Connecting IDE Drives

by
Tilmann Reh

Contents

1 Foreword

This text has been published in TCJ several years ago, in three parts. For your convenience, I concatenated them into one file again. Sometimes the text refers to other articles in TCJ, especially to a description of my ECB-bus based IDE interface. However, I think the information in this text contains all necessary information about IDE interfacing in general. 29. October 2001, Tilmann Reh

2 Remembering the Basics

Let us first have a short look at the drives we want to use. When discussing the different hard disk interfaces in my last article, I already pointed out that IDE drives of the AT-type, thus often called AT-bus drives (Bill Kibler calls them ATA drives, but this abbreviation is not the usual one, at least in Europe), are the ones with the best price/performance ratio one can get. This is even more the case now. So IDE drives still are the very first choice if you are looking for a good and cheap hard disk for your computer.

But what's special with those drives? I already mentioned that the IDE drive contains the complete hard disk controller. It is accessed with a system-bus interface compatible with the PC/AT (ISA) bus and offers control and data registers still compatible with the very first PC/AT hard disk controller (based on the WD 1010 controller chip). But even if those specifications come from something I don't like at all, why not use the low-price components for real computing (i.e., with a CPU280)?

Bringing the hard disk controller into the drive electronics offers some advantages. One of the main features is that you don't have a serial data stream with fixed bit rate between controller and drive. Thus, there's no need for conditioning the signals for the interface, and you can use any bit rate. As a result, the hard disk performance is limited by the drive technology, not by the interface's bit rate. This is one reason why today's drives are so much faster than the older ones. And technologies like Seagate's ZBR (Zone Bit Recording) are possible with hardware- independent interfaces.

There is another main feature of bringing the controller into the disk drive. Today's drives have very extensive checks for data security. They store error correction codes (ECC) together with the sector data and automatically correct single-bit errors, so the sector need not be re-read in those cases. Additionally, if a sector is found to be too unreliable, it is internally marked as bad and the data is mapped to a spare sector (usually there is one spare sector per track). All this is absolutely transparent to the user. So you now know the reason why today's intelligent drives don't have "defect lists" any more.

Since the PC's have such bad software (and hardware, too), there is another thing the integrated controller can do: translate virtual addressing information into physical. That means that the IDE drive is able to emulate another drive with different parameters (cylinder count, number of heads, and sectors per track). For the PC this is necessary because many PCs don't support drives with other than the historical 17 sectors per track, and many do not support free configuration of the drive parameters (only selection from a table is allowed). Also, some PCs mask off some bits of the cylinder number, since the first controller only had a 10-bit cylinder register – so nearly every IDE drive still supports an emulation mode with less than 1024 cylinders and 17 sectors per track.

3 The IDE Interface

As mentioned above, the IDE interface is almost completely identical with a subset of the PC/AT expansion bus, so the drive can be connected (almost) directly to that. The only things required externally are two select signals (I/O address decoding). This gives us some information about how the interface works. In a PC the drive is accessed directly by the CPU via I/O accesses to registers internal to the drive. The disk data is transferred via the 16-bit data bus, but for compatibility to the older systems (again!) only 8 bits are used for command and status information. Besides the data bus, there are the standard Intel-type data strobe signals (/IORD and /IOWR), a few address lines, and some special signals. The connector is a 40-pin header, not to be confused with the XT-type IDE interface connector, which is also a 40-pin header but needs somewhat different hardware and totally different software!

The IDE interface allows connection of two drives with one cable. The second drive (slave) is then chained to the first one (master). However, I heard about problems when trying to connect different drives from different manufacturers. And the capacities of today's drive are so high that a single drive will always be enough for an 8-bit personal computer system! So, I never tried this option.

To understand the interface in detail, let's have a closer look at the IDE interface connector and its signals:

1	/RES	2	GND
3	D7	4	D8
5	D6	6	D9
7	D5	8	D10
9	D4	10	D11
11	D3	12	D12
13	D2	14	D13
15	D1	16	D14
17	D0	18	D15
19	GND	20	No Pin
21	/IOCHRDY	22	GND
23	/IOWR	24	GND
25	/IORD	26	GND
27	/IOCHRDY	28	ALE
29	No Connection	30	GND
31	IRQ	32	/IO16
33	A1	34	/PDIAG
35	A0	36	A2
37	/CS0	38	/CS1
39	/ACT	40	GND

The signals of the IDE interface can be collected in several groups: The general control signals are /RES (Reset) and /PDIAG (Passed Diagnostics). The data bus consists of 16 data lines (D0..D15). The access control lines are three address lines (A0..A2), the select signals /CS0 and /CS1 (Chip Select 0/1), and the strobe signals /IORD and /IOWR (and eventually ALE, the address strobe). The remaining signals (IOCHRDY, IRQ, /ACT, /IO16) are status signals.

4 Now Let's Go Into Details.

The reset signal normally is active-low. However, I heard about drives with an active-high reset signal, but I never saw one (or read such specifications). The /PDIAG pin carries a bidirectional signal used for chaining two IDE drives (master/slave). It normally can also be left open.

The data bus carries the 16-bit data words to and from the host. However, when accessing the control and status registers of the IDE drives, only data bits 0 through 7 are used (8-bit transfer). The data bus lines are tri-state lines that may be connected directly to the host's data bus. However, to meet the host bus specs and to avoid noise problems caused by the interface cable, a bus driver IC should be used to decouple the IDE bus and the host bus.

The drive is accessed using the selection signals /CS0 and /CS1. This also has historical (compatibility) reasons. Together with the three address lines, there could be two- times-eight addresses being occupied by an IDE drive. However, while the main register set really has eight registers and is accessed with /CS0 active, the other set (with /CS1) has only two valid addresses. We will have a deeper look at all the registers later. The data transfer is always strobed by the timing signals /IORD and /IOWR, for reading and writing, respectively. The address strobe (ALE) is often unused in the drive; it should be pulled high for static address lines (non-multiplexed busses).

The status signals are not absolutely needed to use IDE drives. Some of these signals are not commonly delivered at all (for example, /IOCHRDY (I/O Channel Ready), which is a WAIT signal for the host when the drive is much slower than the host processor in terms of interface access times). The /IO16 line informs the host of 16-bit transfers. Since we already know that data transfers are always 16-bit and everything else is always 8-bit, this is redundant (however, needed in the PC/ATs for their ISA bus). Line /ACT (Active) is an output which can be used for driving a drive-busy LED. Line IRQ is an interrupt request line that goes active on some internal events (if enabled by software).

Most IDE drives contain some jumpers that allow some options to be selected. This normally includes at least master/slave selection. Sometimes the /ACT signal may also be jumpered as an output signalling the presence of a second (slave) drive. The default state of the jumpers normally need not be changed (single drive, no special situation).

All interface lines carry CMOS-TTL-compatible signal levels. However, some signals (IRQ, /PDIAG, /IO16, /ACT) are able to drive higher currents. Those details should be looked up in each drive's specifications (for example, the /ACT output sinks 20 mA on my Conner drive, more than enough for an LED).

Accessing the drive is done with the following sequence of operations: First, the address lines and the chip

selects must be set according to the desired register address. After some time (a minimum of 25 ns), /IORD or /IOWR is activated. This causes the data to appear on the data lines (when reading) or to be written to the drive (with the trailing edge of /IOWR, but there are setup and hold times to take care of). After a minimum of 80 ns, the strobe signal has to be removed. There are some more timing requirements, but these are the main ones.

The above timing details might differ from drive to drive. Always keep in mind that the IDE definition follows the PC/AT system expansion bus and that official standards were not specified until two years ago, when the IEEE finally defined some specifictions (which many PC manufacturers are not following).

Unfortunately, I found that the drives do not match their own specs in every detail. For example, I found that the address lines of my Conner drive (a CP-3044 with 42 MB) must be kept stable for much more than the specified setup time. In addition, the drive is very sensitive to spike noise on the address lines, even if the noise appears long before an access is initiated. I spent a great deal of time struggling with such unlucky details (fixing other people's bugs).

5 IDE Interface Registers

Now that we've covered the interface signals and their meaning and usage, let's look at the registers of the interface. We saw that there are eight addresses being accessed through /CS0 and two addresses through /CS1. The following is a list of all the internal registers of an IDE drive:

/CS0	/CS1	A2	A1	A0	Addr.	Read Function	Write Function
0	1	0	0	0	1F0	Data Register	Data Register
0	1	0	0	1	1F1	Error Register	(Write Precomp Reg.)
0	1	0	1	0	1F2	Sector Count	Sector Count
0	1	0	1	1	1F3	Sector Number	Sector Number
0	1	1	0	0	1F4	Cylinder Low	Cylinder Low
0	1	1	0	1	1F5	Cylinder High	Cylinder High
0	1	1	1	0	1F6	SDH Register	SDH Register
0	1	1	1	1	1F7	Status Register	Command Register
1	0	1	1	0	3F6	Alternate Status	Digital Output
1	0	1	1	1	3F7	Drive Address	Not Used

The above addresses are those used in the PC/AT. Of course they are dependent on the decoding of the chip-select signals. The registers accessed via /CS1 might differ depending on the manufacturer of the drive. As far as I know, they don't always follow the compatibility principle with the first hard disk controller of the PC/AT. The registers being accessed with /CS0 are also called the "Task File", so sometimes the IDE is also referenced to as "Task File Interface".

The error register can only be read. It contains valid information only if the error bit in the status register is set. Only five of the eight bits are used. They have the following meaning:

Bit 7: Bad block. This bit is set when the requested sector's ID contained a bad block mark (can be set when formatting the disk).
Bit 6: Uncorrectable data error. Set when the sector data can't be recreated (even with ECC).
Bit 4: Requested sector ID not found (wrong sector number).
Bit 2: Command was aborted due to drive status error or invalid command.
Bit 1: Track 0 has not been found when recalibrating.

The unused bits are always read as zero. However, I guess it's best not to rely on that!

The write precompensation register was previously used to set the starting cylinder for write precompensation (a slight shift of the serial data stream pulses to compensate for some magnetic effects on the disk surface). Since IDE drives handle all that internally, this function is not needed any more. Today, this register is often used as a parameter register for enabling or disabling look-ahead reading. We'll have a deeper look at that when talking about the various commands of IDE drives.

The sector-count register defines the number of sectors to be read or written with the next read/write command. A zero value causes 256 sectors to be processed, so the count varies from 1 to 256. This register is also used during drive initialization to specify the number of sectors per track (remember the emulation capability).

The sector-number register contains the starting sector number for any disk access. After a sector is processed, and after the command is completed, this register is updated. When an error occurs, this register contains the ID number of the erroneous sector. Normally, the sector numbers start with 1 and increase with each sector. However, by reformatting the disk, this order and the values may be changed.

The cylinder-low and cylinder-high registers contain the 10-bit cylinder number to be accessed. Since many drives have more than 1024 cylinders today, the cylinder-high register is often expanded to more than two bits.

Like the sector-number register, these registers are updated after command completion and after errors. They are also used during drive initialization as the number-of-cylinders parameter.

The SDH register is a special register serving several functions. SDH is an abbreviation for "Sector size, Drive and Head". The bits of this register are arranged as follows:

Bit 7:   Historical: Extension Bit. When zero, CRC data is appended to the sector's data fields. When set to one, no CRC data is appended. Since today's drives always use ECC error correction, this bit must always be set (no CRC).

Bit 6-5:   Sector Size. Since today's drives always have 512- byte sectors (unchangeable by the user) because PCs are not able to support other sizes, these bits must always be 0-1.

Bit 4:   Drive. This bit distinguishes between the two connected drives when using the master-slave chain. Single drives are always accessed with the drive bit set to zero.

Bit 3-0:   Head number. These four bits contain the head number (that is, the disk surface number) for all following accesses. Similar to the cylinder and sector number, these bits are updated by the drive. The head number field is also used for drive initialization to specify the number of heads.

The read-only status register contains eight single-bit flags. It is updated at the completion of each command. If the busy bit is active, no other bits are valid. The index bit is valid independent of the applied command. The bit flags are:

Bit 7:   Busy flag. When this flag is set, the task file registers must not be accessed due to internal operations.

Bit 6:   Drive ready. This bit is set when the drive is up to speed and ready to accept a command. When there is an error, this bit is not updated until the next read of the status register, so it can be used to determine the cause of the error.

Bit 5:   Drive write fault. Similar to "drive ready", this bit is not updated after an error.

Bit 4:   Drive seek complete. This bit is set when the actuator of the drive's head is on track. This bit also is updated similarly to "drive ready".

Bit 3:   Data request. This bit indicates that the drive is ready for a data transfer.

Bit 2:   Corrected data flag. Set when there was a correctable data error and the data has been corrected.

Bit 1:   Index. This bit is active once per disk revolution. May be used to determine rotational speed.

Bit 0:   Error flag. This bit is set whenever an error occurs. The other bits in the status register and the bits in the error register will then contain further information about the cause of the error.

The command register is used to pass commands to the drive. There are many commands, not always using all parameters in the task file. Command execution begins immediately after the command is written to this register. Since this article is already quite long, I will cover the commands, their parameters, and their usage in another article, probably in the next TCJ issue.

The alternate status register contains the same information as the status register in the task file. The only difference is that reading this register does not imply interrupt acknowledge to reset a pending interrupt (as the main status register does).

The digital output register contains only two valid data bits. Bit 2 is the software reset bit, which causes a drive reset when being set, and bit 1 is the interrupt enable flag.

The drive-address register simply loops back the drive select bit and head select bits of the currently selected drive. This information normally is of no use for the programmer or user.

6   Last Words

Now that we had a look at the IDE interface, we also see the physical limits of this interface definition. With a fully expanded cylinder-high register, we are able to address up to 65536 cylinders, with up to 16 heads and up to 256 sectors per track. This results in a maximum addressable drive capacity of 128 gigabytes. I think this should be enough for microcomputing!! However, even if the PC/AT BIOS limitations are encountered, we could address 1024 cylinders with 16 heads and 64 sectors per tracks, giving 512 megabytes maximum capacity. This is also not bad, at least for small (8-bit) computer systems, where complete application software packages require only about 100 kilobytes of disk space.

Next time I would like to talk about the applicable commands of IDE drives and give examples of how to write software that accesses those drives.

List of Abbreviations:

AT	Advanced Technology	Class of PC's
BIOS	Basic I/O System	Hardware-dependent part of OS
CMOS	Complementary Metal-Oxid-Silicon	Semiconductor technology
CRC	Cyclic Redundancy Check	Error detection code, see also ECC
ECB	???	European standard 8-bit system bus
ECC	Error Correction Code	Additional data for security
IDE	Integrated Drive Electronics	Intelligent hard disk interface
IEEE	Institute of Electrical and Electronics Engineers	
I/O	Input/Output	(self-explanatory)
ISA	Industry Standard Architecture	PC/AT expansion bus
LED	Light Emmitting Diode	Optoelectronical component
OS	Operating System	Software which makes computers usable
PC	Personal Computer	Synonym for the worst computer architecture
TTL	Transistor-Transistor-Logic	Digital component standard (74xx series)
XT	eXtended Technology	Class of PCs, previous to AT
ZBR	Zone Bit Recording	Variable Density Recording Method

7 Terminology

Using common terminology, I often simply refer to the "drive" when, in fact, I am thinking of the integrated controller of an IDE drive. However, when explicitly talking of an external controller like the WD1010, I always refer to the "controller".

8 Register Accessing

Let us first recall the Task File. It consists of the data register, a set of six parameter registers, and the command/status register. For those who don't have Part II lying nearby, here is a shortform:

Relative Address	Register	Abbr.
0	Data Register	D
1	Error Reg. / Write Precomp. Reg.	E / WP
2	Sector Count	SC
3	Sector Number	SN
4	Cylinder Low	C
5	Cylinder High	C
6	SDH (Sector Size, Drive, Head)	D,H
7	Status Reg. / Command Reg.	

Also remember that the data register is the only 16-bit register!

Every parameter register of the task file is freely accessible as long as there is no active command. Before loading the command register, all related parameter registers must contain the appropriate values. They may be loaded in any order. After the command register is loaded, the issued command is immediately started. The original WD1010 hard disk controller chip had a flag (bit 1 of the status register) which was set during execution. With IDE drives, the BUSY flag of the status register is simply set until the command execution is completed.

The WD1010 controller chip knew only 6 commands. However, some of the commands have option flags within them. To support additional features, today's drives have many more commands. The following is a list of common commands, options, and needed parameters, with the WD1010 commands marked by an asterisk and the manufacturer-dependent expansions marked with a plus sign:

Command	Type	7	6	5	4	3	2	1	0	Hex	Parameters
Recalibrate	*	0	0	0	1		(Rate)			10-1F	D
Read Sector	*	0	0	1	0	0	M	L	T	20-27	SC,SN,C,D,H
Write Sector	*	0	0	1	1	0	M	L	T	30-37	SC,SN,C,D,H
Scan ID / Verify	*	0	1	0	0	0	0	0	T	40,41	D,(SC,SN,C,H)
Write Format	*	0	1	0	1	0	0	0	0	50	C,D,H,(SC,SN)
Seek	*	0	1	1	1		(Rate)			70-7F	C,D,(H)
Exec Diagnostics		1	0	0	1	0	0	0	0	90	D
Set Drive Parameters		1	0	0	1	0	0	0	1	91	SC,(C),D,H
Read Multiple	+	1	1	0	0	0	1	0	0	C4	SC,SN,C,D,H
Write Multiple	+	1	1	0	0	0	1	0	1	C5	SC,SN,C,D,H
Set Multiple	+	1	1	0	0	0	1	1	0	C6	SC,D
Power Commands	+	1	1	1	0	0	x	x	x	E0-E6	SC,D
Read Sector Buffer		1	1	1	0	0	1	0	0	E4	D
Write Sector Buffer		1	1	1	0	1	0	0	0	E8	D
Identify Drive		1	1	1	0	1	1	0	0	EC	D
Cache On/Off	+	1	1	1	0	1	1	1	1	EF	D,WP
Power Save	+	1	1	1	1	1	x	x	x	F8-FD	?

Parameters in parentheses are needed with some drives and ignored by others (depending on the manufacturer and age). Any required parameters must be valid before a command is started.

Although most of the commands are manufacturer-dependent, this usually does not raise problems. For normal operation of the drive, only the WD1010's and few of the really common commands are needed. Now let's have a look at the options.

In the Restore and Seek commands, there is a four-bit rate field. This was originally intended to specify the step rate for head movements, with a zero value meaning 35 us per step and all other values representing counts of 0.5 ms per step (so that the range was from 0.5 to 7.5 ms). The hard disk controller had a memory for each drive's step rate, so the same value would be used for implied seeks later. But very soon, even with later ST-506 controller boards, this step-rate field became obsolete (due to handshake mechanisms between controller and drive). With today's IDE drives, the four lower bits of those commands are generally ignored.

The Read and Write commands originally had an option flag (M) for multi-sector (m/s) transfers. Today this flag is nonexistent. For doing m/s transfers, the sector-count register is simply set to the desired number of sectors to be processed.

But today's drives have another flag which originally wasn't there: the "Long" flag (L). When it is set, the ECC (error- correction) data is transferred after the sector's data field. I assume this was meant for error correction when there are too many errors for the drive to correct automatically. However, there is a very odd characteristic: the ECC data is transferred in bytes through the data register. This is the only case where the data register doesn't transfer word data! By the way, the number of ECC bytes differs from drive to drive, but can be read out using the Identify Drive command.

All generations of hard disk controllers and IDE drives support the last option flag for Read/Write commands, the Retry Flag (T). Normally, the drive retries to read/write a sector after non- fatal errors. When this option flag is set, automatic retries are disabled. The Scan ID command also supports this option.

9 The Commands

I will now explain the commands and their parameters in detail. To do this, besides drawing on my own experience in IDE interfacing, I collected detailed information and specifications from three different, independent sources. However, there still might be some slightly different drives or controllers out there. Please inform me if you encounter problems or differences with your disk.

One general feature of both the WD1010 controller and modern IDE drives is implied seeks. That means you don't have to explicitly move the read/write (r/w) heads to the desired cylinder before starting to read from or write to the disk. When the command is issued and the actual cylinder number doesn't match that of the cylinder registers, an implied seek is performed, transparent to the user. This applies to every command where it may be needed (Read, Write, Verify, Format).

9.1 Recalibrate (1xh):

This command moves the r/w heads of the selected drive from anywhere to cylinder 0. The controller waits for the drive to complete the seek before the task file is updated and the busy flag is reset. Upon successful completion of the command, the error register and the cylinder registers are set to zero, while SC, SN, and SDH remain unchanged.

9.2 Read Sectors (2xh):

This is probably the mostly used command. It will read from 1 to 256 sectors of disk data as specified in the SC register (with an SC value of 0 meaning 256 sectors to be read). The starting sector is defined by SN, C, and H in the task file.

When the task file contains invalid parameters, an error occurs. Otherwise, the r/w heads are moved to the requested cylinder if they are not already there (implied seek). Then the data of the starting sector is read into the sector buffer, and the DRQ (data request) bit in the status register is then set. This informs the host that the sector data can be read from the sector buffer. When this is completed, DRQ is reset and the drive is ready again.

For reading multiple sectors (SC>1), when the sector buffer is completely read and DRQ is reset, the busy flag is set again immediately, and the next sector's data is read into the sector buffer. When DRQ is set again, the next sector's data can be read from the buffer. This is repeated until the SC register value reaches zero (this register is decremented with every successfully read sector).

In any case, after successful completion of this command, the SC register contains a zero value, and the other registers in the task file are updated to contain the cylinder, head, and sector number of the last-read sector. If an error occurs, the task file will contain the parameters of the sector at which the error was detected.

For the Read Sectors command, two options are possible. The "M" option is valid only with the original WD1010 controller (thus with old AT's). When M was not set (0), the SC content was ignored, and exactly one sector was read. When M was set (1), the SC value was taken as the count of sectors to read. Today things are different. With modern IDE drives, if the M bit is set, an error occurs. So this bit must always be cleared! The other option ("L") is valid only with modern IDE drives. "L" stands for long and means that the additional ECC data, which the drive automatically puts after each sector, is transferred after the net data of each sector. When this option is set, the drive also does not check these ECC bytes, so it won't detect or correct errors. This provides a way to read a sector's (remaining) data even if it is nonrecoverably damaged. When using this option, remember that the ECC data is transferred as bytes through the word-wide data register!

9.3 Write Sectors (3xh):

The Write Sectors command is very similar to the Read Sectors command. Of course the data flow direction is different... This command will write up to 256 sectors of data to the disk. All parameters and options are similar to those of the Read Sectors command. After writing the command to the command register, the drive sets the DRQ flag, informing the host that the data can be written into the sector buffer. When all data has been transferred, DRQ is reset, and the drive starts writing the data buffer contents to the disk. The busy flag is set as long as the drive is physically writing to disk. The SC register is decremented, and, if not zero thereafter, DRQ is set again for the next sector. When using this command with the "L" option, the drive will use the ECC bytes delivered by the host and not generate any by itself. For the "M" option, the details described above apply.

9.4 Scan ID / Verify Sectors (4xh):

This is a very strange command. As far as I know, it is the only one that is totally incompatible between the old AT's hard disk controller and today's IDE drives. It would appear that this command was never used by common system implementation or application software...

For the WD1010 controller, this is the Scan ID command. It takes no parameters at all (except for the drive and head which originally had to be contained in a register external to the WD1010). When the command is started, the controller searches for the next ID field and reads the contents into the task file. This way the actual drive, head, cylinder, and sector size could be examined. The sector number was also transferred into the task file, so the sector numbering order could be figured out by repeating this command fast enough.

For the IDE drives, this is a completely different command: Verify Sectors. It is similar to the Read Sectors command except that no data is transferred to the host, and the "L" option is not supported. Thus, it needs all parameters in the task file. Up to 256 sectors of data will be read into the sector buffer, and their ECC bytes will be verified. The DRQ flag will never be set. The completion status of the command can be read from the status register.

It is interesting that both types of controller/drive support the retry option - so this is the only compatibility of this command.

9.5 Format Track (5xh):

Originally, this command was used to physically format an entire track of the hard disk, exactly as it's done when formatting floppy disks. The Format Track command is started similarly to the Write Sectors command: first the task file must be set up, then the command written to the command register. After that, the drive

responds by setting the DRQ flag. The host must then write data into the sector buffer until the DRQ flag is reset. After that, the command is executed.

For the format command, the sector buffer must contain special data. As with the index field array when formatting a floppy disk, it must contain valid sector ID's for every physical sector of the track that will be formatted, beginning at the start of the buffer. Each sector ID in the buffer consists of two bytes. The unused remainder of the buffer is ignored by the format command, but must also be written for the DRQ signal to disappear. The first byte of each sector ID is a flag byte. The WD1010 knew only two different values for this descriptor:

    00h   = good sector,
    80h   = bad sector.

Today's IDE drives offer two more descriptor values:

    40h   = assign sector to alternate location,
    20h   = unassign alternate location for this sector.

We'll look further at these values below.

The second byte of each ID is the sector-number byte. It contains the number by which the related sector is referenced later during normal r/w operation. The ID fields in the sector buffer are assigned to the physical sectors (created through formatting) in the order they are stored in the buffer. So it is possible to define an interleave factor by appropriate physical sector numbering. Here is an example:

```
Addr. 00 =  00 01  00 11  00 02  00 12
      08 =  00 03  00 13  00 04  00 14
      10 =  80 05  00 15  00 06  00 16
      etc.
```

Here we see the first 12 (of 32) ID words. The starting sector has number 1 (as usual). The interleave factor is two, since each sector appears two sectors after its logical predecessor. You can also see that sector number 5 (the 9th sector physically) is marked bad.

Due to surface errors on the hard disk, there are some positions where the media won't store magnetic information reliably enough (if at all). The defect list for a particular drive then shows the cylinder, head, and "BFI" (byte from index) value of the defect. People then had to calculate the bad-sector position and number from each of those BFI values. However, it is not commonly known that the relationship between the BFI value and the sector number depends not only on the sector size but also on the interleave factor and the starting sector number...

Again, things changed as the years went by... I already mentioned when introducing the features of modern IDE hard disks, that those drives don't have defect lists any more, due to the usage of internal spare sectors. For compatibility reasons, these drives still accept the Format Track command. However, most drives only simulate its execution – internally they don't really format any track. Modern drives are "hard-sectored" by the manufacturer, with the sector size unchangeable by the user. But by virtually formatting a track, one can assign new sector numbers (for example, starting with 0 instead of 1). However, the sector numbering order is often ignored. Because IDE drives commonly have built-in cache memories, the definition of an interleave factor would make no sense. So, the drive always uses the fixed sector ordering which gives maximum performance in combination with the cache.

To make things still more complicated, the Format Track command of IDE drives allows for the assignment of data sectors to the spare sectors and for the release of those assignments (look at the descriptor bytes above). All IDE drives have some spare sectors to which the data of defective sectors is automatically mapped. Normally, there is one spare sector per track, resulting in about 2-3% spare capacity. This is more than enough. When a sector appears too unreliable during normal operation, the drive simply marks that sector as bad internally and moves the data to the nearest free spare sector. As long as not all spare sectors are assigned, the user won't notice anything.

However, these assignments can also be done explicitly by use of the Format Track command. But it is strongly recommended not to do that! First, one will normally get no defect list for an individual IDE drive containing the BFI positions. Second, even if a sector which was assigned to one of the spares is marked good again, the related spare sector can not be used again! So with every unassignment of a spare sector, you loose that irretrievably.

So we come to this result: with standard (i.e., ST-506) drives and external controller (i.e., WD1010) it makes sense to format the drive in order to freshen the surface magnetism, to get a defined state (sector numbering and order), and to mark defect sectors as bad (so that the operating system can behave accordingly). With IDE drives, it's best to leave them just as they are coming from the factory!

9.6 Seek (7xh):

This command is used to move the r/w heads to a particular cylinder explicitly. For normal operation of the drive, it is usually not necessary, since all r/w commands perform implied seeks. However, this command can easily be used for benchmarks to determine the drive's seek times. With the WD1010 controller, the four lower bits of the command byte contain the step rate (described above). IDE drives simply ignore these four bits.

9.7 Execute Diagnostics (90h):

This command is common to all IDE drives but not available with the WD1010 controller. When issued, the drive performs an internal self-test. If the drive is a master drive, and a slave drive is connected to it, the master also waits a limited time for the slave to complete its self-test. During all this time, it is busy (the according flag in the status register is set). After finishing the test procedure, its results are placed in the error register. In this special case, the content of the error register has to be considered as a single byte value, not as several bit flags. There are the following error codes:

  01h   no error detected,
  03h   sector buffer error.
(These codes are supported by Conner drives. Maybe other manufacturers use more or different codes.)

If the slave drive diagnostics failed, the MSB of the error register is set, leading to values of 8xh. However, even with single drive configurations this bit sometimes is accidently set. It may be ignored then.

9.8 Set Drive Parameters (91h):

An IDE-only command again. After power-up or reset, the drive can immediately be used in its default mode. However, the drive's logical parameters can be changed by setting them with this command. This way, the drive can be set up to different modes in order to emulate the parameters of another common drive.
The task file registers which are used with this command, and the way in which they are used, may differ. Some drives are really flexible and allow any parameters that result in no more than the drive's real capacity. Other drives (for example, my Conner CP- 3044) support only two or three modes with fixed parameters. So for their selection, only part of the task file's registers are needed. Most, if not all, drives will accept this command with valid parameters in the SC, C, and H registers (even if not all the parameters are required), defining the number of sectors per track, cylinders, and heads.
Because of the differences, it is advisable to first collect detailed information about the supported emulation modes of a particular drive, before defining its operating parameters. Normally, it's best to operate a drive in its native mode (so the logical parameters equal the physical ones). However, there's another strange detail: there are drives which don't support the native mode! My Conner drive again serves as example: the drive has 1053x2x40 sectors (cylinders by heads by sectors) physically, but supports only a pseudo-native mode with 526x4x40 sectors, and an emulation mode with 981x5x17 sectors (which is for compatibility with older 40 MB drives). Additionally, depending on the internal software version, the drive defaults to the emulation mode or to the pseudo-native mode.
As a result, it is recommended that the operating parameters always be defined after power-up or reset. And to define them, you must have detailed information about the drive you want to use. There is a "Product Manual" for every drive type, describing all those details. Unfortunately, these manuals are hard to get. Most dealers are not willing to give them to their customers (and some even don't have them in stock). The other way is to try out some parameters, starting with the information delivered by the Identify Drive command.

10 The break - a sample program

I realize that I've already filled quite a few pages again. So I'll make a break here and continue the command descriptions in Part IV of the "Connecting IDE Drives" article series. Instead, I'll show you a short program which reads the ID information of an IDE drive within a PC/AT. This sample program was written with Turbo Pascal 5.5 but may easily be used with any version above 4.0.
You can try out this program on your AT (if you have one with an IDE drive) and play with it until receiving the next issue of TCJ with Part IV of the article. That part will finish the command descriptions and will also contain some more programming examples and shortform tables as a programmer's overview of the IDE interface definition.

Abbreviation list:

BFI Byte From Index (position of surface defect)
DRQ commonly used for Data Request (bit flag or signal line)
IDE Integrated Drive Electronics (hard disk interface type)
I/O Input/Output
PC/AT Personal Computer/Advanced Technology (computer class)
r/w read/write
ST-506 older hard disk interface standard, used between separate controllers and MFM/RLL drives

11 I must apologize!

Sorry for the badly formatted Pascal listing printed with part III in the previous issue of TCJ. Bill had to delete all the empty lines in order to compress it to a single page. Now I know that this doesn't make a program more readable or easier to understand, even if it's written in Pascal. We will try to do this better in the future.

12 Commands Continued. . .

We already covered most of the manufacturer-independent commands in the previous part. However, there are three commands not explained yet. Let's get started with the command which was already used in the sample program printed with the previous part – so you'll now know what you really did there (in case you ran that program).

12.1 Identify Drive (ECh):

This command reads some detailed parameter information from the IDE drive. Again, it's invalid for the older (external) controllers. It is started by writing the command code into the command register, and then it executes like a Read Sectors command. The DRQ Flag will be set, declaring that data can be read. After having read a complete "sector" (256 words, 512 bytes) of data, the DRQ flag will be reset and the drive will be ready again. The data consists of the following fields:

Word Adr.	Byte Adr.	Type	Content
0	0	word	Configuration/ID word
1	2	word	Number of fixed cylinders
2	4	word	No. of removable cylinders
3	6	word	No. of heads
4	8	word	No. of unformatted bytes per physical track
5	10	word	No. of unformatted bytes per sector
6	12	word	No. of phyical sectors per Tracks
7	14	word	No. of bytes in the inter-sector gaps
8	16	word	No. of bytes in the sync fields
9	18	word	0
10-19	20-39	20 char	Serial number
20	40	word	Controller type
21	42	word	Controller buffer size (in sectors)
22	44	word	No. of ECC bytes on "long" commands
23-26	46-53	8 char	Controller firmware revision
27-46	54-93	40 char	Model number
47	94	word	No. of sectors/interrupt (0 = no support)
48	96	word	Double word transfer flag (1 = capable)
49	98	word	Write protected
50-255	100-511	-	reserved (read as zero values)

Some of these fields have special meanings. The configuration/ID word consists of 16 single-bit flags. However, I don't know for sure if their meaning is really manufacturer- independent. The "controller type" word is encoded as a number representing a particular type.

Configuration/ID word bit flags:

15   Non-magnetic drive
14   Format speed tolerance gap required
13   Track offset option available
12   Data strobe offset option available
11   Rotational frequency tolerance > 0.5%
10   Data transfer rate > 10 MB/s
9   Data transfer rate > 5 MB/s, ≤ 10 MB/s
8   Data transfer rate ≤ 5 MB/s
7   Removable disk
6   Non-removable disk
5   Spindle motor can be switched off
4   Head switching time > 15 us
3   Not MFM encoded
2   Soft sectored
1   Hard sectored
0   reserved

Controller type word values:

0   Not specified
1   Single ported, single sector buffer
2   Dual ported, multiple sector buffer
3   = 2, with look-ahead read capabilities

The string-type data fields (character arrays) contain plain text information about the serial number, controller model, and firmware revision of the drive. Each word holds two characters, which must be displayed with the high-byte character first in order to get readable results.

As far as I know, most IDE drives follow the data field description above. However, there still are many things which are manufacturer-dependent. Fortunately, these details are not critical. To give you some examples: The controller model field of Conner drives contains plain text with the complete drive description like

    "Conner Peripherals 40 MB - CP3044".

Seagate's IDE drives offer only a short cryptic ID string, which sometimes doesn't even contain the drive type. A very interesting difference, even between drives of the same manufacturer, shows up with the "Number of cylinders/heads/sectors" fields. Some drives show their physical values there, independent of the active emulation mode (for example, my CP-3044 does so). Other drives always show the parameters of the active emulation, or those of the default emulation mode. Surprising especially with my drive is that the physical parameters can't be used for drive operation! As a result, the data delivered by this command must be considered carefully. However, it's normally possible to extract useful information by reading the drive's ID information for several different active emulation modes.

12.2   Read/Write Sector Buffer (E4h/E8h):

These are the last two common IDE commands. With these commands it's possible to read or write the drive's sector buffer directly. I haven't found any use for these yet, but probably there is (at least was) one. In my opinion, these commands are useless for normal operation.

12.3   Block Mode Commands (Read/Write/Set Multiple, C4..C6h):

By the use of these commands, one can access disk data in larger blocks than the physical sector size. Several sectors are grouped together and handled as a block of data. However, many drives don't support this mode. I don't have detailed information regarding the parameters. If a particular drive supports the block mode, the details will surely be printed in its user manual.

12.4   Power Commands (E0..E6h, except E4h):

The power commands are not supported by every IDE drive. However, if they are, they are normally compatible. The power commands are commonly used within portable computers (laptops, notebooks, handhelds, or whatever the names are). They allow for automatic or manual changing between normally four operation modes:

Read/Write Mode	(4.2 W)	complete drive circuitry operating
Idle Mode	(2.0 W)	motor running, r/w circuitry turned off while no command is active
Standby Mode	(0.5 W)	motor stopped, r/w circuitry turned off, interface active
Sleep Mode	(n/a)	everything stopped, exit only with reset

The power requirements mentioned in this table are those of my Conner 42-MB drive. While no r/w operation is in progress, the drive normally is in idle mode (also when being reset). Read/write mode is always automatically entered when a r/w command is issued; after completion of that command, the drive enters idle mode again. When the drive is put into standby mode (manually or automatically, see below), the drive (motor, r/w circuit) is shut down while the host interface remains active. So when a command is issued which requires motor or r/w operation, the appropriate circuitry is automatically switched on again.

Once the sleep mode is entered, there is no way out except for reset by means of hardware or software. This is because even the drive's local processor and interface controller are stopped, so there is no way to communicate with the drive. (However, the task file can still be read.)

As mentioned above, there are six power commands:

12.4.1 Set Standby Mode (E0h), Set Idle Mode (E1h):

The drive will enter the desired mode immediately. There are no parameters required. If the drive already is in that mode, the command will have no effect.

12.4.2 Set Standby (E2h) or Idle (E3h) Mode with Auto-Power-Down:

These commands take a parameter in the sector count register. If that parameter is non-zero, the Auto-Power-Down (APD) feature is enabled (with a zero value, APD is disabled). When one of these commands is issued, the drive immediately enters the desired mode. If APD is enabled, the drive will automatically enter standby mode after being in idle mode without activities for a given period of time. This delay can be specified by means of the parameter for these two commands: the SC register must contain the delay time in counts of 5 seconds. The minimum delay of 60 seconds will be set if the SC register contents is smaller than 12. With a maximum value of 220, the maximum delay is about 18 minutes. These limits again apply to my particular drive; other drives may have other specifications.

12.4.3 Read Power Mode (E5h):

This command reads the actual mode. If the motor is spinning (meaning that the drive is in idle mode), the value FFh will be returned in the SC register. Else (when in standby mode or just spinning up) a zero value will be placed in the SC register.

12.4.4 Set Sleep Mode (E6h):

This command puts the drive into sleep mode immediately. Every internal activity is terminated and all circuitry switched off.

There are some more power-related commands, having the command codes F8..FDh (except FCh). Their general meaning is similar to the power commands described above (E0..E5h), except that the time delays are specified more exactly (in counts of 0.1 seconds). However, I have not yet seen a drive which supported these commands, and I don't have detailed information about them.

12.5 Cache On/Off (EFh):

This is the last command which I will explain here. It is used for enabling or disabling the automatic read-ahead feature (read cache) of the drive. The write precompensation register (WP) is (mis-) used as a parameter register for this command (today, this is the only use of the WP register). If the WP register contains AAh, the feature is enabled; with 55h, it is disabled. Every other value will result in an aborted command error. After reset, the drive defaults to read-ahead feature enabled.

Whew – this was a lot of stuff! (I hope it was not too hard.) However, now you should know about IDE commands in detail (if you didn't fall asleep while reading). Before we start practical work, here, for the programmers, are the short-form tables that I promised.

13   Overview

Table 1: Task File Registers (as printed in part II)

/CS0	/CS1	A2	A1	A0	Addr.	Read Function	Write Function
0	1	0	0	0	1F0	Data Register	Data Register
0	1	0	0	1	1F1	Error Register	(Write Precomp Reg.)
0	1	0	1	0	1F2	Sector Count	Sector Count
0	1	0	1	1	1F3	Sector Number	Sector Number
0	1	1	0	0	1F4	Cylinder Low	Cylinder Low
0	1	1	0	1	1F5	Cylinder High	Cylinder High
0	1	1	1	0	1F6	SDH Register	SDH Register
0	1	1	1	1	1F7	Status Register	Command Register
1	0	1	1	0	3F6	Alternate Status	Digital Output
1	0	1	1	1	3F7	Drive Address	Not Used

Table 2: Error Register

Bit	Flag	Meaning
7	BBK	Bad block mark detected
6	UNC	Uncorrectable data error
5	-	-
4	IDNF	Sector ID not found
3	-	-
2	ABRT	Command aborted (status error or invalid command)
1	TK0	Track 0 not found during recalibration
0	-	-

Table 3: SDH Register

Bit	Flag	Meaning
7	EXT	Extension Bit. Always 1.
6-5	SIZE	Sector Size. Always 01 (512 byte sectors).
4	DRV	Drive bit. Master/single drive = 0, slave = 1.
3-0	HEAD	Head field. Binary head number 0..15.

Table 4: Status Register, Alternate Status Register

Bit	Flag	Meaning
7	BSY	Drive busy. Task file cannot be accessed.
6	DRDY	Drive ready (up to speed and ready for command).
5	DWF	Drive write fault.
4	DSC	Drive seek complete (actuator on track).
3	DRQ	Data request (ready for data transfer).
2	CORR	Corrected data (bit is set when data has been recovered by use of ECC).
1	IDX	Index. Active once per disk revolution.
0	ERR	Error. See other bits and error register.

Table 5: Digital Output Register

Bit	Flag	Meaning
2	SRST	Software reset (active when set to 1).
1	/IEN	Interrupt enable (active when set to 0).

Table 6: Drive Address Register

Bit	Flag	Meaning
7	-	not driven (for PC floppy compatibility)
6	/WTG	Write gate (active when 0)
5-2	/HSx	Head select 3..0, one's complement of active head
1	/DS1	Drive 1 selected (active when 0)
0	/DS0	Drive 0 selected (active when 0)

Table 7: Commonly needed Commands with Parameters

Code	Command	Parameters
1x	Recalibrate	D
20	Read Sectors with retry	SC,SN,C,D,H
30	Write Sectors with retry	SC,SN,C,D,H
40	Verify Sectors with retry	SC,SN,C,D,H
50	Format Track	C,D,H
7x	Seek	C,D
90	Exec Diagnostics	D
91	Set Drive Parameters	SC,(C),D,H
Ex	Power Commands, see below	
E4	Read Sector Buffer	D
E8	Write Sector Buffer	D
EC	Identify Drive	D
EF	Cache On/Off	D,WP

Power Commands:

Code	Command	Parameters
E0	Standby Mode	-
E1	Idle Mode	-
E2	Standby Mode with APD	SC
E3	Idle Mode with APD	SC
E5	Read Power Mode	(SC)
E6	Sleep Mode	-

Table 8: Error Conditions
When an error occurs, the error flag in the status register (ERR) is always set. For the different groups of commands, the following status/error flags are valid then:

Recalibrate	ABRT,TK0,DRDY,DWF,DSC
Read, Verify	BBK,UNC,IDNF,ABRT,DRDY,DWF,DSC,CORR
Read Long, Write, Write Long	BBK,IDNF,ABRT,DRDY,DWF,DSC
Format, Seek	IDNF,ABRT,DRDY,DWF,DSC
Diagnostics, Initialize, R/W Buffer, Identify, Set Cache	ABRT
Invalid command	ABRT

Table 9: Interrupt Conditions
The drive generates an interrupt (if enabled) under the following conditions:

Recalibrate	after successfully reaching track 0
Read	each time DRQ is set
Write	when DRQ is set, from second sector on (only when multiple sectors are written)
Verify	after completion for all sectors
Format Track	after completion
Seek, Initialize, Power Commands (except Sleep)	after command is issued/initiated
Set Sleep Mode	when drive is in sleep mode
Read Buffer, Identify	when data is ready for reading

14   Example Program

Now let's come to the example routines for accessing an IDE drive. These examples are given as Turbo-Pascal (3.0) source (based on my IDE test program). They apply to the use of my IDE interface board (described in TCJ #56), so there always are 512 data bytes transferred instead of 256 data words.
In all examples, named constants are used for accessing the IDE registers at their particular I/O addresses. These named constants must be declared elsewhere. Their names are derived from the related IDE register names and IDE commands.
The examples are programmed in a very modular fashion so that they are easy to understand. For implementation in a system BIOS, for example, most of the subroutines will contain so little code that the complete read/write routines will normally be coded inline. In addition, a real implementation, unlike these examples,

will have time-out functions in most loops. If someone is interested in the IDE driver of my CPU280 system BIOS, please contact me (however note, it's Z280 assembly language and commented in German).

14.1   General access: Wait for drive ready / wait for data request

In Pascal, two small procedures serve this purpose. In assembly language, I use two macros instead, because the subroutine calling overhead would be too much.

```
procedure Wait_Ready;
begin
  repeat until port[IDE_CmdStat]<=128;
  end;

procedure Wait_DRQ;
begin
  repeat until port[IDE_CmdStat] and 8<>0;
  end;
```

14.2   General access: Command issue

```
procedure IDE_Command(Cmd:byte);
begin
  Wait_Ready;
  port[IDE_CmdStat]:=Cmd;
  end;
```

14.3   General access: Reading/Writing the sector buffer

In the Pascal implementation, the two functions return a Boolean value which is true if there were no errors during r/w of the buffer.
Both routines require the drive to be ready for data transfer!

```
function Read_SecBuf(var Buf:BufType):boolean;
var i : integer;
begin
  Wait_DRQ;
  i:=port[IDE_Data];  (* specific to my IDE interface board *)
  for i:=0 to 511 do Buf[i]:=port[IDE_Data];
  Read_SecBuf:=port[IDE_CmdStat] and $89=0;
  end;

function Write_SecBuf(var Buf:BufType):boolean;
var i : integer;
begin
  Wait_DRQ;
  for i:=0 to 511 do port[IDE_Data]:=Buf[i];
  Wait_Ready;
  Write_SecBuf:=port[IDE_CmdStat] and $89=0;
  end;
```

14.4   General access: First access, initialization

```
procedure HD_Init(Cyls,Heads,Secs:integer);
begin
  port[Dig_Out]:=6;
  delay(10);               (* Drive Software Reset *)
  port[Dig_Out]:=2;
  Wait_Ready;
  port[IDE_SecCnt]:=Secs;
  port[IDE_CylLow]:=lo(Cyls);
  port[IDE_CylHigh]:=hi(Cyls);
  port[IDE_SDH]:=pred(Heads)+$A0;
```

```
  IDE_Command(Cmd_Initialize);
  end;
```

14.5   Data access: Single sector read

```
function HD_ReadSector(Cyl,Head,Sec:integer; var Buf:BufType):boolean;
begin
  Wait_Ready;
  port[IDE_SecCnt]:=1;
  port[IDE_SecNum]:=Sec;
  port[IDE_CylLow]:=lo(Cyl);
  port[IDE_CylHigh]:=hi(Cyl);
  port[IDE_SDH]:=$A0+Head;
  IDE_Command(Cmd_ReadSector);
  HD_ReadSector:=Read_SecBuf(Buf);
  end;
```

14.6   Data access: Single sector write

```
function HD_WriteSector(Cyl,Head,Sec:integer; var Buf:BufType);
begin
  Wait_Ready;
  port[IDE_SecCnt]:=1;
  port[IDE_SecNum]:=Sec;
  port[IDE_CylLow]:=lo(Cyl);
  port[IDE_CylHigh]:=hi(Cyl);
  port[IDE_SDH]:=$A0+Head;
  IDE_Command(Cmd_WriteSector);
  HD_WriteSector:=Write_SecBuf(Buf);
  end;
```