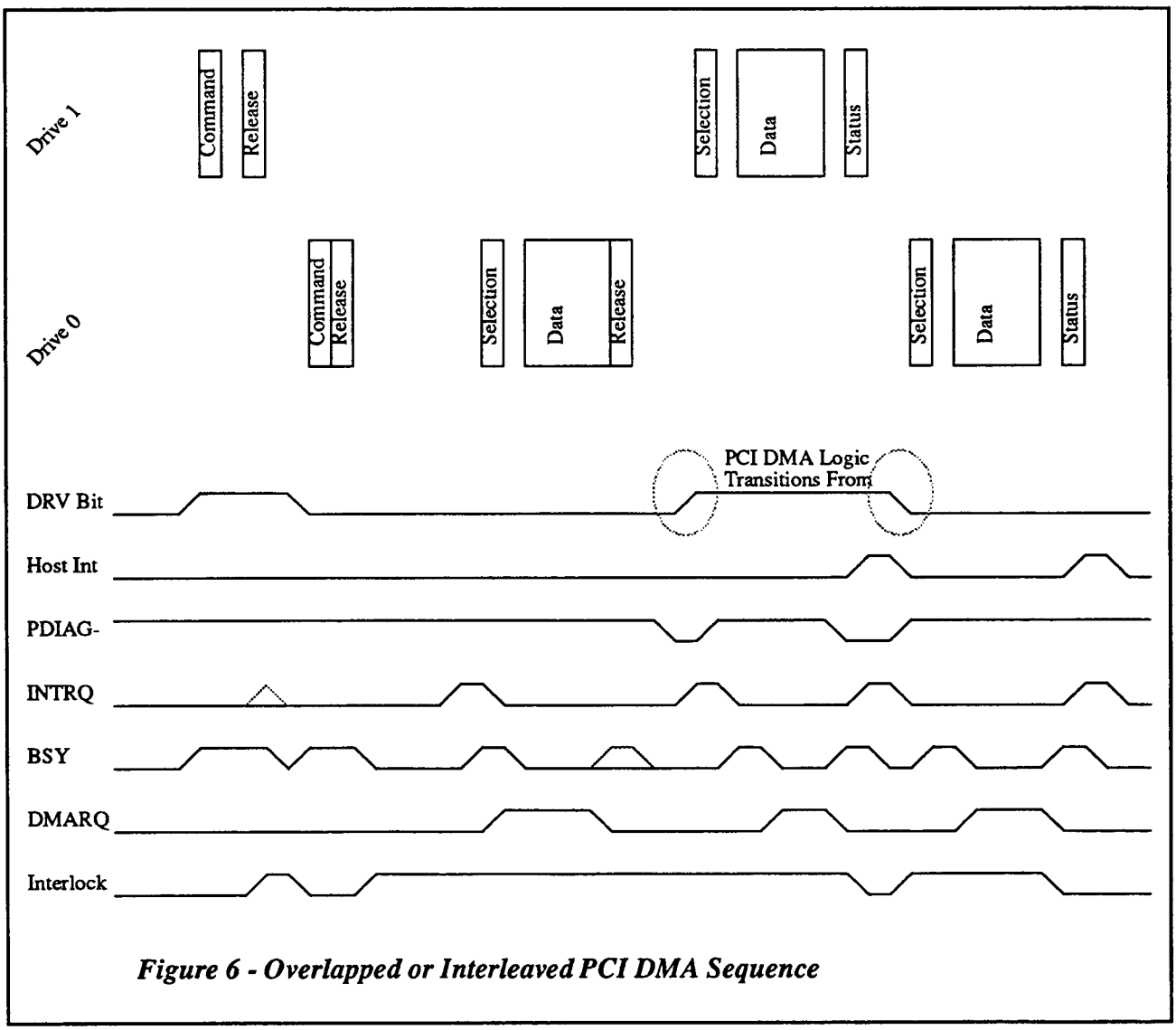

Figure 6 - Overlapped or Interleaved PCI DMA Sequence

4.5.4 Command Queuing

- Tags are used to identify each command
- ATAPI uses the Tag register to communicate the Tag To and From the Device
- ATA Drives use the ATA Feature Register to send the Tag on a command, then use the ATAPI Tag register when asking for service.
- For Both ATAPI and ATA the Tag is placed in the ATAPI Tag register after the A2h Command is issued.
- Simple Queue Tags only are implemented.
- Errors abort all commands in the Queue.
- The Overlap Capable PCI Bus Master DMA logic should read the Tag Register after the SELECT Command has been issued and if the Tag is not the same as the previous, then Interrupt the Host for processing. This should be the basic functionality, even if command queuing is not supported by the PCI DMA Logic.

4.5.5 Advanced or Descriptor Based DMA

- Layered on Overlapped PCI DMA & Homogeneous Overlap capability.
- Advanced DMA Controller reads Tag from Task File.
- Uses a memory block with one DMA Control entry for each Tag and Device.

5.0 Paths

Although what has been discussed thus far are the goals, they will not be implemented all at once. The goals will be met in groups. These groups represent a phased implementation approach, with less complicated but most beneficial features implemented first. Other more complicated features or those features that require host hardware modification will be phased in over a longer time frame.

Although some features will be phased in, the actual definitions for all the enhanced functions will be standardized at the same time.

A real issue is silicon development cycles, with drive vendors wanting to get this worked out soon as there are new chips on the operating table and we need closure.

5.1 Phase 1: Mixed Legacy Overlap Environment

These are the first improvements that will be implemented. They do not require any hardware changes to the Host or the Overlap Device. This capability will be the foundation for all the other overlap and Queuing capabilities. This phase will allow the OEMs to place one ATA Hard Drive and one ATAPI CDROM on a single ATA Cable, thus reducing the overall cost of the system by at least $0.50 or more.

One Fast Device and one slower (ATAPI) Device on one ATA Cable.
Doesn't require host LSI changes.
The Host (Driver) arbitrates the DMA hardware usage by the ATA & ATAPI Drives.
Requires OS Driver changes.
Requires minor ATAPI Device Firmware changes.
Only the ATAPI Device needs to be capable of the Overlap function.

5.2 Phase 2: Homogeneous Overlap Environment & Reduced Host Interrupts for DMA

This is where most of the performance improvements will be created. This phase supports both ATA Disk drives and ATAPI CDROM/Tape. The capabilities added here will form the foundation for Command Queuing and more advanced DMA capabilities. During this phase there will be some skewing in the use of Host Silicon and Devices that can implement the overlap capabilities. Although the full improvement in performance will not be obtained when this occurs, some improvement is still expected.

Mixing of Legacy and Overlap Capable Devices possible.
Mixing of Non-enhanced Hosts and Enhanced Drives keeps most of the performance gains.
Doesn't require host LSI changes when using existing PCI DMA or PIO.
Reduced Host Interrupts requires new PCI DMA Bridge logic.
The Host arbitrates the DMA hardware usage by the ATA and ATAPI Devices.
Requires the use of Proxy Interrupts in the Devices.
Requires OS Driver changes (Should have already been done in the first phase.)
Requires minor ATA Drive Firmware changes (ATAPI Drive changes occurred in phase one.)

5.3 Phase 3: Command Queuing

In this last phase, Queuing of commands in the Faster ATA Disks is implemented. In addition, optional PCI DMA bridge logic could be implemented that will allow the Queued commands to be processed out of order without interrupting the Host Driver when the Tag changes.

All Drives must be Shared Interrupt capable, no mixing allowed.
Optional DMA channel handles Tag and Drive changes automatically (No Driver intervention).
Requires Host LSI changes if Automated DMA is used.

Requires OS driver changes.
Requires Drive Firmware changes.

5.4 OS Improvements

Although not part of any of the above "Phases" the Operating System and its drivers are an integral part of any performance improvements that can be achieved. Once the definition for the Overlap capabilities is complete the next releases should implement ALL the new capabilities. There are a number of areas that Western Digital believes are important for the Operating Systems to consider:

- Get rid of the monolithic ATA disk/CD-ROM driver and move fully to the NT like layered I/O model.
- Support disk, CD-ROM, and tape TSDs.
- Rename SCSI Manager and call it I/O Manager.
- Run ATA and ATAPI hardware via mini-ports.
- Allow the ASPI interface to communicate with the ATA and ATAPI mini-ports so that CDHD, CR-R, and CD changer can be supported by 3rd party applications.
- Support PCI Bus Master DMA.
- Support for Overlapped ATA channel.
- Support for Command Queuing.

6.0 Key Issues

Before launching into technical discussion of various solutions, it is necessary to provide a foundation of understanding for the reader. This can be thought of as a list of problems, but even more than that, a tutorial.

6.1 Tutorial and Problem Definition

The ATA interface of today has evolved from the simple Winchester controllers, where there was only one set of registers and up to two ST506 or ESDI drives. In this environment there was only one controller for the peripherals and as such the interface was created as "Single Threaded."

Here we find the first of the problem areas for enhancing the interface to provide some level of "Multi-threading" capabilities. ATA controllers use a protocol to talk with the host, that provides a simple semaphore for locking access to the ATA Interface Control Registers (Task File). This is made up of the Busy (BSY) signal and the Data Request (DRQ) signals. These signals are used to control "Ownership" of some of the Task File Registers. When either BSY or DRQ is set only the peripheral (that is selected) is allowed to Read and/or Write to the controlled registers.

6.1.1 Task File ownership

Logic conventions are: A = signal asserted, N = signal negated, x = does not matter which it is.
Dark Gray are registers where ownership is controlled by BSY & DRQ.
Light Gray are Registers that are not defined for use by ATA.

Addresses					Functions	
CS1FX	CS3FX	DA2	DA1	DA0	Read (DIOR-)	Write (DIOW-)
A	N	0	0	0	Data	
A	N	0	0	1	Error Register	Features
A	N	0	1	0	ATAPI Interrupt Reason / Sector Count	
A	N	0	1	1	Sector Number	
A	N	1	0	0	ATAPI Byte Count LSB / Cylinder Low	
A	N	1	0	1	ATAPI Byte Count MSB / Cylinder High	
A	N	1	1	0	Drive Select	
A	N	1	1	1	Status	Command
N	A	0	0	0	Floppy A Status	Unused
N	A	0	0	1	Floppy B Status	Unused
N	A	0	1	0	Unused	Floppy Digital Output
N	A	0	1	1	Floppy ID / Tape Control	RESERVED
N	A	1	0	0	Floppy Controller Status	RESERVED
N	A	1	0	1	Floppy Data Register	
N	A	1	1	0	Alternate Status	Device Control
N	A	1	1	1	Change / Drive Address	Unused

Table 1 - Registers Controlled by BSY & DRQ

As you can see it is impossible to issue a new command for a number of reasons:

Can't write new command parameters into the Task File while BSY or DRQ is set.
Either BSY or DRQ is always set from the time the command is issued until it is completed.

Changing the DRV bit will cause commands to be aborted in some existing ATA Hard Drives.

6.1.2 Sharing of ATA Hardware Signals

Provided that the problems of sending overlapped commands is solved, a new set of issues is exposed. These center around the sharing of signals from the ATA Device to the Host. These signals are used to Generate an Interrupt (INTRQ) and Control DMA operations (DMARQ, DMACK).

6.1.2.1 Sharing INTRQ

For example with one or more commands being processed in more than one device at the same time, when the drive needs to signal to the host that data or status is available, it uses INTRQ. This signal in ATA devices today is driven high when an interrupt is desired. In addition it is only driven by the device that is selected via the DRV bit. Thus there is no way, currently, to allow each device to independently generate an interrupt.

There are several basic approaches to solve this problem. The INTRQ Signal could be inverted and each peripheral can then drive it low to signal an interrupt, or the devices can decide between themselves which will be allowed to drive the signal. A hybrid of these approaches is to continue to use the existing INTRQ hardware, but allow the selected device to generate an interrupt for the non-selected device. This proposal will use this hybrid as it is believed that the arbitration by the devices themselves is far too complicated for any benefit that is derived. This would also force significant hardware changes in the devices. Note that the other proposals would make use of DASP and PDIAG- for device to device arbitration, and this would then prohibit the use of these signals for future enhancements (e.g. Parity.)

The request for the other device to generate an interrupt will be referred to as a "Proxy Interrupt" in this proposal. To allow the selected device to generate an interrupt for the non-selected device (Proxy), there must be some communication from the non-selected device to the selected device. This proposal will make use of the PDIAG- signal. This signal is not used after the Reset and Diagnostics have been completed and is already used to communicate between the devices. The PDIAG- is an inverted signal, and thus also is perfect for bidirectional communication. If during transitions the signal is driven by both the devices then there will be no damage or loss of protocol. The PDIAG- signal is currently driven by firmware in the devices and is thus available for a new protocol. It is assumed that only in the first generations of devices will this function be performed by firmware. Firmware delays will cause reduced performance compared with a hardware implementation for passing the PDIAG- through to the INTRQ signal.

Another advantage of the proxy interrupt is that the device that is currently selected has intimate knowledge of the current Task File state. It can thus only generate an interrupt for the non-selected device when the registers are idle. Thus if a data transfer or interrupt request or command is still being processed, the selected device will ignore any proxy interrupt request.

6.1.2.2 Sharing DMARQ & DMACK

When it comes to the DMA control signals the problem again can be solved in one of two ways. Unlike the INTRQ signal the DMARQ can not just be driven by each of the devices at the same time. Only one device at a time must use the signal to communicate with the DMA logic on the mother board. In addition the DMA logic must also be kept in sync with the specific location in a specific command from a specific device. The easiest way to do this of course is to use the Host computer to setup the DMA logic and select the device that will be using it.

A more complicated technique would be to program the Host DMA Logic with information on all the outstanding commands, then have the devices themselves arbitrate for the use of the DMA control signals, then provide a communication path from the Device to the Host DMA Logic to tell it which device, command and location within the command is currently using the control signals. Western Digital has a real problem with this level of complexity. Adding all this logic to the host DMA will add significant cost and as such will never be "Main Stream." In addition this would not allow a phased approach to implementation, which Western Digital believes is important.

Thus the solution Western Digital proposes is to not change the existing DMA logic in the Devices, and allow either the Host Driver or the PCI DMA Overlap Logic to change the DRV bit.

This proposal is for the DMA operations to use a protocol similar to the current PIO protocol. In the current ATA DMA environment, the device assumes that it has complete control of the Host's DMA logic. Of course in an overlapped environment this is not the case. Thus when overlapped operation is enabled the device will not start DMA operations until the Select Command has been received. When the device wishes to transfer data via DMA, it requests the use of the DMA logic by asserting INTRQ (or PDIAG- when not selected) and waiting for the Select Command (A2h). Once the Select Command is received, the device can start the DMA by asserting DMARQ and following the normal ATA DMA protocol. The device must also place the number of bytes that will be transferred in the Byte Count Registers before BSY is cleared when processing the Select Command.

Once all the data that the device has available to transfer has been exhausted the device can either release the Task File registers by clearing BSY or not release the registers by keeping BSY set. The only allowable reason for keeping BSY set is to allow the drive to place the Completion Status into the Task File Registers and then generating an Interrupt. This removes the extra interrupt to regain the us of the Task File Registers.

6.1.3 Other DMA issues

An overall issue of DMA, is that of using the interface for any other operations while the DMA is in progress. DMA makes actual reads and writes to the Data Register of the device performing the DMA. Thus it would be impossible to issue new commands to the device that is performing active DMA. In this proposal the DMA as well as the arbitration is performed by the Overlapped PCI DMA Bridge Logic. Today when DMA is used the command is issued to the device, and no further interrupts are generated until the DMA is complete.

In this proposal the DMA operation will be changed to be much closer to that of the PIO protocol used today. This will add some extra interrupts if the Host does not support the Overlapped PCI DMA Bridge Logic, but will still allow DMA and overlap to be used. This it is believed will still provide significant performance improvements.

6.1.4 Other Problem areas with ATA

Although the Overlapped capability is one of the major areas that need to be addressed, there are other weaknesses as well.

Status is not provided after the data is transferred.
The data transfers are always 512 bytes or some fixed multiple.

6.2 Driver Issues

This section should contain any currently understood sequences that the drivers will be responsible for.

- When there will be more than one ATA Driver in the system some form of interlock (semaphore) will need to be employed. Currently there is no mechanism to lock access to the Task File Registers between Drivers.

7.0 Proposal Details

This proposal makes the attempt to combine discussions of the solutions into just a few categories. In each of these categories all the pertinent information related to the Host Driver, Host Silicon, Drive Firmware and Drive Silicon will be covered. Rather than just cover the device side implementation as is usual for X3T10, all the interrelated functionality needs to be discussed.

This proposal is broken up into four separate capabilities:

- Overlapped Operation.
- Proxy Interrupts.
- Command Queuing.
- Advanced or Descriptor Based DMA.

8.0 Overlapped Operation

This section will discuss the overall capability of "Overlap." This is the foundation for all the other capabilities. There are two very slightly different variations of the Overlap Capability for the Devices, one for ATA and another for ATAPI. In the ATAPI world there are plenty of registers available in the Task File for the Link Layer information. In the ATA world all the bits are dedicated to passing parametric information and as such none (too few) are available to extend the functionality. This proposal requires the Overlap Capability to be enabled on a command by command basis. Although this is strictly not needed for just the simple overlapping of commands, when the Tags are introduced later, this becomes more important. Rather than try to change the method of overlap between simple overlapped and queued commands, this proposal makes them identical with the exception of the use of the Tags when sending the command to the Device.

8.1 Task File Changes for Overlap

8.1.1 Overlap for ATA commands

The overlap is enabled by using two new ATA Commands, READ Tagged and WRITE Tagged commands only. All other commands can not be overlapped. There is no real reason for commands that do not access the media to be overlapped. These commands operate quickly and thus do not need overlapping. If overlapping was to be supported for ATA Commands, then there would need to be a Command by Command enable, so that there would be no backward compatibility issues when dropping back to an "Older Style" driver in sever error conditions. As there is no place in the Task File for an Enable bit, this approach of the two new commands was taken.

In addition the definition for the ATA Feature Register changes when the Overlap Commands are used.

Figure 7 - Changes to the ATA Feature Register (ATA Tag) for Overlapped Operation

D7	D6	D5	D4	D3	D2	D1	D0	
Tag Value						Reserved	*DMA*	Write Only

8.1.2 Overlap for ATAPI Commands

As has already been discussed, the overlap capability is enabled on a command by command basis. There exists in ATAPI a register dedicated to this function, the Features Register. Thus each command is changed into an overlapped style command via the Overlap Enable bit in the ATAPI Features Register. As this register is used for every ATAPI command, they can all can be overlapped.

Figure 8 - Changes to the ATAPI Feature Register for Overlapped Operation

D7	D6	D5	D4	D3	D2	D1	D0
Reserved						*Overlapped Command Enable*	DMA Enable

8.1.3 Status register for Overlapped Operation

Figure 9 - Changes to Status and Alternate Status Registers for Overlapped Operation

D7	D6	D5	D4	D3	D2	D1	D0	
BSY	DRDY	*DMA Ready*	*SERVICE*	DRQ	CORR	IDX	CHECK or ERROR	Read

Bit 7 BSY — Busy is set whenever the drive has access to the Command Block.

Bit 6 DRDY — Indicates that the drive is capable of responding to a command. This bit shall be cleared at power on. Devices that implement tagged commands shall clear this bit when they are not ready to accept another host command into their queue.

Bit 5 DMA Ready — ***This bit signals that the Device is ready to start a DMA data transfer. This is used to communicate to the Overlap Capable PCI DMA Logic that this service interrupt is going to transfer data via DMA.***

Note that this bit is used for Drive Fault (DF) when the Overlap operation is not enabled.

Bit 4 SERVICE — ***This bit signals that the Device is requesting service or interrupt. This bit will be set when the interrupt is requested and not cleared until the SELECTION command is issued.***

Note that this bit is used for the DSC function with the Overlap operation is not enabled.

Bit 3 DRQ — Data Request - Indicates that the device is ready to transfer a word or byte of data between the host and the drive. The information in the ATAPI Interrupt Reason will also be valid during a Packet Command when the DRQ is set.

Bit 2 CORR — Indicates if a Correctable Error occurred.

Bit 1 IDX — Index status. This bit is Vendor Specific.

Bit 0 CHECK or ERROR — Indicates that an error occurred during execution of the previous command.

8.1.4 Task File Register usage for ATAPI Devices implementing Overlap

Addresses	D7	D6	D5	D4	D3	D2	D1	D0	Register
1F0									Data
1F1	Sense Key				MCR	ABRT	EOM	ILI	Error Register
							Overlap	DMA	ATAPI Feature Register
1F2						*Release*	IO	CoD	Interrupt Reason Reg
1F3	*Tag*								Tag Register
1F4	Byte Count that will be used for PIO or for *DMA operations*								Byte Count Low
1F5									Byte Count High
1F6				DRV	Head				Drive Select
1F7	BSY	DRDY	*DMA*	*Service*	DRQ	CORR	IDX	ERR	Status
	0xA0 or 0xA2								Command
3F6	BSY	DRDY	*DMA*	*Service*	DRQ	CORR	IDX	Check	Alternate Status
						SRST	nIEN		Device Control
3F7		L		DEV	Head Selected				Change / Drive Address

Table 2 - ATAPI Overlapped Commands, Register Usage

8.2 Release

One of the capabilities that is the foundation for Overlapped operation is Release. There are three different forms of release used in this proposal, after the receipt of a Command, after transferring some data and after the receipt of the SELECT Command.

This proposal will allow the device to implement these release operations either in the Firmware or in Hardware. Each of these release points has its own complexities. For example before the Task File Registers can be released after the receipt of a command, the Command and parameter information must be saved. Once the release is performed the contents of the Task File Registers can no longer be used by the Device. Although this save would seem simple, adding a complete set of shadow registers is expensive and will not allow the device to perform command queueing. For command queuing there must be separate registers/memory locations for each of the commands that can be queued. Thus if the device supports 16 commands, then 16 back to back commands could be sent to the device by the host faster then the device could process them. The device could have only one set of shadow registers and only automatically release the Task File Registers when moving the standard set into the shadow set, but this would incur delays for release when the second command was loaded if the first had not yet been processed by the device. This would thus cause inconsistent delays until the Task File Registers were released.

In this proposal it is assumed that device will consistently unload the information in the Task File Registers. Although holding the BSY longer in some cases would most likely be acceptable from a system performance standpoint, forcing the driver to poll for varying lengths of time is not. This proposal forces the device to report the maximum length of time that the device will EVER require to unload and then Release the Task File Registers. Further to reduce the length of time that the Driver would have to poll for the Release, this proposal adds an Interrupt on Release Capability.

The Interrupt on Release capability is enabled by the Host Driver using a SET FEATURES Command. To assist the Driver in determining if the Interrupt should be enabled the IDENTIFY DRIVE Command returns the length in microseconds that the device will use to Release for both a Overlapped Command and the SELECT Command. The Driver can then make its own decision to enable the interrupt. Thus if the Device reports 1000 microseconds, the Driver could decide that it wants to poll and not enable the interrupt (Unlikely). This should remove the debate on how long should the driver have to poll.

The Release after the transfer of data must be performed by hardware for DMA operations and as such there is no Interrupt generated after the release when transferring data.

The Release after the SELECT Command can only occur after the parameters for the Interrupt are loaded into the Task File Register. Thus for a hardware implementation of this Release, there must exist a separate set of information (Shadow Registers) for these parameters e.g. Byte Count, Interrupt Reason, Status. Note that even for Command Queuing there needs to be only one set of Shadow Registers. The acceleration of the SELECTION Command will be very important to the overall system performance when using overlap. It is highly recommended that the time required to perform the Release after the SELECTION Command is less than 5µs.

8.3 Using the Select Command (A2h)

The Arbitration of the Task File Registers is performed by logic outside of the Devices attached to the ATA Cable. The basic premise of this proposal is that the Device releases the use of the Task File Registers when it is processing the command and no longer needs the registers. This of course makes it difficult to place the arguments for the Interrupt into the registers as the device no longer owns them. The Select command essentially hands the registers back to the device so that the correct parameters can be placed into them. These parameters include the Byte Count, Interrupt Reason and Command Tag values.

When an overlapped command requests service the Host Driver or PCI DMA Logic is responsible for determining which device should be serviced, and then issuing the Select. This causes the device to place information on the reason for the service into the Task File registers. Note that for both ATA and ATAPI devices the results of the Select Command are the same.

Addresses	Register	Contents
1F0	Data	
1F1	Error Register	If the Status indicates an Error then this is Valid
1F1	ATA Tag for Command	TAG for ATA Command. Only used when sending the command to the Device.
1F2	Interrupt Reason	Contains IO and CoD
1F3	Tag for Command	Contains the Tag for the command requiring Service
1F4	ATAPI Byte Count LSB	Number of bytes that need to be transferred, both for PIO or for DMA
1F5	ATAPI Byte Count MSB	
1F6	Drive Select	Same before and after "Select"
1F7	Status	DRQ along with IO, Cod and Release determine the reason for the Service Request
3F0	Floppy A Status	Unused
3F1	Floppy B Status	
3F2	Unused	
3F3	Floppy ID / Tape Control	
3F4	Floppy Controller Status	
3F5	Floppy Data Register	
3F6	Alternate Status	Same as Status register
3F7	Change / Drive Address	Same before and after "Select"

Table 3 - Registers after the Selection Command

8.4 Error Handling with Overlapped Commands

An issue can arise with because overlapped commands are enabled on a Command by Command basis. If an overlapped command is in progress and a non-overlapped command is then received, the Device must abort without any status any outstanding overlapped or queued command(s).

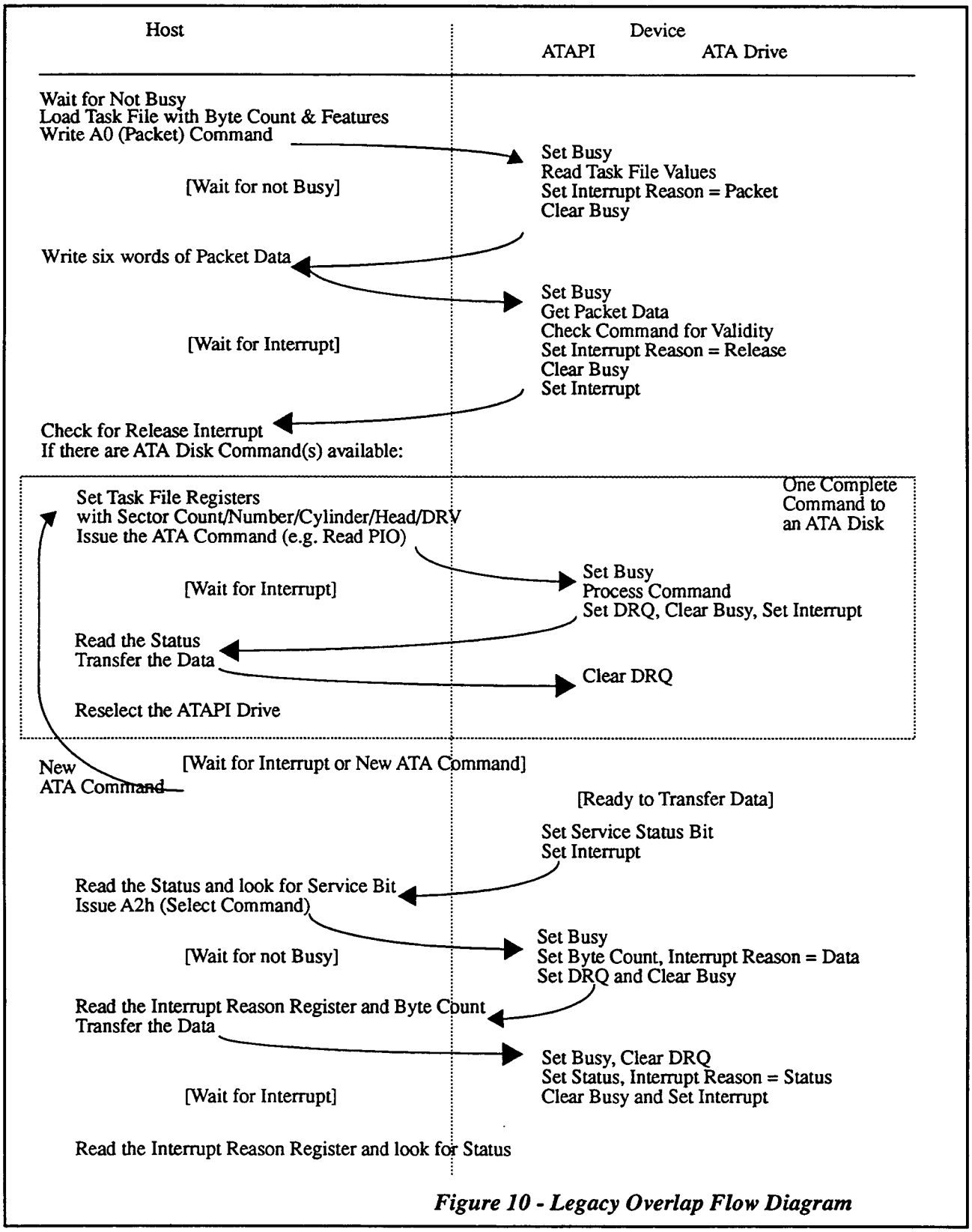

Figure 10 - Legacy Overlap Flow Diagram

9.0 Proxy Interrupts

The sharing of the Hardware ATA signal for requesting an interrupt (INTRQ) to the host is enabled by a SET FEATURES command. Note that this feature must be enabled for both devices on the cable. If either of the devices does not support Proxy, or if the set features command to enable the capability is aborted, the capability must be disabled in both devices.

When Proxy interrupts are enabled the SERVICE bit in the Status and Alternate Status registers will indicate that an interrupt was generated. This status bit will remain set until the condition that caused the interrupt is cleared. This could be Sending a Selection Command (A2h) or starting a data transfer for an ATA style command. Because this is a capability that is enabled once, it must be kept backward compatible with existing ATA commands and Drivers.

9.1 Rules regarding use of Proxy Interrupts

The PDIAG- signal is used to request a "Proxy" interrupt. The PDIAG- signal is defined in the ATA Standard as tristate. When the Proxy capability is enabled, each device will Tristate the PDIAG- signal. This signal has a pullup that will then drive the signal to +5 volts. This high level is the Deasserted or Idle level. Whenever the Device is NOT requesting a Proxy Interrupt, the PDIAG- shall be tristated. When a Proxy Interrupt is to be requested the PDIAG- signal will be enabled and driven to a low level. This is the "Asserted" state.

In this proposal the INTRQ signal continues to obey the ATA Standard definition. INTRQ will only be driven by the selected drive. The INTRQ signal will be driven to a High logic state when an Interrupt is being requested. The INTRQ signal, once driven to a high logic level will be again driven to a low logic level when the ATA STATUS REGISTER is read by the host.

1. When a selected device is not actively using the Task File e.g.
   Device not interrupting; and
   BSY is clear; and
   DRQ is clear; and
   Proxy Interrupts are enabled using set features command

Then the selected device shall:

   Assert INTRQ if PDIAG- is at a logic low (Proxy Interrupt is being requested)

2. When a selected device releases the Task File (Bsy & DRQ Cleared) the device shall discontinue requesting an interrupt, before BSY & DRQ are cleared, by tristating the devices PDIAG- driver and pulling PDIAG- high through the existing 10k pull up resistors on each device.

3. When a non-selected device, with overlap enabled, wishes to interrupt the host, the device shall set the SERVICE and optionally the DMA READY status bits, Enable PDIAG- and drive PDIAG- active low.

4. When a selected device receives a SELECT command, it shall clear any pending SERVICE request and discontinue requesting an interrupt (Tristate PDIAG-, allowing it to be pulled up to +5 volts)

5. The ATA EXECUTE DRIVE DIAGNOSTICS command shall cause Proxy Interrupts to be disabled. The host shall re-enable Proxy Interrupts following each diagnostic command using the enable Proxy Interrupts set features command.

6. When a Device that is driving PDIAG- low is selected, the Device shall tristate PDIAG- and Drive INTRQ high.

9.2 Why Proxy

Is this arbitration proposal a basic PROXY protocol? Yes, in the proposal the selected drive is still the only device which can issue an interrupt to the host. The non selected device drives PDIAG- to a logic low to request that the selected device interrupt the host for it.

What happens if the selected device is busy with a command or transferring data? When the selected device is busy it can block the interrupt request from the non-selected device. This is a crucial interlock which allows the selected device to prevent conflicts over the task file and DMA resources.

How long can the selected device block an interrupt request without causing the host to miss the requested interrupt? Like all level based ATA interrupts there is no time limit. The non-selected device will continue to drive PDIAG- to a logic low, to request an interrupt, until it is selected or reset by the host.

If I implement this arbitration protocol in F/W then there will probably be an overlap between the time my device is selected and the time I stop driving PDIAG-. How will we guarantee that we do not blow up drivers when both devices are driving PDIAG-. This proposal relies on using PDIAG- drivers which tristate with a pull-up to go to a logic high (NO INTRQ Request) and drive the signal to a logic low (Interrupt Request). This is a pseudo Open collector configuration and as such no driver conflicts exist.

Why don't we just use PDIAG- as a second interrupt to the host controller logic? The current proposal works without changes to the ATA host electrical interface which has been shipping for over five years. Many Mother board VLSI manufacturers have also stated that they are pin limited on embedded IDE implementations. Drive manufacturers have also noticed that features which can be implemented with hardware changes only in the drive find their way to the mass market much sooner than those which require changes to the host interface logic.

How does the arbitration proposal fit in with the ATA-3 discussion on WEAK Overlap, STRONG Overlap and Queuing? This arbitration proposal is designed to eliminate the difference between WEAK and STRONG overlapped command protocols. Western Digital believes that this proposal addresses the concerns on both WEAK and STRONG Overlapped commands. e.g. it provides for a F/W only implementation of strong Overlapped commands eliminating the need for a separate weak overlapped command protocol. This should simplify the task of device and driver development.

10.0 Enabling Enhanced Capabilities

The Set Features command is used to set some interface timing and protocol modes. These modes are set at Post by many BIOSes. The contents of the ATA Features Register indicates the function to be performed.

Table 4 - Contents of the Feature Register for Set Features Command

Bit Byte	7	6	5	4	3	2	1	0
0	Set (1)/ Clear (0) Feature	Feature Number						

Feature Register Contents	Set Feature Commands	Support
5Dh	Enable Interrupt for Release after the receipt of an Overlapped Command	Mandatory if the Device requires more than 50µs to process a Release operation
5Eh	Enable Interrupt after the processing of Select (A2) Command	
5Fh	Enable Proxy Interrupt Capability	Mandatory for an Overlap Capable Device
DDh	Disable Interrupt for Release after the receipt of an Overlapped Command	Mandatory if the Device requires more than 50µs to process a Release operation
DEh	Disable Interrupt after the processing of Select (A2) Command	
DFh	Disable Proxy Interrupt Capability	Mandatory for an Overlap Capable Device

Table 5 - Feature Number Description for Set Feature Command

11.0 Identifying Device's Capabilities

Table 6 - Identify Drive Data - Capabilities Word (49)

Bit Byte	7/15	6/14	5/13	4/12	3/11	2/10	1/9	0/8
0	Vendor Unique							
1	Inter-leaved DMA Sup-ported	Command Queuing Supported	Overlap Supported	Reserved for Pseudo DMA Sup-port	IORDY Supported	IORDY can be dis-abled	LBA Sup-ported	DMA Sup-ported

Bit 8 DMA Supported — This bit indicates that the Device supports the DMA mode of data transfer.

Bit 9 LBA Supported — Indicates that the Device supports the LBA form of addressing. The ATAPI Devices must set this bit to one.

Bit 10 IORDY can be disabled — Is used to indicate a device's ability to enable or disable the use of IORDY. If this bit is set to one, then the device does not support the changing of IORDY.

Bit 11 IORDY Supported — This is used to help determine whether a device supports IORDY. If this bit is set to one, then the device supports IORDY operation. If this bit is zero then the device may support IORDY (this insures backward compatibility.)

Bit 13 Overlap Operation Supported — This device supports the features in this proposal. Both the Release and Select protocol and the Proxy Interrupt capability must be supported to report that the Device supports Overlap Operation.

When the Overlap Operation is supported, words 71 and 72 must also contain valid information.

Bit 14 Command Queuing — This device supports Command Queuing and Command Tagging.

Bit 15 Interleaved DMA — This device supports the Interleaving of DMA data as specified in this proposal.

There is no need to Enable or Disable Command Queuing/Tagging. This capability is enabled by the use of the Queuing Command opcodes. Note that these commands can be used, even if Overlapped operations are not enabled. If READ or WRITE TAGGED is used then the Overlapped operation is automatically enabled. In this case the Overlap being enabled causes the SERVICE and DMA Ready status bit to be used.

11.1 Reporting of Overlap Timing Constraints

When Overlapped Operation is supported there are other informational words in the Identify Drive Data:

Length (Maximum) in μ seconds for the Release after Command Function.
Length (Maximum) in μ seconds for the Release after Select.

Thus the Driver can determine if it wants to Poll for the operation or to enable the interrupt so that it does not need to poll. Note that if the time reported for either Release Operation is greater than 50μs then interrupts must also be supported.

Table 7 - Identify Drive, Fields supported

Word	Description
0	General Configuration.
1	Cylinders
3	Heads
4	Number of unformatted bytes per track
5	Number of unformatted bytes per sector
6	Number of sectors per track
10-19	Serial Number
20	Buffer Type
21	Buffer Size
22	ECC bytes available
23-26	Firmware revision
27-46	Model Number
47	Multiple Sector Command, Sector Count
48	Double Word I/O
49	*Capabilities*
51	PIO Cycle Timing
52	DMA Cycle Timing
53 - 56	Current Cylinder/Heads/Sectors and Enable
57-58	Current Capacity
60-61	User Addressable Sectors
62	Singleword DMA mode
63	Multiword DMA mode
64	Enhanced PIO mode
65	Blind PIO minimum cycle time
66	Manufacturer's Recommended Multiword DMA Cycle Time
67	Minimum PIO Transfer Cycle Time without Flow Control
68	Minimum PIO Transfer Cycle Time with IORDY Flow Control
69-70	Reserved for Advanced PIO support
71	*Length (Maximum) in μ seconds for the Release after Command Function*
72	*Length (Maximum) in μ seconds for the Release after Select*
73-127	Reserved
128-159	Vendor Specific
160-255	Reserved

12.0 Overlapped PCI DMA

The overlapped PCI DMA Logic emulates in a simple form what the Host Driver would do when an interrupt is detected. Thus the ATA or ATAPI Device is unaware of any difference between a driver and the PCI DMA Logic.

To allow the PCI DMA Logic to more easily detect that DMA transfer is required, a new status bit has been added, called DMA Ready. This bit in conjunction with the Service bit and INTRQ signals to the PCI DMA Logic that it should start a DMA Operation for this device. Note that the DMA Logic must have previously been programmed for the transfer.

Use of the DMA Ready status and Service bits is enabled when overlapped Command are issued with the DMA enable bit set in the ATA or ATAPI Feature Registers.

Whenever the PCI DMA Logic senses a condition where it can not handle the interrupt (e.g. No DMA was setup for this device, or the DMA Ready and Service bits were not set) the Interrupt is passed through to the Driver. The Driver is then responsible for providing the same level of functionality as in the PCI DMA Logic. Note also that if the Host doesn't support the Overlapped DMA Capability, then the driver will provide that functionality.

To prevent the Host from accessing the ATA bus while any possible contention could occur, the PCI DMA Logic contains an Interlock mechanism. This is nothing more than a simple semaphore. When the Host commands the PCI DMA Logic to start overlapped operations, the Semaphore is seized by the DMA Logic. When the Driver wishes to gain access to the ATA Bus, it requests that the Semaphore is released and then waits until it is.

This might be a case for an addition of an Interrupt so that the driver won't have to poll.

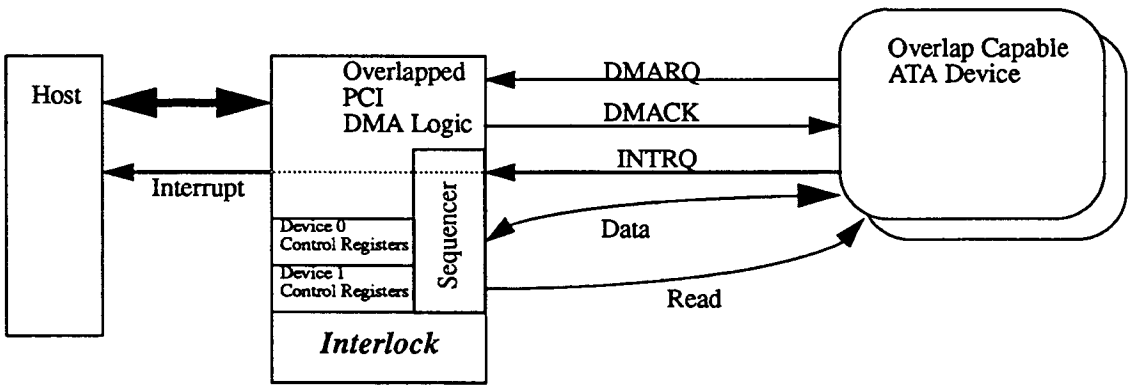

Figure 11 - Overlapped PCI DMA Block Diagram

12.1 Error Handling with Overlapped DMA

Error handling is no different than is used today.

13.0 IDE ATA Bus Master DMA

This section proposes a register level programming interface for a bus master ATA compatible (IDE) disk controller that directly moves data between IDE devices and main memory. By performing the IDE data transfer as a bus master, the Bus Master Device off-loads the CPU (no programmed IO for data transfer) and improves system performance in multi-tasking environments.

Controllers that implement this programming interface will benefit from bundled software shipped with major OS's limiting the amount of software development required to provide a complete product.

The master mode programming interface is an extension of the standard IDE programming model. This means that devices can always be dealt with using the standard IDE programming model, with the master mode functionality used when the appropriate driver and devices are present. Master operation is designed to work with any IDE device that support DMA transfers on the IDE bus. Devices that only work in PIO mode can be used through the standard IDE programming model.

The programming interface defines a simple scatter/gather mechanism allowing large transfer blocks to be scattered to or gathered from memory. This cuts down on the number of interrupts to and interactions with the CPU.

The interface defined here supports two IDE channels (primary and secondary). Individual controllers that support more than two channels will need to appear to software as multiple controllers if the standard drivers are to be used.

Master IDE controllers should default to Mode 0 Multiword DMA timings to ensure operation with DMA capable IDE devices without the need for controller-specific code to initialize controller-specific timing parameters.

13.1 Physical Region Descriptor Table

Before the controller starts a master transfer it is given a pointer to a Physical Region Descriptor Table. This table contains some number of Physical Region Descriptors (PRD) which describe areas of memory that are involved in the data transfer. The descriptor table must be aligned on a 4 byte boundary and the table cannot cross a 64K boundary in memory.

13.2 Physical Region Descriptor

The physical memory region to be transferred is described by a Physical Region Descriptor (PRD). The data transfer will proceed until all regions described by the PRDs in the table have been transferred.

Each Physical Region Descriptor entry is 8 bytes in length. The first 4 bytes specify the byte address of a physical memory region. The next two bytes specify the count of the region in bytes (64K byte limit per region). A value of zero in these two bytes indicates 64K. Bit 7 of the last byte indicates the end of the table; bus master operation terminates when the last descriptor has been retired.

Table 8 - Physical Region Descriptor Table Entry

31			0
Memory Region Physical Base Address [31:1]			0
EOT	Reserved	Byte Count [15:1]	0

NOTE: The memory region specified by the descriptor is further restricted such that the region cannot straddle a 64K

boundary. This means the byte count can be limited to 64K, and the incrementer for the current address register need only extend from bit [1] to bit [15]. Also, the total sum of the descriptor byte counts must be equal to, or greater than the size of the disk transfer request. If greater than, then the driver must terminate the Bus Master transaction (by resetting bit zero of the command register to zero) when the drive issues an interrupt to signal transfer completion.

13.3 Bus Master IDE Register Descriptions

The bus master IDE function uses 16 bytes of IO space. All bus master IDE IO space registers can be accessed as byte, word, or Dword quantities. The description of the 16 bytes of IO registers follows in Table 2-1:

Table 9 - PCI DMA IO Register Map

Offset from Base Address	Register	Register Access
00h	Bus Master IDE Command register Primary	R/W
01h	Device Specific	
02h	Bus Master IDE Status register Primary	RWC
03h	Device Specific	
04h-07h	Bus Master IDE PRD Table Address Primary	R/W
08h	Bus Master IDE Command register Secondary	R/W
09h	Device Specific	
0Ah	Bus Master IDE Status register Secondary	RWC
0Bh	Device Specific	
0Ch-0Fh	Bus Master IDE PRD Table Address Secondary	R/W

Figure 12 - PCI DMA IO Registers, BIt Definitions

Address	D7	D6	D5	D4	D3	D2	D1	D0	
Base +0 Command	*Soft Stop*	Reserved			Read/ Write	*Inter-leave Enable*	*Overlap Enable*	Start/Stop	RWC
Base +1	Reserved or Device Specific								R/W
Base +2 Status	Simplex	Drive 1 DMA Capable	Drive 0 DMA Capable	*Channel is Over-lap and Inter-leave Capable*	*Select (A2h) has been issued*	Interrupt	Error	Active and also the Inter-lock Sta-tus	RWC
Base +3	*Tag Register (0 when not using Tags)*								R/W
Base +4 - Base +7 PRD Table	A31	A30	A29	A28	A27	A26	A25	A24	R/W
	A23	A22	A21	A20	A19	A18	A17	A16	R/W
	A15	A14	A13	A12	A11	A10	A9	A8	R/W
	A7	A6	A5	A4	A3	Reserved			R/W

13.3.2 Bus Master IDE Status Register

Register Name: Bus Master IDE Status Register
Address Offset: Primary Channel: ..... Base + 02h
Secondary Channel: ..... Base + 0Ah
Default Value: 00h
Attribute: Read/Write Clear
Size: 8 bits

Table 11 - Bus Master IDE Status Register

Bit	Description
7	Simplex only: This read-only bit indicates whether or not both bus master channels (primary and secondary) can be operated at the same time. If the bit is a '0', then the channels operate independently and can be used at the same time. If the bit is a '1', then only one channel may be used at a time.
6	Drive 1 DMA Capable: This read/write bit is set by device dependent code (BIOS or device driver) to indicate that drive 1 for this channel is capable of DMA transfers, and that the controller has been initialized for optimum performance.
5	Drive 0 DMA Capable: This read/write bit is set by device dependent code (BIOS or device driver) to indicate that drive 0 for this channel is capable of DMA transfers, and that the controller has been initialized for optimum performance.
4	***Channel is Overlap Capable. This indicates that the Channel has the hardware necessary to Filter interrupts and provide DMA Signal Arbitration.***
3	***Select has been issued. This indicates that an IDE interrupt caused the DMA Overlap Sequencer to issue the Select Command, but the contents in the Tag Register did not match, thus the Driver must recover.***
2	Interrupt: This bit is set by the rising edge of the IDE interrupt line. This bit is cleared when a '1' is written to it by software. Software can use this bit to determine if an IDE device has asserted its interrupt line. When this bit is read as a one, all data transferred from the drive is visible in system memory.
1	Error: This bit is set when the controller encounters an error in transferring data to/from memory. The exact error condition is bus specific and can be determined in a bus specific manner. This bit is cleared when a '1' is written to it by software.
0	Bus Master IDE Active: This bit is set when the Start bit is written to the Command register. This bit is cleared when the last transfer for a region is performed, where EOT for that region is set in the region descriptor. It is also cleared when the Start bit is cleared in the Command register. When this bit is read as a zero, all data transferred from the drive during the previous bus master command is visible in system memory, unless the bus master command was aborted.

13.3.3 Descriptor Table Pointer Register

Register Name: Descriptor Table Pointer Register
Address Offset: Primary Channel: ..... Base + 04h
Secondary Channel: ..... Base + 0Ch
Default Value: 00000000h
Attribute: Read / Write
Size: 32 bits

Table 12 - Descriptor Table Pointer Register

Bit	Description
31:2	Base address of Descriptor table. Corresponds to A[31:2]
1:0	reserved

The Descriptor Table must be Dword aligned. The Descriptor Table must not cross a 64K boundary in memory.

13.4 Operation of the PCI Master Mode DMA

13.4.1 Standard Programming Sequence

To initiate a bus master transfer between memory and an IDE DMA slave device, the following steps are required:

1. Software prepares a PRD Table in system memory. Each PRD is 8 bytes long and consists of an address pointer to the starting address and the transfer count of the memory buffer to be transferred. In any given PRD Table, two consecutive PRDs are offset by 8-bytes and are aligned on a 4-byte boundary.
2. Software provides the starting address of the PRD Table by loading the PRD Table Pointer Register. The direction of the data transfer is specified by setting the Read/Write Control bit. Clear the Interrupt bit and Error bit in the Status register.
3. Software issues the appropriate DMA transfer command to the disk device.
4. Engage the bus master function by writing a '1' to the Start bit in the Bus Master IDE Command Register for the appropriate channel.
5. The controller transfers data to/from memory responding to DMA requests from the IDE device.
6. At the end of the transfer the IDE device signals an interrupt.
7. In response to the interrupt, software resets the Start/Stop bit in the command register. It then reads the controller status and then the drive status to determine if the transfer completed successfully.

13.4.2 Data Synchronization

When reading data from an IDE device, that data may be buffered by the IDE controller before using a master operation to move the data to memory. The IDE device driver in conjunction with the IDE controller is responsible for guaranteeing

that any buffered data is moved into memory before the data is used.

The IDE device driver is required to do a read of the controller Status register after receiving the IDE interrupt. If the Status register returns with the Interrupt bit set then the driver knows that the IDE device generated the interrupt (important for shared interrupts) and that any buffered data has been flushed to memory. If the Interrupt bit is not set then the IDE device did not generate the interrupt and the state of the data buffers is unknown.

When the IDE controller detects a rising edge on the IDE device interrupt line (INTRQ) it is required to:

- Flush all buffered data
- Set the Interrupt bit in the controller Status register
- Guarantee that a read to the controller Status register does not complete until all buffered data has been written to memory.

Another way to view this requirement is that the first read to the controller Status register in response to the IDE device interrupt must return with the Interrupt bit set and with the guarantee that all buffered data has been written to memory.

13.4.3 Status Bit Interpretation

"Table 11 - Status Register Interpretation" on page 39, describes how to interpret the Interrupt and Active bits in the Controller status register after a DMA transfer has been started.

Table 13 - Status Register Interpretation

Interrupt	Active	Description
0	1	DMA transfer is in progress. No interrupt has been generated by the IDE device.
1	0	The IDE device generated an interrupt. The controller exhausted the Physical Region Descriptors. This is the normal completion case where the size of the physical memory regions was equal to the IDE device transfer size.
1	1	The IDE device generated an interrupt. The controller has not reached the end of the physical memory regions. This is a valid completion case where the size of the physical memory regions was larger than the IDE device transfer size.
0	0	This bit combination signals an error condition. If the Error bit in the status register is set, then the controller has some problem transferring data to/from memory. Specifics of the error have to be determined using bus-specific information. If the Error bit is not set, then the PRD's specified a smaller size than the IDE transfer size

13.5 Error Conditions

IDE devices are sector based mass storage devices. The drivers handle errors on a sector by sector basis; either a sector is transferred successfully or it is not.

If the IDE DMA slave device never completes the transfer due to a hardware or software error, the Bus Master IDE command will eventually be stopped (by setting Command Start bit to zero) when the driver times out the disk transaction. Information in the IDE device registers will help isolate the cause of the problem.

If the controller encounters an error while doing the bus master transfers it will stop the transfer (i.e. reset the Active bit in the Command register) and set the ERROR bit in the Status register. The controller does not generate an interrupt when this happens. The device driver can use device specific information (e.g.; PCI Configuration Space Status register) to determine what caused the error.

Whenever a requested transfer does not complete properly, information in the IDE device registers (Sector Count) can be used to determine how much of the transfer was completed and to construct a new PRD table to complete the requested operation. In most cases the existing PRD table can be used to complete the operation.

13.6 PCI Specifics

Bus master IDE controllers built to attach to a PCI bus must have the following characteristics:

1. The Class Code in PCI configuration space indicates IDE device (top two bytes have the value 0x0101) and bit 7 of the Programming Interface register (offset 0x09) in PCI configuration space must be set to 1 to indicate that the device supports the Master IDE capability.
2. The control registers for the controller are allocated via the devices Base Address register at offset 0x20 in PCI configuration space.
3. In the controller Status register the Error bit will be set and the Active bit reset if any of the following conditions occur on the PCI bus while the controller is doing a master operation on the bus. The exact cause can be determined by examining the Configuration Space Status register.

Table 14 - PCI Error Status

Error Condition	Configuration Space Status bits
Target Abort	Anytime bit 12 of the Config Space Status register is set.
Master Abort	Anytime bit 13 of the Config Space Status register is set.
Data Parity Error	Anytime bit 8 of the Config Space Detected Status register is set.

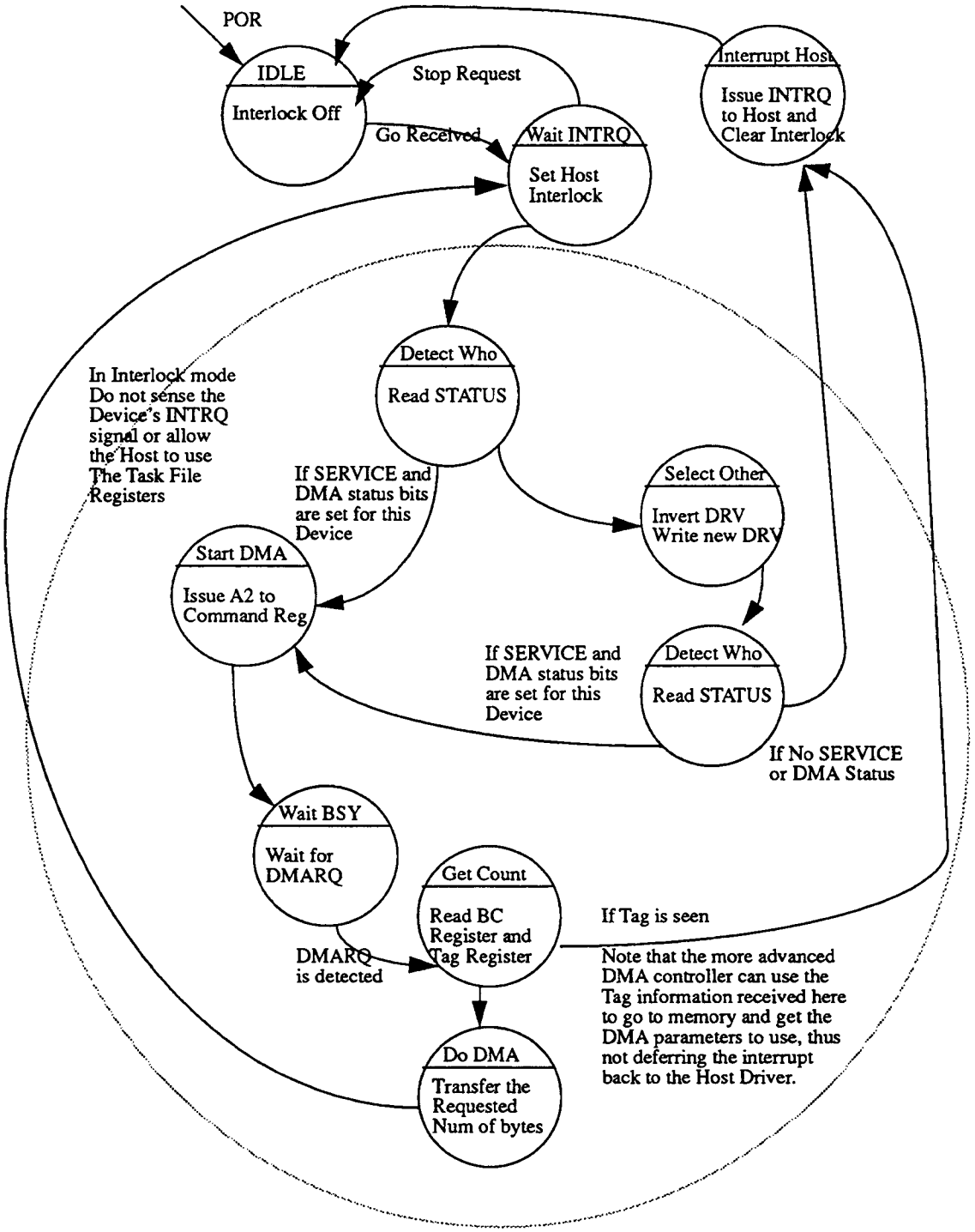

Figure 13 - State Diagram, Overlapped PCI DMA Support

14.0 Command Queuing / Tagging

This document is intended to define the additional functionality required to implement ATA tagged command queuing with minimal impact to backward compatibility with existing ATA BIOS and OS drivers. Adding support for tagged commands provides a simple, backward compatible, method for significantly enhancing the performance of ATA disk drives.

14.1 Performance of Command Queuing

The performance improvements achieved by adding tagged queuing are due primarily to the added ability of the device to receive multiple commands into its internal queue and then order the execution of these commands to optimize their rotational position on the media.

Because rotational delays are typically two to three times typical seek delays, the ability to allow the device to prioritize commands based on rotational priority can improve drive I/O performance by 50% to 150%.

14.2 Tagged Queuing for ATA

In order to support an ATA backward compatible tagging capability the function of the following bits within the task file have been redefined: DRDY, DSC and Index.

14.3 Tags in ATA

Although the stated goal is for the Tagging capability to be implemented the same for ATA as in ATAPI, the register definitions prevent it. In ATAPI the Tag register overlaps the ATA SECTOR NUMBER Register. Thus the tagging protocol will be the same for ATA and ATAPI, but the placement of the Tag will be slightly different.

For ATA the Tag will be placed into the ATA Features Registers. For ATAPI the TAG Register will be used.

14.4 ATA Tag Registers

Figure 14 - ATA Tag Register

D7	D6	D5	D4	D3	D2	D1	D0	
Tag Value						Reserved	DMA	Write Only

Bit 0 DMA — The DMA bit is used just like the least DMA bit in the ATAPI Feature register. If the DMA bit set (1) then DMA will be used for this command, otherwise normal PIO will be used.

Bits 7-2 TAG — This contains the Tag for the command. This allows for values from 0 - 63. All 64 tag values are legal.

14.5 ATA Style Tagged Commands

As the Tag is supplied in the ATA FEATURE Register, not all the commands could be queued. The constraint here is that the ATA FEATURE register is used for some commands to pass arguments. This forces the decision that not all commands can be queued. There are only two functions that will be able to benefit from queueing, Read and Write. Thus

there are two new commands to provide command queuing.

READ TAGGED 0xA6
WRITE TAGGED 0xA7

Addresses	D7	D6	D5	D4	D3	D2	D1	D0	Register
1F0									Data
1F1		UNC	MC	IDNF		ABRT	*DF*	AMNF	Error Register
	Tag							DMA	ATA Feature Register
1F2	Count								Sector Count
1F3	Sector								Sector Number
1F4	Cylinder								Cylinder Low
1F5									Cylinder High
1F6		L		DRV	Head				Drive Select
1F7	BSY	DRDY	*DMA*	*Service*	DRQ	CORR	IDX	ERR	Status
	0xA6 or 0xA7								Command
3F6	BSY	DRDY	*DMA*	*Service*	DRQ	CORR	IDX	ERR	Alternate Status
						SRST	nIEN		Device Control
3F7		L		DEV	Head Selected				Change / Drive Address

Table 15 - ATA Overlap Command Register Usage

Addresses	D7	D6	D5	D4	D3	D2	D1	D0	Register
1F0									Data
1F1		UNC	MC	IDNF		ABRT	*DF*	AMNF	Error Register
	Tag							DMA	ATA Feature Register
1F2							IO	CoD	Interrupt Reason Reg.
1F3	Tag								Tag Register
1F4	Byte Count that will be used for PIO or for DMA operations								Byte Count Low
1F5									Byte Count High
1F6		L		DRV	Head				Drive Select
1F7	BSY	DRDY	*DMA*	*Service*	DRQ	CORR	IDX	ERR	Status
	0xA6 or 0xA7								Command
3F6	BSY	DRDY	*DMA*	*Service*	DRQ	CORR	IDX	ERR	Alternate Status
						SRST	nIEN		Device Control
3F7		L		DEV	Head Selected				Change / Drive Address

Table 16 - ATA Overlap Command Register Usage after A2h (Select) Command

Tagged Commands use the following sequences for normal execution. The host system selects the device and monitors DRDY until the device is ready to accept another command (DRDY=1). The host then sets up the task file for the command including the Tag field. Once the Task file is configured the host writes to the command register to issue the tagged command.

The host may abort a specific tagged command in the peripheral's queue by issuing a new command to the peripheral with the tag used for the command to be aborted. Peripherals can abort tagged commands using the existing aborted command sequence with the additional requirement that the peripheral identify the aborted command with the command's tag.

14.6 Identifying Tag Capability

Devices supporting tagging shall return the tag supported bit, Bit 14 of Capabilities Word 49, set in the Identify device data returned by the Identify Device command.

14.7 Non Tagged Commands

Upon receipt of a non-tagged command the device shall terminate without status any commands in progress, clear all entries in the devices command queue and immediately process the non tagged command using the non tagged protocol.

Implementors Note: Mixing tagged and non tagged commands in the same command stream can create backward compatibility problems. As a rule the OS drivers supporting tagging should ensure that all commands issued by the tagging driver have completed before releasing control to other drivers in the system which may not be compatible with the tagging protocol.

14.8 Effect of Reset on Queued Commands

Upon receipt of any of the three ATA reset conditions; Power On Reset, assertion of the RESET signal or toggling the SRST bit in the device control register, the device shall terminate the current command without status and clear all entries from the devices tagged command queue.

In the case of SRST the device shall remain configured according to the rules governing the SET FEATURES command.

14.9 Errors and Command Queuing

All errors when Queuing is being used will cause ALL the commands that have yet to be processed to be aborted. The fact that the commands have been aborted is implied in the protocol and no specific status will be generated to indicate that these commands have been aborted. Note that this aborting of all commands on an error applies to ANY fatal or hard error that is detected and reported to the Host. This technique will greatly simplify the error recovery procedures that the Driver and Device will need to implement. This obviates the need for any contingent allegiance conditions or any specific command abort capabilities.

14.10 Advanced or Descriptor Based DMA

This capability is intended to allow the host to only field one interrupt per command. The Host's DMA controller must be capable of keeping information on ALL active commands for both ATA devices.

14.11 Device to Device Arbitration

There are two possible ways of handling the sharing of the DMA control signals. One would be for the devices to arbitrate for them, the other would be for the devices to signal to the DMA controller that they wish service, and then for the DMA controller to select one for DMA operations.

14.12 Signaling of Drive, Tag and Location in the Data Stream to Host DMA Logic

14.13 Error Handling for Advanced DMA

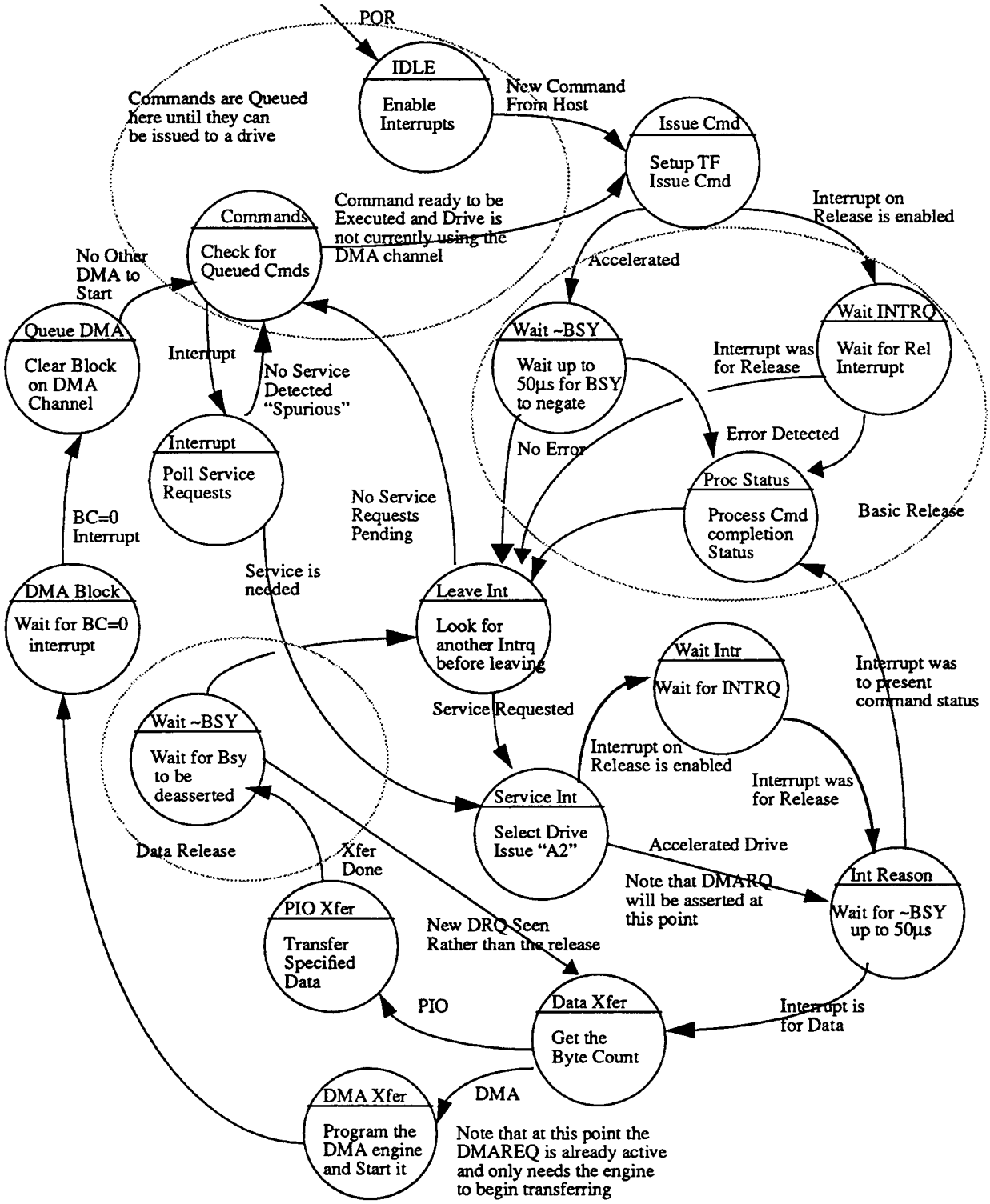

Figure 15 - Host State Diagram, Command Queuing & Shared Interrupts

14.14 Error Handling for Overlapped and Queued Operation

When an error on a command occurs that must report a "Fatal" condition to the host, all queued commands that have not yet been executed will be removed from the command queue and not executed. There will be no errors returned to the host for those commands. This is done to remove the Contingent Allegiance that is used in SCSI to handle getting the Status from the Request Sense. This would not be necessary for Queued ATA commands, but is being done to keep the interface and protocol consistent.

In overlapped operation there will be intermediate command status, as well as the final command completion status. The intermediate status is supplied to indicate if the command was accepted. If the command is not accepted, then there will be no further status supplied. The intermediate status is the status at the point that the device releases the Task File registers back to the host, prior to executing the command. Thus this status can only relate to the validity of the command and not any command execution.

15.0 Comparisons

15.1 Comparisons with Other Proposals

15.2 Shared Interrupt vs. Drive to Drive Arbitration

- Requires very little drive hardware changes.
- Not as complicated as Drive to Drive Arbitration.
- Allows PCI DMA controller to trap interrupt and perform arbitration or pass the interrupt through to the Host Driver when a function is not supported in hardware.
- Reduces communication from the drive to the DMA controller after arbitration.
- Allows some Overlapped commands in a mixed Legacy and Overlap Capable environment.

15.3 New Opcodes vs. using all Existing Opcodes in different "Modes"

- Using new Opcodes prevents any older driver from breaking.
- Allows some redefinition of the Task File Registers.
- Only two commands simplifies the Drive Firmware.
- Enables the Function on a command by command basis automatically.
- Does not break existing prefetch hardware.
- Does not break existing Drive Auto DRQ logic when using Queuing.
- Provides simple Drive Hardware Decode and Sequence logic.

15.4 PCI DMA hardware Arbitration vs. Drive to Drive Arbitration

- Only the Host Hardware changes.
- When the Host Hardware does not support the feature, the Host Driver can provide the capability transparently to the ATA Drive.
- Allows for various levels of performance without changing the Drive Hardware or Firmware.
- Advanced PCI DMA can be implemented without any Drive Hardware Changes.
- No potential race condition on the DMARQ line during arbitration.
- Can be used in mixed Legacy and Enhanced environment.

15.5 Weak and Strong overlap vs. Overlap and Accelerated Overlap

- (ISSUE)

16.0 Issues still unresolved

There are still several areas that remain unresolved.

- Semaphore for multiple IDE Drivers in a System
- SERVICE status is currently enabled when using an overlapped command and will revert back to the ATA definition when the command completes. This is cumbersome and should be handled better.
- There are cases where there is more data available when the DMA operation is complete. Thus it should be allowed for BSY not be cleared, and the operation continuing with another transfer. How can this be done?)
- Should there be an interrupt for Release after transferring data when using PIO. For example if the end of the PIO has been reached and the device then discovers more data to transfer can it then keep BSY high then raise DRQ to start a new data transfer?)
- Will the two 10k pullup resistors on PDIAG- be enough to pull the signal high quickly enough?)
- When requesting the PCI DMA engine to release the Interlock and stop operation should the Driver poll for the release or is this a case for an addition of an Interrupt?
- Need a discussion of the Data Interleaving terminology and function.
- Should the command be called Overlapped Read and Write and have the Queue operation enabled with a nonzero tag?)