12 July 1991

John Lohmeyer
NCR Corporation
3718 N. Rock Road
Wichita, KS 67226

Dear John,

I have forwarded for committee consideration copies of the proposed Standard AT Compatible Register Map and the Common Configuration Method for storage devices. These two documents outline a standard way of addressing AT compatible controllers/host bus adapters as well as a standard way of storing device specific geometry and information on ATA/IDE storage devices.

The proposed Standard AT Compatible Register Map defines four "Standard" sets of addresses based at; 1F0h, 170h, 1E8h, and 168h. The device-specific information is recorded on the device in logical sector number two. Any driver or operating system can access the information for configuration purposes, etc. Specific fields can be updated or maintained by the operating software if desired. If this were done, a new checksum would be recomputed and saved in the location defined. The device-specific information would either be recorded on the device by the manufacturer or by a separate utility program.

Using the Standard Register Map and the Common Configuration Method, a standard software driver would be able to locate each controller/host bus adapter and identify each storage device with its specific information.

This would provide a standard way to access disk drive geometry in configurations using multiple controllers/host bus adapters and multiple storage devices attached to each controller/host bus adapter.

This proposal is not specific to any operating system but is intended for general industry use.

Sincerely,

Royes B. Richins
Sr. Software Engineer

cc Larry Lamers
Maxtor Corp.
Proposed Common Configuration Method

July 9, 1991

The Common Configuration Method (CCM) simplifies configuration of disks and other storage devices. This definition includes a Standard AT Compatible Register Map, and a configuration sector layout. Drivers incorporating this method of configuration will be able to find the controller and recognize the geometry and options of any media formatted and containing the CCM signature and required information in the indicated fields in sector 02 (third logical sector of the media on the device, assuming 512 byte sectors). This method of configuring for drives allow media to be identified universally independent of a given operating system or system BIOS. A driver incorporating this technique for configuration will be required to read sector 02 in the driver, recognize the Common Configuration Signature, and get the necessary information from the fields as defined in the Common Configuration sector layout in the following pages.
### Common Configuration Method

#### Sector 02 Layout

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset (hex)</th>
<th>Offset (dec)</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000 - 0FF</td>
<td>000 - 255</td>
<td>unsigned char [256]</td>
<td>Vendor Unique&lt;br&gt;Reserved for vendor use</td>
</tr>
<tr>
<td>2*</td>
<td>100 - 101</td>
<td>256 - 257</td>
<td>unsigned int</td>
<td>Signature&lt;br&gt;Contains the signature (55AAh)</td>
</tr>
<tr>
<td>3</td>
<td>102 - 105</td>
<td>258 - 261</td>
<td>unsigned long</td>
<td>User blocks (low-order long word)&lt;br&gt;Low order half of following</td>
</tr>
<tr>
<td>4</td>
<td>106 - 109</td>
<td>262 - 265</td>
<td>unsigned long</td>
<td>User blocks (high-order long word)&lt;br&gt;The user sectors divided by the block size**</td>
</tr>
<tr>
<td>5*</td>
<td>10A - 10B</td>
<td>266 - 267</td>
<td>unsigned int</td>
<td>User Data Heads&lt;br&gt;Number of user-accessible data heads**</td>
</tr>
<tr>
<td>6*</td>
<td>10C - 10F</td>
<td>268 - 271</td>
<td>unsigned long</td>
<td>User Cylinders&lt;br&gt;Number of user accessible cylinders** (not including spares)</td>
</tr>
<tr>
<td>7*</td>
<td>110 - 111</td>
<td>272 - 273</td>
<td>unsigned int</td>
<td>Average Sectors Per Track&lt;br&gt;Average number of user-accessible sectors in each track**</td>
</tr>
<tr>
<td>8*</td>
<td>112 - 115</td>
<td>274 - 277</td>
<td>unsigned long</td>
<td>User Sectors (low-order long word)&lt;br&gt;lower half of number of sectors (below)</td>
</tr>
<tr>
<td>9*</td>
<td>116 - 119</td>
<td>278 - 281</td>
<td>unsigned long</td>
<td>User Sectors (high-order long word)&lt;br&gt;Total user accessible sectors on device**</td>
</tr>
<tr>
<td>10</td>
<td>11A - 11B</td>
<td>282 - 283</td>
<td>unsigned int</td>
<td>BlockSize&lt;br&gt;Data transfer size in sectors per block**</td>
</tr>
<tr>
<td>11*</td>
<td>11C - 11D</td>
<td>284 - 285</td>
<td>unsigned int</td>
<td>Sector Data Length&lt;br&gt;Number of data bytes in each sector**</td>
</tr>
<tr>
<td>12</td>
<td>11E - 13D</td>
<td>286 - 317</td>
<td>unsigned char [32]</td>
<td>Support Field&lt;br&gt;Feature identification (for vendor use)</td>
</tr>
<tr>
<td>13*</td>
<td>13E - 13F</td>
<td>318 - 319</td>
<td>unsigned int</td>
<td>Controller Interface Type&lt;br&gt;0 = unknown 3 = RSDI 6 = ST-506&lt;br&gt;1 = IDE/ATA 4 = SMD&lt;br&gt;2 = SCSI 5 = IPI&lt;br&gt;If greater than 6, refer to Controller Name</td>
</tr>
<tr>
<td>14*</td>
<td>140 - 14F</td>
<td>320 - 335</td>
<td>unsigned char [16]</td>
<td>Model Name***</td>
</tr>
<tr>
<td>15</td>
<td>150 - 15F</td>
<td>336 - 351</td>
<td>unsigned char [16]</td>
<td>Controller Name***</td>
</tr>
</tbody>
</table>
### Common Configuration Method
#### Sector 02 Layout (cont)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset (hex)</th>
<th>Offset (dec)</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16*</td>
<td>160 - 161</td>
<td>352 - 353</td>
<td>unsigned int</td>
<td><strong>Peripheral Device Type</strong>**</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00 = Hard disk</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01 = Magnetic Tape</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02 = Printer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>03 = Processor Device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>04 = WORM Device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>05 = CD-ROM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>06 = Scanner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>07 = Optical Disk</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>08 = Medium Changer Device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>09 = Communications Device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0Ah-0Bh = Defined by ASC IT8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0Ch-1Eh = Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1Fh = Unknown or no device type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7Fh = Logical Unit Missing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>80h-FFh = Vendor Unique</td>
</tr>
<tr>
<td>17*</td>
<td>162 - 175</td>
<td>354 - 373</td>
<td>unsigned char</td>
<td><strong>Device Serial Number</strong>*</td>
</tr>
<tr>
<td>18</td>
<td>176 - 179</td>
<td>374 - 377</td>
<td>unsigned char</td>
<td><strong>Unique Device Address</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In family</td>
</tr>
<tr>
<td>19</td>
<td>17A - 199</td>
<td>378 - 409</td>
<td>unsigned long</td>
<td><strong>Start-up Sector Pointers</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sector numbers for devices that require special</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>start-up routines</td>
</tr>
<tr>
<td>20</td>
<td>19A - 1FB</td>
<td>410 - 507</td>
<td>unsigned char</td>
<td><strong>Reserved</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>21*</td>
<td>1FC - 1FF</td>
<td>508 - 511</td>
<td>unsigned long</td>
<td><strong>CRCDoubleWord</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32-bit CRC of the CCM area (0100h-01FBh)</td>
</tr>
</tbody>
</table>

* Fields that must contain valid data if the signature is written to bytes 100h-101h
** An absolute count; i.e., a whole number total; 0=zero units, 1=one unit, etc.
*** A null-terminated and null-padded ASCII string

**Note:** The decimal whole numbers in brackets [] indicate the storage allocation in bytes in the designated data type

#### DEFINITIONS

- **byte** An 8-bit unit, such as a character
- **word** A 16-bit unit, such as an integer
- **long word** A 32-bit unit, such as a long integer
- **vendor** Manufacturer of the device
This section details the contents and definitions of all the fields specified by the Common Configuration Method (CCM) for sector 02 of peripheral devices. All addresses are designated in hex and pertain to the data field of sector 02. All fields are unsigned.

Field 1: **Vendor Unique**  
256 bytes unsigned char

The **Vendor Unique** field is reserved for purposes specific to the manufacturer and is comprised of addresses 000h through 0FFh (the lower-addressed half of sector 02); Because the information stored at this location is vendor-specific, the generic data type unsigned char is assigned to it.

Field 2: **Signature**  
2 bytes unsigned int

The **Signature** field contains the hex number 55AA (decimal 21930) at address 100h of sector 02 to indicate the remaining data follows the CCM standard. This signature also indicates that valid data is stored in the User Data Heads, Average Sectors Per Track, User Type, and Serial Number fields, and a valid checksum for the CCM data is stored in the Checksum field.

Field 3: **User Blocks**, low order long word  
4 bytes unsigned long

Field 4: **User Blocks**, high order long word  
4 bytes unsigned long

The lower 32 bits of the 64-bit number representing the number of blocks on the device is stored at address 102h, the low order long word of the User Blocks field. The upper 32 bits of the same 64-bit number is stored at address 106h, the high order long word of the User Blocks field. This is equal to the User Sectors divided by the Block Size. The number of user blocks is a whole number, indicating an actual count of the number of blocks available to the user; i.e., 0 = zero blocks, 1 = one block, 2 = two blocks, etc.

Field 5: **User Data Heads**  
2 bytes unsigned int

The number of data heads on the device accessible to the user is stored in the User Data Heads field at address 10Ah and does not include servo heads. Valid data must be present in this field if the proper signature is written to the Signature field. The number of user data heads is a whole number, indicating an actual count of the number of data heads available to the user; i.e., 0 = zero heads, 1 = one head, 2 = two heads, etc.
Field 6: **User Cylinders**

4 bytes unsigned long

The number of cylinder so the device accessible to the user is stored in the User Cylinders field at address 10Ch and does not include spare cylinders. Valid data must be present in this field if the proper signature is written to the Signature field. The number of user cylinders is a whole number, indicating an actual count of the number of cylinders available to the user; i.e., 0 = zero cylinders, 1 = one cylinder, 2 = two cylinders, etc.

Field 7: **Average Sectors Per Track**

2 bytes unsigned int

The average number of sectors in each track accessible to the user is stored in the Average Sectors Per Track field at address 110h. Valid data must be present in this field if the proper signature is written to the Signature field. The average number of sectors per track is a whole number, indicating an actual count of the average number of sectors per track available to the user; i.e., 0 = zero sectors per track, 1 = one sector per track, 2 = two sectors per track, etc.

Field 8: **User Sectors**, low order long word

4 bytes unsigned long

Field 9: **User Sectors**, high order long word

4 bytes unsigned long

The lower 32 bits of the 64-bit number representing the number of sectors on the device is stored at address 112h, the low order long word of the User Sectors field. The upper 32 bits of the same 64-bit number is stored at address 116h, the high order long word of the User Sectors field. This is equal to the User Blocks multiplied by the Block Size. Valid data must be present in this field if the proper signature is written to the Signature field. The number of user sectors is a whole number, indicating an actual count of the number of sectors available to the user; i.e., 0 = zero sectors, 1 = one sector, 2 = two sectors, etc.

Field 10: **Block Size**

2 bytes unsigned int

The data transfer size in the number of sectors per block is stored in the Block Size field at address 11Ah. The block size is a whole number, indicating an actual count of the number of sectors per block available to the user; i.e., 0 = zero sectors per block, 1 = one sector per block, 2 = two sectors per block, etc.
Common Configuration Method

Field 11: **Sector Data Length**  2 bytes unsigned int

The number of data bytes in each sector is stored in the Sector Data Length field at address 11Ch. Valid data must be present in this field if the proper signature is written to the Signature field. The sector data length is a whole number, indicating an actual count of the number of data bytes per sector available to the user; i.e., 0 = zero data bytes per sector, 1 = one data byte per sector, 2 = two data bytes per sector, etc.

Field 12: **Support Field**  32 bytes unsigned char

The support Field is an information field that is allocated for feature identification of the device and is reserved for use by the vendor.

Field 13: **Controller Interface Type**  2 bytes unsigned int

The 16-bit number that identifies the type of interface the device communicates with is stored in the Controller Interface Type field at address 13Eh. This information is defined as follows:

- 0 = unknown controller type
- 1 = IDE/ATA
- 2 = SCSI I/SCSI II
- 3 = ESDI
- 4 = SMD
- 5 = IPI
- 6 = ST-506

Controller types represented by numbers greater than 6 are identified in the Controller Name field. Valid data must be present in this field if the proper signature is written to the Signature field.

Field 14: **Model Name**  16 bytes unsigned char

The model name of the device is stored in the Model Name field at address 140h. This is a null-padded and null-terminated ASCII string, meaning that (1) the last byte must be a null (00h) character and (2) all other unused bytes must also be null characters.

Field 15: **Controller Name**  16 bytes unsigned char

The controller name of the device is stored in the Controller Name field at address 150h. This is a null-padded and null-terminated ASCII string, meaning that (1) the last byte must be a null (00h) character and (2) all other unused bytes must also be null characters.
Field 16: **Peripheral Device Type** 2 bytes unsigned int

The 16-bit number that identifies the medium or peripheral device type is stored in the Peripheral Device Type field at address 160h. This information is defined as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Hard Disk</td>
</tr>
<tr>
<td>01h</td>
<td>Magnetic Tape</td>
</tr>
<tr>
<td>02h</td>
<td>Printer</td>
</tr>
<tr>
<td>03h</td>
<td>Processor Device</td>
</tr>
<tr>
<td>04h</td>
<td>WORM Device</td>
</tr>
<tr>
<td>05h</td>
<td>CD-ROM</td>
</tr>
<tr>
<td>06h</td>
<td>Scanner</td>
</tr>
<tr>
<td>07h</td>
<td>Optical Disk</td>
</tr>
<tr>
<td>08h</td>
<td>Medium Changer Device</td>
</tr>
<tr>
<td>09h</td>
<td>Communications Device</td>
</tr>
<tr>
<td>0Ah-0Bh</td>
<td>Defined by ASC IT8</td>
</tr>
<tr>
<td>0Ch-1Eh</td>
<td>Reserved</td>
</tr>
<tr>
<td>1Fh</td>
<td>Unknown/No Device Type</td>
</tr>
<tr>
<td>7Fh</td>
<td>Logical Unit Missing</td>
</tr>
<tr>
<td>80h-FFh</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>

Valid data must be present in this field if the proper signature is written to the Signature field.

Field 17: **Serial Number** 20 bytes unsigned char

The serial number of the device is stored as an ASCII string in the Serial Number field at address 162h. This is a null-padded and null-terminated ASCII string, meaning that (1) the last byte must be a null (00h) character and (2) all other unused bytes must also be null characters. Valid data must be present in this field if the proper signature is written to the Signature field.

Field 18: **Unique Device Address** 4 bytes unsigned char

The unique address that distinguishes the device from other members of the same family is stored in the Unique Device Address field at address 176h.

Field 19: **Startup Sector Pointers** 32 bytes unsigned long

The Startup Sector Pointers field is provided to store sector numbers that locate routines required by some devices to complete the startup process.

Field 20: **Reserved** 98 bytes unsigned char

The Reserved field is allocated for future expansion.
Field 21: **CRCDoubleWord**  
4 bytes unsigned long

This field contains a 32-bit CRC of the CCM data starting at location 0100h and ending with location 01FBh. The CRC is calculated using the supplied "C" routine. Please note that the CRC accumulator is initialized to all ones (0xFFFFFFFh) and that the resulting value is inverted. This standard technique prevents generating a matching CRC when data is erroneously shifted by one or more bytes, with more or less initial zero bytes. The polynomial used in the 32-bit CRC is found in the ANSI X3.66 specification, or FED-STD-1003, and is represented as:

\[ x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^8 + x^7 + x^5 + x^4 + x^3 + x^0 \]

Valid data should be present in this field if the proper signature is written to the **Signature** field.

Following are sample routines to calculate the CRCDoubleWord in both "C" and Assembler, along with a sample definition file to allow the 32-bit mode assembler routine to be loaded as an NLM, exporting the routine for drivers and other NLMs:
Sample 32-bit CRC32 Routine

*ENTRYPOINT: GenerateCRC32

*PURPOSE: Generates 32-bit CRCs for data blocks (32-bit mode flat model)

*PROTOTYPE: CRC = GenerateCRC32(bufferlength, @buffer)

*PARAMETERS: bufferlength - LONG buffer length in bytes
@buffer - LONG pointer to data buffer

*RETURNS: CRC - LONG new accumulated CRC (EAX)

*NOTES: This CRC polynomial is in the ANSI X3.66 specification. 32-bit CRCs are approximately 100,000 times more likely to detect an error than 16-bit CRCs. The 32-bit polynomial used is:

32 26 23 22 16 12 11 10 8 7 5 4 2 1 0
X +X +X +X +X +X +X +X +X +X +X +X +X +X +X +X

*:-
name GenerateCRC32
assume ds: OSDATA, es: OSDATA, ss: OSDATA

CPush macro
push ebp
push ebx
push esi
push edi
cld
endm

CPop macro
pop edi
pop esi
pop ebx
pop ebp
endm
; following used to fetch values from the stack
ParmOffset equ 20 ;for ebx,ebp,esi,edi plus near call
Parm0 equ ParmOffset + 0
Parm1 equ ParmOffset + 4
poly equ 04C11DB3h

OSDATA segment rw public 'DATA'
OSDATA ends

OSCODE segment er public 'CODE'
assume cs: OSCODE
public GenerateCRC32
align 4

CRC32C proc
GenerateCRC32 label near

CPush ;push standard registers for C
mov ecx, [esp + Parm0] ;*** get # bytes to process
mov esi, [esp + Parm1] ;*** get buffer pointer
xor eax, eax ;initialize crc accumulator
dec eax ;to all ones
mov edx, poly ;get polynomial constant
jecxz exit ;*** if no bytes to process
align 4
byteloop:
    rol eax, 8 ;update with new byte
    xor eax, [esi]
    ror eax, 8
    mov bh, 8 ;set bit counter
    align 4
bitloop:
    shl eax, 1 ;shift accumulator left 1 bit
    jnc bitzero ;if bit was zero
    xor eax, edx ;update accumulator
    align 4
bitzero:
    dec bh ;done with byte ?
    jnz bitloop ;no - do next bit
    inc esi ;adjust buffer pointer
    loop byteloop ;loop til buffer processed
exit:
    xor eax,-1 ;1's complement result
    CPop ;restore saved regs
    ret ;return to caller
CRC32C endp
OSCODE ends
end
Sample DOS real-mode CRC32 C-callable routine

*MODULE: CRC32.ASM

*PURPOSE: Generates 32-bit CRCs for data blocks (DOS Real-Mode)

*ENVIRONMENT: DOS Client

*MODEL: HUGE

*MODIFIES INTERRUPTS: n

*REQ CALL DISABLED: n

*C-CALLABLE: y

*PARAMETERS: bufferlength - LONG buffer length in bytes

@buffer - LONG pointer to data buffer

*RETURNS: CRC - (AX:DX) LONG new accumulated CRC

*MODIFIES: AX, BX, CX, DX, FLAGS

*NOTES: Assemble with the /MX option.

The 32-bit polynomial used is:

\[ X^{3} + X^{22} + X^{62} + X^{32} + X^{21} + X^{61} + X^{21} + X^{11} + X^{8} + 754210 \]

The polynomial is defined in ANSI X3.66.

*:~

Parms equ 6 ;far call plus bp
Parm0Lo equ Parms+0
Parm0Hi equ Parms+2
Parm1Lo equ Parms+4
Parm1Hi equ Parms+6
polylo equ 10B3h ;low word of 32-bit polynomial
polyhi equ 04C1h ;high word - note that 32 bit polynomials actually are
;33-bit by definition
public _CRC32
_TEXT segment DWORD public 'CODE'
assume CS:_TEXT
GenerateCRC32 proc far
_CRC32 label DWORD
; this routine alters AX, BX, CX, DX (Turbo C or MSC 5.1 compatible)
push bp
mov bp, sp ;setup bp
push es ;save
push si ;save
push di ;save
mov cx, [bp + Parm0Lo] ;*** get # bytes to process
mov bx, [bp + Parm0Hi] ;*** get high word of count
mov si, [bp + Parm1Lo] ;*** get buffer pointer offset
mov ax, [bp + Parm1Hi] ;*** get segment address
mov es, ax ;set up ES
mov ax, 0FFFFh ;initialize 32-bit accumulator
mov dx, ax
or bx, bx ;hi-order count non-zero
jcxz SHORT exit ;if no bytes to process
align 2
byteloop:
xor dh, ES:[si] ;update with new byte
mov bh, 8 ;loop for each bit in byte
align 2
bitloop:
shl ax,1 ;shift accumulator left 1 bit
rcl dx,1 ;all 32 bits
jnc SHORT bitzero ;if bit shifted out was zero
xor ax, polylo ;update crc with polynomial
xor dx, polyhi
align 2
bitzero:
dec bh ;done with byte ?
jnz bitloop ;no - do next bit
inc si ;adjust buffer pointer
jz SHORT fixseg ;skip if at segment end
align 2
bytechk:
loop byteloop ;loop til buffer processed
dec bl ;theoretical limit 16M bytes
jnz bytechk ;until all done
exit:
xor ax, 0FFFFh ;if's complement result
xor dx, 0FFFFh ;per ANSI standard
pop di ;restore
pop si
pop es
pop bp
ret ;return to caller
fixseg:
mov si, es ;fixup ES:si if end of segment reached
add si, 1000h ;move up 64k
mov es, si ;restore
xor si, si ;registers
jmp bytechk
;return to mainline
GenerateCRC32 endp
_TEXT ends
end
Sample CRC32 routine in C

unsigned long CGenerateCRC32(unsigned long bufferlength, unsigned char *buffer)
{
    unsigned long crc;
    unsigned long tmp;
    unsigned long bytecnt;
    unsigned long bitcnt;

    crc = 0xffffffff;
    if (bufferlength == 0) return (crc ^ 0xffffffff);
    for (bytecnt = 0, bytecnt < bufferlength, bytecnt++)
    {
        tmp = *(unsigned long int) buffer;
        crc ^= (tmp << 24);
        for (bitcnt = 0, bitcnt < 8, bitcnt++)
        {
            tmp = (crc & 0x80000000);
            crc = (crc << 1);
            if (tmp) crc ^= 0x04C11DB3;
        }
        buffer++;
    }
    return (crc ^ 0xffffffff);
}
Proposed Standard AT Compatible

Register Map

25 Jun 1991

This document lists the proposed Standard AT Compatible Register Map for IBM AT systems and true compatibles.
**Proposed Standard AT Compatible Register Map**

(all addresses are in hex)

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Primary Address</th>
<th>Secondary Address</th>
<th>Tertiary Address</th>
<th>Quaternary Address</th>
<th>Register Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>1F0</td>
<td>170</td>
<td>1E8</td>
<td>168</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Error</td>
<td>1F1</td>
<td>171</td>
<td>1E9</td>
<td>169</td>
<td>Read Only</td>
</tr>
<tr>
<td>Write Precomp</td>
<td>1F1</td>
<td>171</td>
<td>1E9</td>
<td>169</td>
<td>Write Only</td>
</tr>
<tr>
<td>Sector Count</td>
<td>1F2</td>
<td>172</td>
<td>1EA</td>
<td>16A</td>
<td>Write Only</td>
</tr>
<tr>
<td>Sector Number</td>
<td>1F3</td>
<td>173</td>
<td>1EB</td>
<td>16B</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Cylinder Low</td>
<td>1F4</td>
<td>174</td>
<td>1EC</td>
<td>16C</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Cylinder High</td>
<td>1F5</td>
<td>175</td>
<td>1ED</td>
<td>16D</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Drive/Head</td>
<td>1F6</td>
<td>176</td>
<td>1EE</td>
<td>16E</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Status</td>
<td>1F7</td>
<td>177</td>
<td>1EF</td>
<td>16F</td>
<td>Read Only</td>
</tr>
<tr>
<td>Command</td>
<td>1F7</td>
<td>177</td>
<td>1EF</td>
<td>16F</td>
<td>Write Only</td>
</tr>
<tr>
<td>Alternate Status</td>
<td>3F6</td>
<td>376</td>
<td>3EE</td>
<td>36E</td>
<td>Read Only</td>
</tr>
<tr>
<td>Fixed Disk</td>
<td>3F6</td>
<td>376</td>
<td>3EE</td>
<td>36E</td>
<td>Write Only</td>
</tr>
<tr>
<td>Digital Input</td>
<td>3F7</td>
<td>377</td>
<td>3EF</td>
<td>36F</td>
<td>Read Only</td>
</tr>
</tbody>
</table>