Table 288. Sequence Register

| Bit | Function |
|---|---|
| 7 | **DMA:** The DMA bit indicates that the transfer is to be done using DMA. If this bit is enabled, FIFO accesses are disabled. |
| 6 | **TMode :** When set, the TMode bit puts the controller into Target mode for executing the command. The TMode bit is cleared for Initiator mode. |
| 5 | **Atn Bit:** For the Selection and Information Transfer phases, the Atn bit indicates that the Atn signal shall also be asserted. |
| 4 | **ActNeg:** If the ActNeg bit is set, the SCSI controller adopts Active Negation mode. In this mode it actively negates the Req and Ack signals by pulling them high instead of relying on line terminators. Only the Req and Ack signals are affected. If both ActNeg and TMode are set, the controller pulls Req actively both low and high; if TMode is clear, then Ack is actively negated. |
| 3-0 | **Seq:** The four Seq bits, seq3..seq0, contain the encoded value of the commands defined in Table 289 on page 151. |

Table 289. Seq bit encoding

| Command | Value | Normal completion | Exception | Error |
|---|---|---|---|---|
| Arb | 0x1 | Arbitration won (2.4µs) | ArbLost, SCSIReset, Reselect | |
| Sel | 0x2 | Destination selected | SelTO, SCSIReset, UnExpDisc | |
| Cmd | 0x3 | Transfer count exhausted | PhaseMM, SCSIReset, UnExp-Disc | |
| Status | 0x4 | Transfer count exhausted | PhaseMM, SCSIReset, UnExp-Disc | ParErr |
| DataOut | 0x5 | Transfer count exhausted | PhaseMM, SCSIReset, UnExp-Disc | |
| DataIn | 0x6 | Transfer count exhausted | PhaseMM, SCSIReset, UnExp-Disc | ParErr |
| MsgOut | 0x7 | Transfer count exhausted | PhaseMM, SCSIReset | |
| MsgIn | 0x8 | Transfer count exhausted | PhaseMM, SCSIReset | ParErr |
| BusFree | 0x9 | Bsy signal cleared | PhaseMM, SCSIReset | |
| EnParChk | 0xA | 3 clock cycles after write | No interrupts | |
| DisParChk | 0xB | 3 clock cycles after write | No interrupts | |
| EnReSel | 0xC | 3 clock cycles after write | No interrupts | |
| DisReSel | 0xD | 3 clock cycles after write | No interrupts | |
| RstMESH | 0xE | 3 clock cycles after write | No interrupts | |
| FlushFIFO | 0xF | 3 clock cycles after write | No interrupts | |

These commands are described in Table 290 on page 151. SCSI output signals are desrcibed in *ANSI X3.131-199x Rev.10L,1992 (SCSI-2): Information Technology - Small Computer System Interface* [13].

Table 290. MESH SCSI Commands

| **Arbitrate Command (‘1’X)** |
|---|

Table 290. MESH SCSI Commands *(Continued)*

Software can use the value in the SourceId register to perform SCSI bus arbitration. The Arbitrate command makes sure that the Bsy and Sel signals have been false for 1200 ns before asserting its encoded SCSI ID along with the Bsy signal. After asserting the Bsy signal, the controller shall wait for 2400 ns and then compare the IDs on the bus. If SourceID is the highest, the controller asserts the Sel signal and wins arbitration. If some other device has a higher ID or has asserted the Sel signal, then the controller has lost arbitration and shall release its Bsy signal and arbitration ID. At this point the ArbLost bit shall be set in the Exception register and an interrupt shall be generated. It is recommended that software load the DestinationID register with the initiator ID before commencing arbitration.

**Selection Command**

In response to the Selection command, the controller shall assert the ID indicated in the DestID register (along with SourceID) and then deassert the Bsy signal. The controller shall wait up to the time indicated in the SelTimeOut register (nominally 250 ms) for the Bsy signal to be asserted by the destination device. The Selection command assumes that the immediately prior command was the Arbitrate command and that the arbitration was won. If the Atn bit is set the Atn signal shall be asserted prior to releasing the Bsy signal indicating Selection phase. If the TMode bit is set, the IO bit shall be set and the reselection phase shall be entered

**Information Transfer Commands**

The Cmd, Status, DataIn, DataOut, MsgIn, and MsgOut commands execute until either the Transfer Count is exhausted, a phase mismatch occurs, or an error occurs. In the Initiator mode of any information transfer command, the Ack signal is left asserted until a subsequent command implies that it must be negated. This behavior is helpful in transitioning to the MessageOut phase or generally responding to messaging or data parity errors. An Information Transfer command can be overwritten at any time. If the TMode bit is set, the MESH controller shall set the Msg, CD, and IO signals to appropriate states and begin the Req/Ack action. In target mode, the Atn bit in the Sequence register is used as a phase comparison bit At the end of any Async data tranfer, when the transfer counter counts down to zero, the Ack signal shall remain low. This is for a variety of reasons—the host may choose to force a disconnect or there may have been a parity error. The Atn signal may need to be asserted prior to the trailing edge of Ack to go into MessageOut phase and to keep the Target from going into Synchronous data transfer before the Sync parameters are set up in the MESH controller. In general, Ack is asserted after the command completes. The Ack signal shall be negated upon issuance of the next command. Synchronous data transfer mode is enabled only during DataIn or DataOut phases. The MESH controller shall stay in synchronous mode between synchronous DataIn or DataOut commands. The Ack signal shall not hang in synchronous mode. For further information about synchronous data transfers, see Section 12.5.14.1, “Synchronous Data Transfers,” on page 157.

**BusFree Command**

The pseudo-phase BusFree command is used to indicate that the bus is expected to transition to the Bus Free phase, which is defined as the Bsy and Sel signals both false. In Initiator mode this implies that the Ack signal be negated first and then Bsy inspected to detect when it goes false. If the MESH controller detects Req asserted before Bsy goes false, then the PhaseMM bit shall be set. If the TMode bit is set, the MESH controller shall release the Bsy signal from the bus.

**RstMESH Command**

The RstMESH command is used to reset the MESH chip through software. Interrupts are masked by this command, so no interrupts are generated. The CmdDone signal shall be active until cleared through the Interrupt register. Note: This command does not cause a SCSI bus reset; this can be done only by direct manipulation of the Status1 register.

**EnParChk Command**

The EnParChk command enables parity checking on data transfers. Parity is always checked on reselection phases and generated on data transfer phases. No interrupts are generated by this command.

**DisParChk Command**

The DisParChk command disables parity checking on data transfers. Parity is always checked on reselection phases and generated on data transfer phases. No interrupts are generated by this command.

**EnReSel Command**

If the TMode bit is false, the EnReSel command enables reselection as a host. It is active until a SCSI bus reset occurs, the DisResel command is issued, or the controller is reselected as a Host. No interrupts are generated from issuing this command, but the Reselected Exception bit is set if the MESH controller is reselected as a host. If the TMode bit is set when this command is issued, it enables the MESH controller to be selected as a Target.

**DisReSel Command**

Table 290. MESH SCSI Commands *(Continued)*

| |
|---|
| The DisReSel command disables selection or reselection as a host or target. Like the EnReSel command, depending on the state of TMode, it disables either reselection as a Host or selection as a Target. No interrupts are generated from this command. |

## 12.5.5 Bus Status Register 0 (Status0)

Table 291. Bus Status Register 0 (Status0) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x04 | R/W* | 0x00 | 0x00 |

* During “read”, the resulting data state corresponds to an OR of the register bit with the SCSI signal state.

The Bus Status registers represent the state of the SCSI bus when read, and can be used to assert signals when written. Normally, these registers are only read to obtain the state of the bus. These registers can be written only when the Exception register is clear. In order to clear a register bit and its corresponding signal, a zero must be written to the status register. The meanings of the bits in Bus Status0 and Bus Status1 are given in Table 292 on page 153.

Table 292. Bus Status Register

| Bit | Status0 Function: |
|---|---|
| 7 | Req32 |
| 6 | Ack32 |
| 5 | Req |
| 4 | Ack |
| 3 | Atn |
| 2 | Msg |
| 1 | CD |
| 0 | IO |
| | Status1 Function: |
| 7 | Rst |
| 6 | Bsy |
| 5 | Sel |
| 4-0 | reserved |

The Req32 and Ack32 signals are reserved for potential use in systems that support 32-bit-wide SCSI.

Writing these registers directly may cause illegal bus phases and lost or corrupted data.

## 12.5.6 Bus Status Register 1 (Status1)

Table 293. Bus Status Register 1(Status1) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x05 | R/W | 0x00 | 0x00 |

See “Bus Status Register 0 (Status0)” on page 153.

## 12.5.7 Bus FIFO Count Register (FifoCount)

Table 294. Bus FIFO Count Register (FifoCount) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x06 | R/W | 0x00 | 0x00 |

The lower 5 bits of the FIFO Count register (4:0) are continually updated to represent the number of bytes in the FIFO. This register is synchronized with the processor interface so that it does not have to be protected from bounce.

Except for the case of synchronous data in (see section 11.4.1), for a data transfer command to be considered done, both the XferCount register and the FIFOCount must be 0. For example, if 6 bytes of CDB (command description block) are put into the FIFO and the transfer count register is not 0, the MESH controller shall expect to get XferCount more bytes before finishing. In this case, the XferCount register shall be set to zero before issuing the Cmd sequence command.

## 12.5.8 Exception Register (Exception)

Table 295. Exception Register 0High (Exception) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x07 | R/W | 0x00 | 0x00 |

The Exception register contains information about conditions which are not usually errors, but can cause the sequence to stop for processor intervention. It is cleared by writing the register with ones in the bits to be cleared or by writing a one to the exception bit in the Interrrupt register.. The meanings of the bits in the Exception register are given in Table 296 on page 154.The bits in the Exception register are discussed in the next sections.

Table 296. Exception Register

| Bit | Function |
|---|---|
| 7-6 | Reserved |
| 5 | **SelWAtn:** The SelWAtn bit is set if the MESH controller was selected as a Target and the Atn signal on the SCSI bus was asserted. |
| 4 | **Selected:** The Selected bit is set if the the controller is selected as a Target and the Atn signal was not asserted at the time of the selection. When this bit becomes true, the SCSI bus IDs of the target and host are in the last byte of the FIFO. |
| 3 | **Reselected:** The Reselected bit is set if the the controller is reselected as a Host. When this bit becomes true, the SCSI bus IDs of the target and host are in the last byte of the FIFO. |
| 2 | **ArbLost:** The ArbLost bit indicates that arbitration was lost when executing the Arb command. This exception may represent one or more lost arbitrations, depending on the hardware implementation. |
| 1 | **PhaseMM:** The Phase Mismatch (PhaseMM) bit shall be set when the pending phase command expects a certain phase but another phase is driven by the Target. When this bit is set, the software must interrogate the bus Status0 register to determine the current phase of the bus and take action appropriately. |
| 0 | **SelTO:** The SelTO bit indicates that the time indicated by the SelectionTO register was exhausted prior to the Destination device being selected. |

## 12.5.9 Error Register (Error)

Table 297. Error Register (Error) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x08 | R/W | 0x00 | 0x00 |

The Error register contains bits that indicate true error conditions. This register is cleared by writing ones to the bits to be cleared or by writing a one to the Error bit in the Interrupt register. The meanings of the bits in the Exception register are given in Table 298 on page 155.

Table 298. Error Register

| Bit | Function |
|---|---|
| 7 | Reserved |
| 6 | **UnExpDisc:** The UnExpDisc bit indicates that the target released the Bsy signal during the time interval between successful selection and the issuing of the BusFree command. |
| 5 | **SCSIRs:** The SCSIRst bit indicates that the Rst signal was (or is) asserted. After receiving this interrupt, the SCSI bus Rst signal (bit 7 of Bus Status1) must be polled until the Rst signal is deasserted. |
| 4 | **SeqErr:** The Sequence Error bit indicates that a command was issued to the MESH controller while an Exception or error interrupt was pending. |
| 3-0 | **ParityErr:** The ParityError0 bit is set if the calculated parity of the incoming data does not match the Parity0 bit on the SCSI bus. The ParityErr3..ParityError1 bits are reserved for future use with up to 32-bit SCSI implementations. |

The bits in the Error register are discussed in the next sections.

## 12.5.10 Interrupt Mask Register

Table 299. Interrupt Mask Register (InterruptMask) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x09 | R/W | 0x00 | 0x00 |

The Interrupt Mask register is used to mask out interrupts from any or all of the interrupt sources. The value of the register is all zeros upon reset. A value of 0 in a particular location masks (disables) the corresponding interrupt source. A value of 1 enables interrupts from the source. The meanings of the bits in the Interrupt mask register are given in Table 300 on page 155. A set condition in the mask corresponds to “masking’ the interrupt.

Table 300. Interrupt Mask Register)

| Bit | Function |
|---|---|
| 7-3 | Reserved |
| 2 | ErrMask |
| 1 | ExcMask |
| 0 | CDoneMask |

## 12.5.11 Interrupt Register

Table 301. Interrupt Register (Interrupt) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x0A | R/W | 0x00 | 0x00 |

The Interrupt register combines the Error, Exception, and CmdDone status bits into one interrupt source. The meanings of the bits in the Interrupt register are given in Table 302 on page 156.

Table 302. Interrupt Register

| Bit | Function |
|---|---|
| 7-3 | Reserved |
| 2 | **Error:** The Error bit represents an OR-combination of all the bits in the Error register. It can be cleared by writing ones to the appropriate bits in the Error register or by writing 1 to this bit in the Interrupt register. |
| 1 | **Exception:** The Exception bit represents an OR-combination of all the bits in the Exception register. It can be cleared by writing ones to the appropriate bits in the Exception register or by writing 1 to this bit in the Interrupt register. |
| 0 | **CmdDone:** The CmdDone bit indicates that the last command issued is done. This bit is automatically cleared by issuing another command to the Command register. It can also be cleared by writing 1 to this bit in the Interrupt register.After a reset action, the CmdDone signal shall be asserted. Because the Interrupt Mask register is cleared on reset, no interrupts shall be generated. |

The bits in the Interrupt register are discussed in the next sections.

## 12.5.12 SourceID Register

Table 303. SourceID Register (SourceID) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x0B | R/W | 0x00 | 0x00 |

The SourceID register contains the SCSI bus ID that is used for arbitration. It is also the register that the MESH controller shall respond to during selection phase (in Target mode) or reselection phase (in Initiator mode).

## 12.5.13 DestinationID Register

Table 304. DestinationID Register (DestinationID) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x0C | R/W | 0x00 | 0x00 |

The DestinationID register contains the SCSI bus ID of the device to be selected or reselected.

## 12.5.14 SyncParms Register (SyncParms)

Table 305. SyncParms Register (SyncParms) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x0D | R/W | 0x00 | 0x00 |

The SyncParms register contains two fields, as shown in Table 306 on page 157.

Table 306. SyncParms

| Bit | Function |
|---|---|
| 7-4 | **Sync Offset:** The SyncOffset field contains the depth to which the FIFO is capable of accepting synchronous data without delay. In the the case of MESH, this value may be from 1 to 15. If it is zero (as upon reset) then all tranfers shall be done asynchronously. See the ANSI *X3T9.2 SCSI-3 SCSI Parallel Interface (SPI)* specification for the complete definition of Synchronous Data transfer mode and the corresponding Synchronous Data Transfer Request (SDTR) and related negotiations that must be done before Synchronous Data mode is enabled. When the Synchronous Offset value in the SyncParms register is nonzero, the value in the SyncPeriod field is used to specify the period of the Ack signal output. When the Synchronous Offset value is 0, Asynchronous transfer mode is specified. In addition, the value in the SyncPeriod field specifies the minimum number of clock cycles between the negation of Ack and the time when the Req signal should be evaluated. Because of the dual-ranked synchronizers in hardware, the actual time may be greater by as much as 1 clock cycle. For example, if the SCSI bus is known to settle in 80 ns, a value of 4 in the SyncPeriod field shall guarantee that Req or Ack is not evaluated in the state machine for at least 80 ns. |
| 3-0 | **Sync Period:** The SyncPeriod field contains the period of the Req or Ack pulse when the MESH controller is in Synchronous Data transfer mode. The value must be adjusted to match the SyncPeriod time agreement in the SDTR messages. Technically, the ANSI SCSI specification allows resolution down to 4 ns, but the driving software must negotiate a period that is acceptable to both the controller and the Target device. A value of 0 indicates 10 MB per second and 3 indicates 5 MB per second. The equation for determining the period of the synchronous Ack signal is (4*clk + 2*clk*SyncPer) where *clk* represents one MESH Clk period and SyncPer is the value in the lower nybble of the SyncParms register. Fast synchronous operation is a special case, where 0 means 100 ns. For Asynchronous transfer modes, the SyncPeriod field must be set to a value of 2 or greater. This field is used to mask the trailing edge of the Req signal so that the internal logic does not incorrectly respond to SCSI bus noise created by cable reflections and other impedance mismatch side-effects. The SyncPeriod value represents the number of clocks delay after the trailing edge of Req before evaluating the Req signal again. The minimum value in the SyncPeriod field is 2 for asynchronous transfers. Anything less results in shorter setup times for data relative to Ack on writes. Achieving the fastest Asynchronous transfer rate with the MESH controller requires a SyncPeriod value of 2 |

### 12.5.14.1 Synchronous Data Transfers

Observe these cautions when performing synchronous data transfers:

- The MESH controller is not able to handle unsolicited synchronous data. Before the issuance of a command, the SyncParms register must be set to match the expected transfer mode.
- The count for the Command Descriptor Block may not be arbitrarily set to a value higher than the number of bytes sent. The CmdDone bit will get set with a corresponding Exception indicating a phase mismatch (the Target went from CommandPhase to Data phase unexpectedly). If the Target then begins to transfer data in synchronous mode, the MESH controller will not be able to handle the Req signals or any corresponding incoming data.
- Target mode does not support synchronous data transfers, regardless of the SyncParms value.

### 12.5.15 MESH ID Register (MeshID)

Table 307. Mesh ID Register (MeshID) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x0E | R/W | 0x00 | 0x00 |

The MESH ID register is used to indicate the version of the MESH controller and also to differentiate MESH from other controllers. This register is read-only and contains two sufields. Bits 7-5 are reserved for definition of vendor ID and should be ignored by software. Bits4-0 correspond to funtionality version ID and are used by software to determine the features and bug-fix level.

### 12.5.16 Selection TimeOut Register (SelTO)

Table 308. Selection TimeOut Register (SelTO) Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Wrt Characteristics | Hardware Reset Value | Software Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | reg[1] | 0x0F | R/W | 0x00 | 0x00 |

The Selection TimeOut register is used for selection and reselection timeouts with its 8-bit value representing the amount of time before the selection timeout is reached. Each count represents 10 ms when the controller is running at 50 MHz.

## 12.6 DBDMA Registers

This section briefly mentions the DBDMA registers for the MESH SCSI controller which are shown in Section 12.3, “Mesh SCSI Open Firmware Properties,” on page 148 and documented in Chapter 15, “Descriptor-Based DMA,” on page 171. The registers are the portion of the MESH SCSI Controller used for controlling bus master data transfers to/from system memory and are named:

1. DBDMA Transmit Channel Control
2. DBDMA Transmit Channel Status
3. DBDMA Interrupt Select
4. DBDMA Branch Select
5. DBDMA Wait Select

**Requirements:**

**12–6.** For the device described herein, all the registers and their associated function defined in Appendix A must be implemented precisely as described.

## 12.7 References

1. *ANSI X3.131-199x Rev.10L,1992 (SCSI-2): Information Technology - Small Computer System Interface* [13]

2. *Macintosh Technology in the Common Hardware Reference Platform* [18] (Use this document for specific physical, timing, and electrical requirements)
3. *Mac IO* [14] CHRP Open Firmware Bindings

# Graphics

# 13

## 13.1 CHRP Requirements

Table 309 on page 161 is an excerpt from *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*, showing the minimum requirements of a graphics subsystem on CHRP platforms.

Table 309. Summary of Minimum Platform Requirements (excerpt)

| Subsystem | Specification | Portable | Personal | Server | Description |
|---|---|---|---|---|---|
| Graphics | | R | R | O | Servers do not require graphics during normal operation and need not support a graphics subsystem. |
| | 1024x768 | O | R | O | Portables may provide screen resolution in accordance with current state-of-the-art LCD technology. |
| | Bi-Endian | R | R | O | |
| | 640x480x8 LFB | R | R | O | |
| | VGA | O | O | O | Some operating environments will prefer platforms that provide hardware support of VGA for performance considerations. |

The CHRP Specification requires that graphics subsystems provide both Little Endian and Big Endian interfaces to frame buffers. *Portable* and *Personal* CHRP platforms must minimally provide a 640x480x8bit linear frame buffer (LFB) for use by the operating system (OS) until such time that the OS installs the drivers necessary to fully operate the video subsystem. It is recommended that systems provide Video Graphics Array (VGA) capability; certain emulation and DOS environments require hardware VGA support for performance reasons.

## 13.2 Bi-Endian Interface

CHRP platforms support both Big Endian (BE) and Little Endian (LE) operating systems and applications. Requiring software to meet a single interface regardless of its own endianess is impractical due to the pervasiveness of the changes that would be required, not only to the OS, but perhaps even applications. There are also environments that may require concurrent BE/LE access to the graphics controller and its frame buffer. An example of this might be a system with a BE operating system with some LE frame grabber hardware on the bus. The graphics controller must respond correctly to both of these interfaces in a performance sensitive manner.

**Requirements:**

**13–1.** Graphics controllers must provide both a Big Endian (BE) and a Little Endian (LE) interface to the frame buffer.

**Hardware and Software Implementation Note:** An endian-neutral aperture may be provided for the control interface, however, if there are control registers greater than a single byte in size, separate BE and LE interfaces are recommended. The capability to simultaneously access the BE/LE apertures (dual apertures) is recommended, although a single aperture under mode control is acceptable.

## 13.3 Linear Frame Buffer

A Linear Frame Buffer (LFB) is a linear range of System Memory or Peripheral Memory Space designated to correspond to the two dimensional arrangement of pixels on the display device, where successive memory bytes are assumed to correspond to successive pixels, possibly with undisplayed bytes at the end of each scan line.

The **`fb8`** generic *frame buffer support package* implements the display device low level interfaces for frame buffers with 8 bits per pixel. **`fb8`** provides routines such as blink/erase/reset/invert screen, insert/delete characters/lines, toggle cursor, and draw-logo (see *IEEE Std 1275-1994, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [9]).

The *8-bit Graphics Extension* adds simple graphics methods to the frame buffer's drivers, including pixel addressable reading and writing of rectangular sections of the frame buffer. Methods are also provided for color-table programming (see *Open Firmware Recommended Practice: 8-bit Graphics Extension* [34]).

The *16-color Text Extension* adds new FCodes that can be used by the **`fb8`** support package for displaying color text. The display support package includes the property named "**`iso6429-1983-colors`**" to indicate the presence of these extensions. The ISO 6429-1983 standard forms the basis of the 16-color model used by this extension. See *Open Firmware Recommended Practice: 16-color Text Extension* [35].

**Requirements:**

**13–2.** CHRP platforms must minimally support a 640x480 linear frame buffer with the Open Firmware characteristics and interfaces described in the *CHRP Linear Frame Buffer Device Binding*.

## 13.4 Video Graphics Array (VGA)

**Requirements:**

**13–3.** VGA compatibility is optional; if VGA compatibility is provided, it must adhere to the programming model given in Appendix A, "VGA Programming Model," on page 195.

**13–4.** If VGA compatibility is provided, it must conform with the characteristics and interfaces described in the *CHRP VGA Display Device Binding*.

**13–5.** If VGA compatibility is provided, the display node of the Open Firmware device tree must indicate a compatible property of "`pnpPNP,900`". Conversely, if the compatibility property of "`pnpPNP,900`" is indicated in the display node of the Open Firmware device tree, the programming model described in Appendix A, "VGA Programming Model," on page 195 must be present.

## 13.5 References

1. *IEEE Std 1275-1994, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices* [9]

2. *Open Firmware Recommended Practice: 8-bit Graphics Extension* [34]

3. *Open Firmware Recommended Practice: 16-color Text Extension* [35]

4. *CHRP Linear Frame Buffer Display Device Binding to: IEEE 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [36]

5. *CHRP VGA Display Device Binding to: IEEE 1275-1994 Standard for Boot (Initialization, Configuration) Firmware* [37]

# Versatile Interface Adapter (VIA) 14

## 14.1 CHRP Requirements

Table 310 on page 165 is an excerpt from *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*, showing the minimum requirements for VIA on CHRP platforms.

Table 310. Summary of Minimum Platform Requirements (excerpt)

| Subsystem | Specification | Portable | Personal | Server | Description |
|---|---|---|---|---|---|
| VIA | | R | R | M | |

## 14.2 VIA Open Firmware Requirements

**Requirements:**

**14–1.** The Open Firmware device tree node for the device described in this chapter must follow the specifications given in the *Mac I/O* [14] Open Firmware device binding.

**14–2.** The Open Firmware "**`reg`**" property information given in Table 311 on page 167 must be provided in the Open Firmware device tree for the VIA device.

## 14.3 Introduction

### 14.3.1 Legacy Application

Macintosh computers have used Versatile Interface Adapter (VIA) integrated circuits to provide a variety of control and interface functions. The VIA contains internal control and data registers that can be accessed by software to configure and determine the state of the VIA and the signals that the VIA monitors.

**Hardware and Software Implementation Note:** VIA hardware will be removed in later versions of this platform specification. It is included here and in initial platforms to support legacy Mac OS code.

**Requirements:**

**14–3.** New capabilities in operating systems must not depend in any way on the continued existence of the facilities described in the VIA chapter.

### 14.3.2 Internal Timings

VIA functions synchronize to an internally generated clock of period = 1.27655 µs; most normal operations will require at least this amount of time to complete. MacOS has used this fact to insert delays of known duration into the execution of timing-critical code. This practice *must not be proliferated* since the VIA function will be removed from this platform definition in the future.

**Requirements:**

**14–4.** New software must not utilize the timing characteristics associated with operations on the VIA hardware as described in Section 14.3.2, “Internal Timings,” on page 166 of the VIA chapter.

## 14.4 VIA Registers

**Requirements:**

**14–5.** All the “**`reg`**” property fields specified in Table 311 on page 167, “VIA Registers”, must comply with the format given in the PCI binding to IEEE Std 1275-1994, as modified by the CHRP binding to IEEE Std 1275-1994.

The positions in the “**`reg`**” property (denoted by reg[1], reg[2], etc.) and the value of the size fields are stated in Table 311 on page 167. The format is described in the referenced Open Firmware bindings.

Table 311. VIA Registers

| Address Offsets | Registers | Reg Properties |
|---|---|---|
| 0x0000 | Data Register B | reg[1],size=7680 |
| 0x0200 | Data Register A | |
| 0x0400 | Data Direction Register B | |
| 0x0600 | Data Direction Register A | |
| 0x0800 | Timer T1 counter (low-order byte) | |
| 0x0A00 | Timer T1 counter (high-order byte) | |
| 0x0C00 | Timer T1 latch (low-order byte) | |
| 0x0E00 | Timer T1 latch (high-order byte) | |
| 0x1000 | Timer T2 counter (low-order byte) | |
| 0x1200 | Timer T2 counter (high-order byte) | |
| 0x1400 | Shift Register | |
| 0x1600 | Auxiliary Control Register | |
| 0x1800 | Peripheral Control Register | |
| 0x1A00 | Interrupt Flag Register | |
| 0x1C00 | Interrupt Enable Register | |
| 0x1E00 | Data Register A | |

## 14.4.1 Data Register B

Table 312. Data Register B Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register when OF passes control to OS | SW Initialized Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | base_adr[0] | 0x0000 | Read/Write | Undefined | N/A |

There are two data registers in each VIA, called *Data register A* and *Data register B,* each with its own data direction register. The Data Register bit definitions are system dependent. They are 8-bit bidirectional buses used for transfer of data, control, and status between the VIA and a peripheral device. Table 313 shows the bit assignments for Data Register B.

Table 313. Data Register B Bit Definition

| Bit | I/O | Description |
|---|---|---|
| PB0-PB7 | | **Unused:** If programmed as inputs, these bits will read 0. |

## 14.4.2 Data Register A

Table 314. Data Register A Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register when OF passes control to OS | SW Initialized Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | base_adr[0] | 0x0200 | Read/Write | Undefined | N/A |

There are two data registers in each VIA, called *Data register A* and *Data register B,* each with its own data direction register. The Data Register bit definitions are system dependent. They are 8-bit bidirectional buses used for transfer of data, control, and status between the VIA and a peripheral device. Table 315 shows the bit assignments for Data Register A.

Table 315. Data Register A Bit Definition

| Bit | I/O | Description |
|---|---|---|
| PA0-PA2 | | **Unused:** If programmed as inputs, these bits will read 0. |
| PA3 | Output | **Sync Modem:** Modem clock select:<br>1: Select the external serial clock to drive the SCC's /RTxCA pin<br>0: Select the C3.6864MHz clock to drive the SCC cell<br>**Note:** This is but one implementation of SCC clock selection (platform dependent). |
| PA4 | | **En WaitReqA:** Lets the WaitReq_L signal from port A of the SCC appear on the PA7 input pin. |
| PA5 | | **Unused:** If programmed as inputs, these bits will read 0. |
| PA6 | Output | **En WaitReqB:** Lets the WaitReq_L signal from port B of the SCC appear on the PA7 input pin. |
| PA7 | Input | **SCC WREQ:** Reflects the state of the Wait/Request pins from the SCC. |

### 14.4.3 Data Direction Register B

Table 316. Data Register B Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register when OF passes control to OS | SW Initialized Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | base_adr[0] | 0x0400 | Read/Write | Undefined | N/A |

Table 317. Data Register B Bit Definition

| Bit | Description |
|---|---|
| 0-7 | A bit set as 1 in a data direction register causes the corresponding bit of the data register to be used for output, whereas a 0 causes it to be used for input. The Macintosh Operating System sets up the data direction registers at system start-up or reset. |

### 14.4.4 Data Direction Register A

Table 318. Data Register A Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register when OF passes control to OS | SW Initialized Reset Value |
|---|---|---|---|---|---|---|
| 1 byte | Direct | base_adr[0] | 0x0600 | Read/Write | Undefined | N/A |

Table 319. Data Register A Bit Definition

| Bit | Description |
|---|---|
| 0-7 | A bit set as 1 in a data direction register causes the corresponding bit of the data register to be used for output, whereas a 0 causes it to be used for input. The Macintosh Operating System sets up the data direction registers at system start-up or reset. |

### 14.4.5 Event Timers

The VIA event timers are unused in CHRP compliant platforms.

### 14.4.6 Shift Register

The Shift Register is unused in CHRP compliant platforms.

### 14.4.7 Auxiliary Control Register

The Auxiliary Control register controls various parameters pertaining to the VIA timers and the VIA Shift register, and is unused in CHRP compliant platforms.

### 14.4.8 Peripheral Control Register

The VIA Peripheral Control register allows software to set some very low-level parameters for certain VIA signals. ***Do not write the Peripheral Control register.*** The result of a read operation is undefined.

**Requirements:**

**14–6.** Software must not write to the Peripheral Control Register in the VIA hardware.

### 14.4.9 Interrupt Flag Register

The Interrupt Flag Register (IFR) is used to indicate the presence of interrupt conditions generated internally by VIA functions or from external interrupt conditions on the control input pins.

### 14.4.10 Interrupt Enable Register

The Interrupt Enable Register (IER) is used to enable/disable the interrupts indicated in the Interrupt Flag Register. There is bit in the IER corresponding to each bit in the IFR.

# Descriptor-Based DMA 15

## 15.1 Overview

This chapter defines the hardware and software operation of descriptor-based direct memory access (DBDMA) technology for I/O data transfers.

The DBDMA architecture uses *command descriptors,* which are structured as linked arrays. For the sake of simplicity, the scalability of this model has been limited; for example, there are no mechanisms for sharing DMA devices among processors or for efficiently reporting completion status from large numbers of DMA devices. Another (more complex) model may be needed in platforms containing large numbers of processors or I/O devices. For more information about command descriptors, see Section 15.7, “Commands,” on page 182.

There is a more extensive DBDMA architecture which is documented in the *Macintosh Technology in the Common Hardware Reference Platform* [18]. Only those DBDMA features used by the devices described in this document are documented here.

## 15.2 DBDMA Characteristics

The DBDMA architecture has the following characteristics:

- **Fixed-size commands.** All command entries are 16 bytes long.
- **Fewest processor interrupts.** No more than one interrupt is needed per I/O operation.
- **Initiator-resident data structures.** All command and status list components are expected to be located in initiator-resident system memory
- **Simple command structure.** Descriptors are organized in a simple linear array. Linking can be accomplished through an optional branch action, either associated with a data transfer command or through a NOP command that specifies a branch.
- **Logically distinct channels.** To avoid dynamic usage conflicts, the following distinct DMA channels are provided for controlling transfers of independent data streams:
  - **Duplex channels.** Different DMA channels are provided for logically distinct data-transfer paths, such as full duplex transmit and receive.
  - **Event channels.** A special DMA channel can be provided for reporting asynchronous events from any controller channel.
- **Conditional waits.** DMA commands can be conditionally suspended, based on the DMA controller’s internal Channel Status bits. Since external access to certain of these state bits is provided, they can be used to temporarily suspend DMA activity until a flow-control conflict is resolved.

- **Embedded branches.** To minimize command latency in channel programs that require branching, the INPUT, OUTPUT, and NOP commands have a conditional branch capability.

## 15.3 Conventions

The discussion of DBDMA in this specification uses the conventions described in this section.

- The term *descriptor* refers to the DMA command list elements, which are often simply buffer descriptors.
- The term *DMA* indicates that data transfers are performed by a relatively simple state machine (called the *target*) that processes commands generated by a relatively sophisticated processor (called the *initiator*).
- Since the devices defined herein that use DBDMA are PCI devices, only little endian addressing for DBDMA is supported. The little-endian variant of the DBDMA architecture is defined by the ordering of bytes in any quadlet register within the DMA unit or the command/status memory locations it accesses. For such quadlets, the data byte with the smallest address is the least significant byte.
- The formats of storage locations, field values, and DMA-command constants are defined in this appendix. To minimize confusion when these are referenced, a consistent nomenclature is used throughout. This nomenclature is defined in Table 320 on page 172.

Table 320. Name notation examples

| Name | Description |
|---|---|
| **CommandPtr** | Register or element of a memory-resident data structure<br>**Explanation:** In names of registers and memory locations, the first letter is capitalized. For multiple-component names, like CommandPtr, the first letters of additional run-together words are also capitalized. |
| commandDependent | Name of a field within a register or memory-resident data structure<br>**Explanation:** In names of fields, the letters within the first word are not capitalized. For multiple-component field names, like commandDependent, the first letters of additional run-together words are also capitalized. |
| Command.field | Name of a field within a register or memory-resident data structure |
| CMD_VALUE | Name of a defined constant value<br>**Explanation:** In names of defined constants, all letters within the name are capitalized. For multiple-component field names, like OUTPUT_MORE, underscore characters separate the name components. |

**Requirements:**

**15–1.** Every DBDMA reserved field must be 0 when written and must be ignored when read. This must be true whether the reserved field is accessed by the DBDMA hardware or by device driver software.

**15–2.** Some address pointers are constrained to be multiples of 4, 8, or 16 bytes. For example, the NewCommandPtr value is always 16-byte aligned. The least-significant bits of these aligned addresses must be reserved, to ensure deterministic behavior when unaligned address values are used.

# 15.4 Controller Registers

## 15.4.1 Register Organization

Controllers using the DBDMA architecture described herein have two distinct sets of registers:

- Channel Registers: These are the DBDMA registers that are described in this appendix.
- Device Registers: These are registers which are device specific and are not described in this appendix. Refer to the specific devices which use DBDMA for definitions of these registers.

Table 321 shows the offset for each of the channel registers. The offset value appears literally in the Command address field when the register is accessed indirectly (Command.key = KEY_REGS). When accessed directly by the host, or by a channel command (Command.key = KEY_SYSTEM) the register's address is the sum of the system-defined channel base address and the register's offset value.

Table 321. Channel registers

| Offset | Register | Required /optional | |
|---|---|---|---|
| 0x00 | ChannelControl | Required | ChannelControl is used to control the activity of the channel. |
| 0x04 | ChannelStatus | Required | ChannelStatus is used to observe the activity of the channel. |
| 0x08 | Reserved | | |
| 0x0C | CommandPtrLo | Required | CommandPtrLo is used to indicate where the channel's command list is located in memory. |
| 0x10 | InterruptSelect | Optional: See chapter on applicable device | |
| 0x14 | BranchSelect | Optional: See chapter on applicable device | |
| 0x18 | WaitSelect | Optional: See chapter on applicable device | |

The initial values, read values, and write effects for the channel registers are shown in Table 322 on page 173.

Table 322. DBDMA register summary

| Register Name | Initial Value | Read Value | Write Effect |
|---|---|---|---|
| ChannelControl | n.a. | 0x00000000 | update selected bits |
| ChannelStatus | 0x00000000 | ChannelStatus | ignored |
| CommandPtrLo | undefined | CommandPtrLo | store if idle |
| InterruptSelect | undefined | InterruptSelect | store |
| BranchSelect | undefined | BranchSelect | store |
| WaitSelect | undefined | WaitSelect | store |

Detailed descriptions of the full register set are contained in the next sections.

## 15.4.2 ChannelControl Register

Table 323. ChannelControl Register Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register When OF Passes Control to OS | S/W Initiated Reset Value |
|---|---|---|---|---|---|---|
| 4 bytes | Direct | See chapter on device using DBDMA channels | 0x00 | Read/Write | Undefined | N/A |

The format of the ChannelControl register is shown in Table 324. The ChannelControl register serves as the writing port for bits that are presented in the ChannelStatus register..

Table 324. Channel Control Register

| Bit | Function |
|---|---|
| 31 -16 | Channel Control Mask: The ChannelControl.mask field selects which of the lower 16 bits are to be modified by a write action. Bits in the lower half of the ChannelControl register are written only if the corresponding bits in ChannelControl.mask are set. |
| 15-0 | Channel Control Data: The value in the ChannelControl.data field is conditionally written to the ChannelStatus register, based on the ChannelControl.mask field. Note that certain bits of the ChannelStatus register are not writeable or should be written only to a set state or to a cleared state. See Section 15.4.3 for full details. |

## 15.4.3 ChannelStatus Register

Table 325. ChannelStatus Register Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register When OF Passes Control to OS | S/W Initiated Reset Value |
|---|---|---|---|---|---|---|
| 4 bytes | Direct | See chapter on device using DBDMA channels | 0x04 | Read/Write | Undefined | N/A |

The read-only ChannelStatus register provides access to the channel's internal status bits. The ChannelControl register bits are also visible through the ChannelStatus register. A write to the ChannelStatus register shall be ignored. The format of the ChannelStatus register is shown in Table 326 on page 174.

Table 326. Channel Status Register

| Bit | Function |
|---|---|
| 31 -16 | Reserved |

Table 326. Channel Status Register *(Continued)*

| Bit | Function |
|---|---|
| 15 | ChannelStatus.run: ChannelStatus.run is set to 1 by software to start execution by the channel. This should be done only after the CommandPtr registers are initialized; otherwise, the operation of the channel is undefined. Software can clear this bit to 0 to abort the operation of the channel. When channel operation is aborted data transfers are terminated, status is returned, and an interrupt is generated (if requested in the Command.i field of the command descriptor). Data that is stored temporarily in channel buffers may be lost. |
| 14 | ChannelStatus.pause: ChannelStatus.pause is set to 1 by software to suspend command processing. Hardware responds by suspending data transfers and command execution and then clearing the ChannelStatus.active bit. Software must reset ChannelStatus.pause to 0 to allow command processing to resume. |
| 13 | ChannelStatus.flush: ChannelStatus.flush can be set to 1 by software, to force a channel that is executing an INPUT_MORE or INPUT_LAST command to update memory with any data that has been received from the device but has not yet been written to memory. When the memory update is complete, hardware updates the xferStatus and resCount fields in the current memory resident channel command, then clears the flush bit. Thus, a partial status update is characterized by 1 in the flush bit of the xferStatus field and a final status update is characterized by 0. |
| 12 | ChannelStatus.wake: ChannelStatus.wake is set to 1 by software to cause a channel that has gone idle to wake up, refetch the command pointed to by CommandPtr, and continue processing commands. The channel becomes idle after executing a STOP command. The STOP command does not increment the CommandPtr register.ChannelStatus.wake shall be reset to 0 by hardware immediately after each command fetch. |
| 11 | ChannelStatus.dead: ChannelStatus.dead is set to 1 by hardware when the channel halts execution due to a catastrophic event such as a bus or device error. The current command is terminated and hardware attempts to write status back to memory. Further commands are not executed. If a hardwired interrupt signal is implemented, the controller shall generate an unconditional interrupt. When hardware sets the ChannelStatus.dead bit, the ChannelStatus.active bit is simultaneously deasserted. Hardware shall reset ChannelStatus.dead to 0 when the ChannelStatus.run bit is cleared by software. |
| 10 | ChannelStatus.active: ChannelStatus.active is set to 1 by hardware in response to software setting the ChannelStatus.run bit to 1. ChannelStatus.active is reset to 0 by hardware in response to software clearing the ChannelStatus.run bit or setting the ChannelStatus.pause bit. It is also reset to 0 after a STOP command is executed or when hardware sets the ChannelStatus.dead bit to 1. |
| 9 | Reserved |
| 8 | ChannelStatus.bt: ChannelStatus.bt is set by hardware at the completion of the NOP, INPUT, and OUTPUT commands to indicate whether a branch has been taken. Branching is governed by the Command.b field in the command descriptor, the ChannelStatus.s7..s0 bits, and the values in the mask and value fields of the BranchSelect register. The presence of this bit in the Command.xferStatus field allows software to follow the actual channel program flow with minimal overhead. |
| 7-0 | ChannelStatus.s7-s0: Each DMA channel has up to eight general purpose state bits (ChannelControl.s7 .. ChannelControl.s0) that can be written through the ChannelControl register and read through the ChannelStatus register. In some channel implementations, it may be desirable to provide an additional method for setting and clearing these bits directly through hardwired connections—for example, when errors occur or to indicate the completion of a logical record. These bits can be tested at the completion of each command to determine if certain actions should be taken. These actions may include generation of a hardwired interrupt signal, suspension of further command processing, and branching to a new location for further command fetches. |

The run and pause bits are control bits that are set and cleared by software. The flush and wake bits are command bits that are set by software and are cleared by hardware when a given action has been performed. The dead, active, and Bt bits are hardware status bits. The bits s7..s0 can be used for general purpose status and control. Their meaning is channel-specific, and they can be controlled by either hardware or software.

## 15.4.4 CommandPtrLo Register

Table 327. CommandPtrLo Register Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register When OF Passes Control to OS | S/W Initiated Reset Value |
|---|---|---|---|---|---|---|
| 4 bytes | Direct | See chapter on device using DBDMA channels | 0x0C | Read/Write | Undefined | N/A |

The and CommandPtrLo register specifies the address of the next command entry to be fetched. Since all channel commands are 16-byte aligned, the four least-significant bits of the CommandPtrLo register must always be written with zeros. If they are written with a nonzero value, the operation of the channel is indeterminate and the value returned when these bits are read is undefined. Table 328 on page 176shows the format of the CommandPtr register.

Table 328. CommandPtrLo

| Bit | Function |
|---|---|
| 31 -0 | Address Aligned on a 16 byte boundary |

The CommandPtrLo registers can be read at any time, but writes to the CommandPtrLo register are ignored unless the ChannelStatus.run and ChannelStatus.active bits are both 0.

## 15.4.5 InterruptSelect Register

Table 329. Interrupt Select Register Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register When OF Passes Control to OS | S/W Initiated Reset Value |
|---|---|---|---|---|---|---|
| 4 bytes | Direct | See chapter on device using DBDMA channels | 0x10 | Read/Write | Undefined | N/A |

The InterruptSelect register is used to generate the interrupt condition bit that is tested at the completion of each command to determine if a hardwired interrupt signal should be asserted. Its format is shown in Table 330 on page 176.

Table 330. Interrupt Select Register

| Bit | Function |
|---|---|
| 31 -24 | Reserved. Undefined on reads. |
| 23-16 | Mask |
| 15-8 | Reserved. Undefined on reads. |

Table 330. Interrupt Select Register *(Continued)*

| Bit | Function |
|---|---|
| 7-0 | Value |

The 8-bit mask and value fields affect the generation of interrupts when command descriptors are processed.

An interrupt condition signal, which can be tested by each channel command, is generated according to the following equation:

```
c = (ChannelStatus.s7..s0 & InterruptSelect.mask)
      == (InterruptSelect.value & InterruptSelect.mask)
```

## 15.4.6 BranchSelect Register

Table 331. BranchSelect Register Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register When OF Passes Control to OS | S/W Initiated Reset Value |
|---|---|---|---|---|---|---|
| 4 bytes | Direct | See chapter on device using DBDMA channels | 0x14 | Read/Write | Undefined | N/A |

The BranchSelect register is used to generate the branch condition bit that is tested at the completion of each command to determine if a branch should be performed. The BranchSelect register has the same format as the InterruptSelect register.

The branch condition signal is generated according to the following equation:

```
c = (ChannelStatus.s7..s0 & BranchSelect.mask)
      == (BranchSelect.value & BranchSelect.mask)
```

## 15.4.7 WaitSelect Register

Table 332. WaitSelect Register Characteristics

| Register Size | Register Addressing | Associated Reg Property | Offset Address | Read/Write Characteristics | Value in Register When OF Passes Control to OS | S/W Initiated Reset Value |
|---|---|---|---|---|---|---|
| 4 bytes | Direct | See chapter on device using DBDMA channels | 0x18 | Read/Write | Undefined | N/A |

The WaitSelect register is used to generate the wait condition bit that is tested at the completion of each command to determine if command execution should be suspended. The WaitSelect register has the same format as the InterruptSelect register.

The wait condition signal is generated according to the following equation:

```
c = (ChannelStatus.s7..s0 & WaitSelect.mask)
      == (WaitSelect.value & WaitSelect.mask)
```

## 15.5 Summary of DBDMA Operations

DBDMA operations are expected to involve an initiator, which is the component that generates commands. In a typical operation, the initiator generates a new sequence of commands and appends them to a preexisting command list in memory. A wakeup signal is then sent to the target, to initiate the target's processing of the recently appended commands. This process is illustrated in the Figure 6 on page 178.

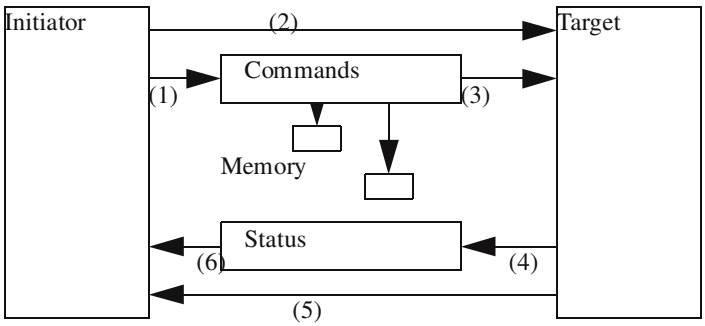


Figure 6. DBDMA Operation

A target is a component that connects to one or more I/O devices such as a disk controller or network interface. When the target is activated, it reads and executes the new commands. As command processing continues, the target returns status information to prespecified locations and sends a signal to the initiator, indicating that another descriptor has been processed.

There are no provisions for having multiple initiators concurrently append additional command entries; multiple initiators are expected to serialize their command list updates through other shared-memory semaphores that are beyond the scope of this specification. The unsharable nature of the command list implies that another architecture may be appropriate when the DMA controller is shared by loosely-coupled multiprocessors.

Costs are reduced by eliminating the needs to support status-queue structures. Status information is simply returned to prespecified locations within command descriptors, which are checked by the processor after an interrupt is received. The mandatory polling of multiple status locations implies that another architecture may be appropriate when large numbers of DMA controllers are shared by one processor.

### 15.5.1 Multiplexed Channels

The DBDMA architecture assumes that each device is connected to its own data-transfer channel. I/O driver software is thus freed from the responsibility of dynamically assigning DMA channels to I/O devices. Two data-transfer

channels would typically be assigned to a full-duplex device, such as Ethernet; one is sufficient for a half-duplex device such as a disk controller.

In most cases, the transfer of control information (such as a seek address) is performed before the data transfer is initiated. Similarly, status information (such as short-transfer counts) is returned after the data transfer finishes. In both cases, the control, data transfers, and control transfers are performed sequentially, so the same DMA channel resources can be shared.

Other forms of status information (such as a disk pack removal) are unexpected and cannot be associated with a data transfer channel. A special event channel may be used to report such asynchronous events.

Since the event channel is rarely used, it may be shared by all devices attached to a DBDMA controller. Hence the controller may contain one shared event channel and one or more device-connected data transfer channels, as shown in Figure 7 on page 179.

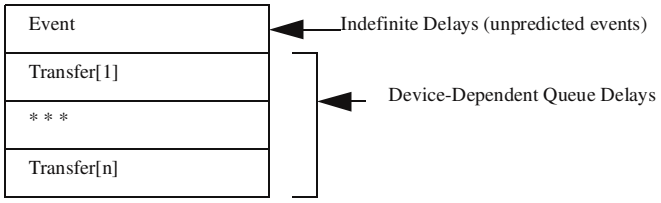


Figure 7. Multiplexed channel types

The commands in the event list are expected to be Read commands that (on demand) return unexpected event status. In this case, the returned list-index value *n* is used by I/O driver software to identify the affected transfer list, transfer[n]. Only one or two Read commands are expected to be queued, since a queued event command is not preallocated to any list but can be used to return event status from any of the affiliated lists. Note that the commands may remain in the event list for an indefinite period of time, depending on the arrival rate of unexpected events.

The commands in the transfer[1] through transfer[n] lists are expected to be used individually (to support half-duplex traffic) or in pairs (to support full-duplex traffic). The commands in these lists are typically transient, in that they are constantly consumed as I/O operations are performed. However, some I/O operations (such as terminal read actions or flow-controlled write actions) may have an indefinite lifetime.

The event channel is optional; it need not be implemented when the interrupt mechanism and device status registers are sufficient to report device-dependent asynchronous events.

## 15.5.2 Command-List Structure

The command list space is allocated by the initiator, which fills a set of command-list entries before passing ownership of them to the target. The target fetches command entries and performs the command-entry-specified data-transfer operation, processing command entries until a STOP entry is reached. The structure of a typical command list is shown in Figure 8.

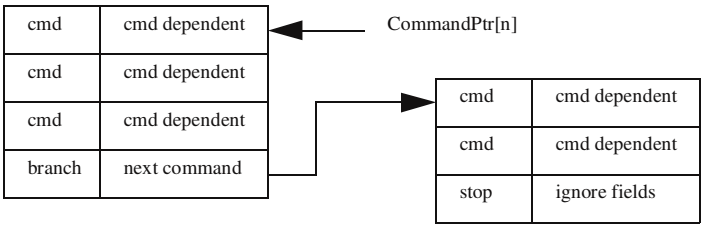


Figure 8. Command list structure

The command list may be structured in several different ways: to loop on itself (a circular queue), to link individual data-transfer operations (linked lists), or to link groups of data-transfer operations (a hybrid approach, as shown in Figure 8).

Note that branches may be embedded in most commands. Therefore, the branch shown in the example could be collapsed into the previous command or it could be encoded as a NOP command that specified a branch. The detailed structure of command lists is dependent on device driver software conventions and is not specified by the DBDMA architecture.

In the idle state, the target has a CommandPtr[n] value for each of the *n* multiplexed channels that are supported. When activated on channel *n,* a command entry is fetched from the physical address specified by CommandPtr[n]. Command fetching and execution normally continue until a STOP command is executed.

The autonomous fetching and execution of multiple commands provides a mechanism to execute different data-transfer operations (command chaining), provide several memory addresses for a single data-transfer operation (scattered memory addressing), or provide several target addresses for a single data-transfer operation (scattered device addressing).

During the execution of a command entry, the CommandPtr[n] value is either incremented by 16, replaced by the command-entry's branchAddress value, or left unchanged (for the STOP command value).

Additional commands may be dynamically appended to an existing command descriptor list while the previously initiated DMA operations are in progress. Command-list appending is performed by creating an additional command list and overwriting the STOP command at the end of the old command list with a NOP or a branch to the first command of the new list.

The WAIT field can be used to provide flow control within a channel. A channel can be programmed to wait until signaled by the processor or by another device.

## 15.6 Design Model

This section presents the DBDMA design model, which illustrates how the DBDMA components are structured

### 15.6.1 Controller Components

A DBDMA controller is expected to connect multiple devices indirectly to a system bus that supports read and write transactions to system memory. Processor interrupts are either generated through special signal lines (for motherboard designs) or by sending wakeup events (a 32-bit write to a prespecified register address).

The DBDMA controller may have shared states, as well as channel states that are replicated for each of several connected devices, as shown in Figure 9 on page 181.

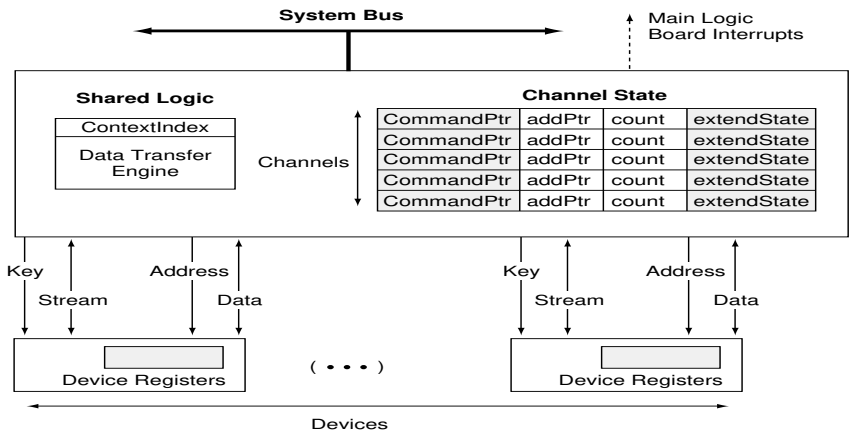


Figure 9. DBDMA controller model

Hardware is expected to use some form of ContextIndex state to identify which of the DMA contexts is currently active, but this state is not directly or indirectly accessible to software. The data-transfer engine provides the facilities for performing the requested data transfers.

The DBDMA architecture provides a well-defined interface to attached I/O devices. An I/O device may be connected to a data-byte stream, where the controller handles the movement of unaddressed data bytes into or out of system memory. The controller provides a 3-bit key value. Four of the Command.key encodings are used to identify which byte-stream is being used. The standard data stream is typically STREAM0 and an optional control/status stream is typically STREAM1.

As an alternative option, a DBDMA controller may support the direct movement of data between device registers and memory. In this case, the controller provides a sequence of device register addresses while the data is being transferred. An additional Command.key encoding is used to identify transfers directed to device registers.

Each channel provides a few memory-mapped registers. These registers include the CommandPtr register (used to initialize the sequencer's pointer to the command list) and the ChannelStatus and ChannelControl registers. Writes to ChannelControl registers can reactivate idle command-list processing. Optional BranchSelect, InterruptSelect, and WaitSelect registers can be used to specify the manner in which the conditional execution of the branch, interrupt, and wait actions is selected.

## 15.6.2 System Bus Errors

Unrecoverable errors on system memory accesses includes parity and addressing errors. A busy-retry error is treated as a recoverable bus error, and the bus action is retried until it finishes successfully or the DBDMA controller's command processing is terminated.

An unrecoverable error sets the dead bit in the ChannelStatus register of the currently executing DMA channel. Software must explicitly reset the ChannelStatus.run bit to 0 and later set it back to 1 to return to an operational state.

### 15.6.3 Device Status

The device's control and status registers are mapped into system memory space, so the processor can access them directly using read and write transactions. For some applications, the processor is required to update these registers at the end of every data transfer, so there is no need for the channel program to return device status explicitly.

However, some devices have the intelligence to process multiple data transfer requests autonomously. For these applications, processor performance would be reduced unless the channel could autonomously return device status before initiating the next data transfer. The DBDMA architecture provides two mechanisms for autonomously returning device status:

- **Status stream.** An additional data-transfer command can be used to transfer a status-byte stream into memory. A unique Command.key value is used to distinguish the status stream from the normal data-transfer stream.
- **Status registers.** An additional data-transfer command can be used to transfer a range of device register addresses into memory. Another unique Command.key value is used to distinguish this memory-mapped transfer from the normal data-transfer stream.

Simple device status information may also be returned in the Command.xferStatus field described in Section Bit.

### 15.6.4 Conditional Actions

The INPUT, OUTPUT and NOP commands provide Command.b and Command.branchAddress fields that specify the condition and address for optional branches in command sequencing. A similar mechanism is used to specify conditional interrupt generation and conditional suspension of command processing.

## 15.7 Commands

All the commands listed herein are mandatory and must be implemented as specified.

### 15.7.1 Command Formats

The general format of a DBDMA command descriptor contains 16 bits of opcode fields (including a 4-bit cmd field), a 16-bit reqCount field, a 32-bit address parameter, and a 32-bit cmdDep parameter, as shown in Table 324 on page 174. The xferStatus and resCount fields are used by the channel to report status after a command finishes. Although the location and size of the cmdDep field is standardized for all commands, its interpretation is dependent on the Command.cmd field value.

Table 333. Command Entry Formatr

| Word | Bit | Function |
|---|---|---|
| Word 3 (Little Endian Byte Ordering) | 31 -28 | cmd |
| | 27 | Reserved |
| | 26-24 | key |
| | 23-22 | Reserved |
| | 21-20 | i |
| | 19-18 | b |
| | 17-16 | w |
| | 15-0 | ReqCount |
| Word 2 | 31-0 | Address |
| Word 1 | 31-0 | CmdDep |
| Word 0 | 31-16 | XferStatus |
| | 15-0 | ResCount |

**Bit 31-28, Word 3 (Command.cmd):** The Command.cmd field specifies which type of data transfer is performed, as defined in Table 334 and described in the following sections.

Table 334. Command.cmd field values

| Value | Name | Description |
|---|---|---|
| 0 | OUTPUT_MORE | Transfer more memory to stream |
| 1 | OUTPUT_LAST | Transfer last memory to stream |
| 2 | INPUT_MORE | Transfer more stream to memory |
| 3 | INPUT_LAST | Transfer last stream to memory |
| 4 | STORE_QUAD | Store immediate 4-byte value |
| 5 | LOAD_QUAD | Load immediate 4-byte value |
| 6 | NOP | No data transfer |
| 7 | STOP | Suspend command processing |
| 8–15 | | Reserved |

**Bit 26-24, Word 3 (Command.key):** The Command.key field is used to select which of the device access ports is used, as specified in Table 335 on page 183.

Table 335. Effects of Command.key field

| Value | Name | Description | Usage |
|---|---|---|---|
| 0 | KEY_STREAM0 | Default device stream | Data |
| 1 | KEY_STREAM1 | Device-dependent stream | Control and status |
| 2 | KEY_STREAM2 | Device-dependent stream | |
| 3 | KEY_STREAM3 | Device-dependent stream | |
| 4 | | Reserved | |
| 5 | KEY_REGS | Channel-state register space | |
| 6 | KEY_SYSTEM | System memory-mapped space | |

Table 335. Effects of Command.key field *(Continued)*

| Value | Name | Description | Usage |
|---|---|---|---|
| 7 | KEY_DEVICE | Device memory-mapped space | |

With INPUT and OUTPUT commands, the key field may specify alternate device streams (e.g. data and status). With LOAD_QUAD and STORE_QUAD commands, the key field specifies to which address space the immediate data is to be transferred. Table 336 on page 184 summarizes the operation of the key field in combination with each of the data transfer commands. A key field value of KEYSTREAM0 through KEYSTREAM3 combined with a LOAD_QUAD or STORE_QUAD command is illegal and generates undefined results.

Table 336. Data transfer operations

| Command | Key | Source | Destination |
|---|---|---|---|
| OUTPUT | KEY_STREAM0..3 | SysMem(address) | Stream 0..3 |
| | KEY_SYSTEM | SysMem(address) | SysMem(Data2Ptr) |
| | KEY_DEVICE | SysMem(address) | Device(Data2Ptr) |
| | KEY_REGS | SysMem(address) | ChannelRegs(Data2Ptr) |
| INPUT | KEY_STREAM0..3 | Stream 0..3 | SysMem(address) |
| | KEY_SYSTEM | SysMem(Data2Ptr) | SysMem(address) |
| | KEY_DEVICE | Device(Data2Ptr) | SysMem(address) |
| | KEY_REGS | ChannelRegs(Data2Ptr) | SysMem(address) |
| STORE_QUAD | KEY_STREAM0..3 | Illegal; see note below | See note below |
| | KEY_SYSTEM | data32 | SysMem(address) |
| | KEY_DEVICE | data32 | Device(address) |
| | KEY_REGS | data32 | ChannelRegs(address) |
| LOAD_QUAD | KEY_STREAM0..3 | Illegal; see note below | See note below |
| | KEY_SYSTEM | SysMem(address) | data32 |
| | KEY_DEVICE | Device(address) | data32 |
| | KEY_REGS | ChannelRegs(address) | data32 |

**Bit 21-20, Word 3 (Command.i):** Upon completion of each command, a hardwired interrupt may optionally be generated. Interrupt generation is controlled by the 2-bit Command.i field in conjunction with an internal interrupt condition bit that is generated by the channel. The channel generates the interrupt condition based on the current values of the ChannelStatus.s7..s0 bits along with mask and data values found in the InterruptSelect register of each channel. The suggested algorithm for generating the interrupt condition is given in Table 337 on page 184.

Table 337. Algorithm for Generating Interrupt

```
c = (ChannelStatus.s7..s0 & InterruptSelect.mask)
== (InterruptSelect.value & InterruptSelect.mask)
```

**Bit** In many implementations there will be no need to allow all of the general purpose status bits to generate interrupts. In these cases, the generation of the interrupt condition may be as simple as tying it to a single status bit or tying it to 0. The equation above should be used only in those cases where it is desired to allow interrupts to be generated based on the values of multiple status bits. Table 338 on page 185 indi-

cates how the interrupt condition bit is used in conjunction with the Command.i field to determine whether or not to generate an interrupt.

Table 338. Effects of Command.i field

| Value | Effect |
|---|---|
| 0 | Never interrupt |
| 1 | Interrupt if the Interrupt Condition bit is set |
| 2 | Interrupt if the Interrupt Condition bit is cleared |
| 3 | Always interrupt |

In implementations that have no hard-wired interrupt signal (such as a DBDMA controller used with certain I/O expansion buses), the Command.i field value is ignored.

**Bit 19-18, Word 3 (Command.b):** Upon completion of each data transfer command, the CommandPtr is updated to point to either the next command or the target of a branch. Branching is controlled by the Command.b field in conjunction with an internal branch condition bit that is generated by the channel. The channel generates the branch condition based on the current values of the ChannelStatus.s7..s0 bits along with a mask and a data value found in the BranchSelect register of each channel. The algorithm for generating the branch condition is given in Table 339 on page 185.

Table 339. Algoritm for Generating Branch Condition

```
c = (ChannelStatus.s7..s0 & BranchSelect.mask)
== (BranchSelect.value & BranchSelect.mask)
```

Table 340 on page 185 indicates how the branch condition bit is used in conjunction with the Command.b field to determine whether or not to take the branch.

Table 340. Effects of Command.b field

| Value | Effect |
|---|---|
| 0 | Never branch |
| 1 | Branch if the Branch Condition bit is set |
| 2 | Branch if the Branch Condition bit is cleared |
| 3 | Always branch |

**17-16, Word 3, (Command.w):** Upon completion of each command, channel operation may optionally be suspended. This action is controlled by the Command.w field in conjunction with an internal wait condition bit that is generated by the channel interface. The channel interface generates the wait condition based on the current values of the ChannelStatus.s7..s0 bits along with mask and data values found in the WaitSelect register of each channel. The suggested algorithm for generating the wait condition is given in Table 341 on page 185.

Table 341. Algorithm for Wait Condition

```
c = (ChannelStatus.s7..s0 &
WaitSelect.mask)
      == (WaitSelect.value &
WaitSelect.mask)
```

Table 340 on page 185 indicates how the wait condition bit is used in conjunction with the Command.w field to determine whether or not to suspend command execution. If a wait action is invoked, it occurs before the normal command completion sequence (branch determination, status update, interrupt generation, or next command fetch).

Table 342. Effects of Command.w field

| Value | Effect |
|---|---|
| 0 | Never wait |
| 1 | Branch if the Wait Condition bit is set |
| 2 | Branch if the Wait Condition bit is cleared |
| 3 | Always wait |

**Bit 15-0, Word 3 (Command.reqCount):** The Command.reqCount field is used to specify the number of bytes to be transferred by data transfer commands.

**Bit 31-0, Word 2 (Command.address):** The Command.address field is used to specify the primary 32-bit address used in data transfers. With INPUT and OUTPUT commands, it is always a system memory address and points to a data buffer. With LOAD_QUAD and STORE_QUAD commands, it is the address to or from which the immediate data is to be moved. In the latter case, it may point to system memory space, device memory space, or channel registers, as controlled by the Command.key field.

**Bit 31-0, Word 1 (Command.cmdDep):** The meaning and format of the 32-bit Command.cmdDep field is dependent on the Command.cmd values. With STORE_QUAD and LOAD_QUAD commands it is the data value to be written or read. With input and output commands it specifies a 32-bit conditional branch target. See the descriptions of individual commands in the next sections.

**Bit 31-16, Word 0 (Command.xferStatus):** Upon completion of a command, the current content of the channel's ChannelStatus register is written to the Command.xferStatus field. (The format of the ChannelStatus register is defined in Section 15.4.3, "ChannelStatus Register," on page 174.) When Command.xferStatus is written to memory, the bit corresponding to ChannelStatus.active is always set to 1. This serves as an indication that the corresponding command has been executed, and both the xferStatus and resCount fields have been updated. To make use of this feature, host software should clear the active bit in the Command.xferStatus field to 0 when the command descriptor is initialized.The ChannelControl.flush bit provides a mechanism for software to force buffered data to be written to memory and for status to be updated before the completion of the command. Partial status reports are characterized by the flush bit being set to 1 in the xferStatus field. Some devices may wish to provide intermediate status autonomously, identifying how many data bytes have been transferred, before completing the transfers specified by the command descriptor. This may be accomplished by providing the device with the ability to set the flush bit in the ChannelControl register. Intermediate status reports may be useful in cases such as terminal input, where the host needs to respond quickly to input data but would like to avoid interrupts on every byte. The intermediate status capability is optional. When it is implemented, the conditions under which partial status is returned are device-dependent; for example, status could be returned periodically, when signaled through device-dependent state bits, when a minimum number of data bytes are available, or in response to a combination of these conditions.

**Bit 15-0, Word 0 (Command.resCount):** Upon completion of a command, the 16-bit residual byte count value is written to the Command.resCount field. This value is normally 0, but may be more when the device prematurely terminates the data transfer.

**Note:** The Command.resCount and Command.xferStatus fields shall be updated in an indivisible operation.

## 15.7.2 INPUT and OUTPUT Commands

The INPUT_MORE, INPUT_LAST, OUTPUT_MORE, and OUTPUT_LAST commands transfer data between system memory and the device stream. The OUTPUT_MORE and OUTPUT_LAST commands transfer data from memory (which typically goes into the attached device). The INPUT_MORE and INPUT_LAST commands transfer data (which typically comes from the attached device) into memory. Data chaining is accomplished through the use of the MORE and LAST versions of these commands. The MORE commands indicate that the current buffer is not expected to complete a logical record (such as a network packet). The LAST commands indicate that the buffer is expected to complete a logical record. The reqCount field specifies the number of bytes to be transferred, the address field specifies a starting system memory address, and the branchAddress field specifies the address from which to fetch the next command if the branch test is successful. The Command.key field selects which of the device access ports is used. These fields are shown in Table 343 on page 187.

Table 343. Input and Output Command Formatr

| Word | Bit | Function |
|---|---|---|
| Word 3 (Little Endian Byte Ordering) | 31 -28 | cmd |
| | 27 | Reserved |
| | 26-24 | key |
| | 23-22 | Reserved |
| | 21-20 | i |
| | 19-18 | b |
| | 17-16 | w |
| | 15-0 | ReqCount |
| Word 2 | 31-0 | Address |
| Word 1 | 31-0 | Branch Address |
| Word 0 | 31-16 | XferStatus |
| | 15-0 | ResCount |

**Note:** An OUTPUT or INPUT command that performs a conventional transfer between system memory and an unaddressed device stream should specify a Command.key value of KEY_STREAM0. Additional stream identifiers allow access to other streams that may be provided by the device, such as control and status information. Command.key values of KEY_SYSTEM and KEY_DEVICE are used in conjunction with an (optional) Data2Ptr register to perform two-address transfers between system memory and the specified space.

## 15.7.3 STORE_QUAD Command

The STORE_QUAD command stores a 32-bit immediate value into memory. The 32-bit data32 field specifies the data value, and the address field (in combination with the key field) specifies the destination address, as shown in Table 344 on page 188.

Table 344. Store_Quad Command Format

| Word | Bit | Function |
|---|---|---|
| Word 3 (Little Endian Byte Ordering) | 31 -28 | cmd |
| | 27 | Reserved |
| | 26-24 | key |
| | 23-22 | Reserved |
| | 21-20 | i |
| | 19-18 | Reserved |
| | 17-16 | w |
| | 15-3 | Reserved |
| | 2-0 | ReqCount |
| Word 2 | 31-0 | Address |
| Word 1 | 31-0 | Data32 |
| Word 0 | 31-16 | XferStatus |
| | 15-0 | Reserved |

The field normally used for specifying whether a branch is to be taken (Command.b) is reserved in this command. Software shall write it with a 0 value. The operation of the channel is undefined when a nonzero value is written to this field.

The only valid values for the reqCount field are 1, 2 and 4. Only aligned transfers are supported. For example, if the count is 4, the low-order two bits of the address are assumed to be 0 and are ignored. Likewise, if the count is 2, the low-order bit of the address is assumed to be 0. When the count is less than 4 bytes, the least significant bytes of data32 are transferred.

STORE_QUAD maps illegal reqCount values into legal ones as shown in Table 345 on page 188. In Table 345 on page 188, *x* represents a bit that is ignored.

Table 345. STORE_QUAD reqCount Values

| reqCount | Effective value |
|---|---|
| 0b00*x* | 1 |
| 0b01*x* | 2 |
| 0b1*xx* | 4 |

During the execution of a STORE_QUAD command, the Command.key value specifies the destination address for the immediate data. Only key values of KEY_REGS, KEY_SYSTEM, and KEY_DEVICE are allowed. The operation of the channel is undefined when other key values are used.

## 15.7.4 LOAD_QUAD Command

The LOAD_QUAD command shall load a 32-bit immediate value from memory. The 32-bit data32 field is the destination and the address field (in combination with the key field) specifies the source address, as shown in Table 346 on page 189.

Table 346. Load_Quad Command Format

| Word | Bit | Function |
|---|---|---|
| Word 3 (Little Endian Byte Ordering) | 31 -28 | cmd |
| | 27 | Reserved |
| | 26-24 | key |
| | 23-22 | Reserved |
| | 21-20 | i |
| | 19-18 | Reserved |
| | 17-16 | w |
| | 15-3 | Reserved |
| | 2-0 | ReqCount |
| Word 2 | 31-0 | Address |
| Word 1 | 31-0 | Data32 |
| Word 0 | 31-16 | XferStatus |
| | 15-0 | Reserved |

The field normally used for specifying whether a branch is to be taken (Command.b) is reserved in this command. Software shall write it with a 0 value. The operation of the channel is undefined when a nonzero value is written to this field.

LOAD_QUAD handles the size and alignment of data transfers in the same way that the STORE_QUAD command does.

During the execution of a LOAD_QUAD command, the Command.key value specifies the source address for the immediate data. Only key values of KEY_REGS, KEY_SYSTEM, and KEY_DEVICE are allowed. The operation of the channel is undefined when other key values are used.

## 15.7.5 NOP Command

The NOP command performs no data transfers. However, it can use standard mechanisms to specify that interrupt, branch, or wait actions be performed. The NOP command is shown in Table 347 on page 189.

Table 347. NOP Command Format

| Word | Bit | Function |
|---|---|---|
| Word 3 (Little Endian Byte Ordering) | 31 -28 | cmd |
| | 27-22 | Reserved |
| | 21-20 | i |
| | 19-18 | b |
| | 17-16 | w |
| | 15-0 | Reserved |
| Word 2 | 31-0 | Address |
| Word 1 | 31-0 | Data32 |
| Word 0 | 31-16 | XferStatus |
| | 15-0 | Reserved |

### 15.7.6 STOP Command

The STOP command deactivates channel processing. It is placed at the end of a command list to indicate that there are currently no further commands to be processed. The only effect of the STOP command is that the channel goes idle and clears the ChannelStatus.active bit. The format of the STOP command is shown in Table 348 on page 190.

Table 348. Stop Command Format

| Word | Bit | Function |
|---|---|---|
| Word 3 (Little Endian Byte Ordering) | 31 -28 | cmd |
| | 27-0 | Reserved |
| Word 2 | 31-0 | Reserved |
| Word 1 | 31-0 | Reserved |
| Word 0 | 31-0 | Reserved |

To add additional commands to an active channel program, the following sequence should be performed:

1. Append the new commands to the existing channel program in memory.
2. Overwrite the STOP command with a NOP command (or the first new channel command).
3. Set the wake bit in the ChannelControl register to 1. This lets the channel process the new commands whether or not it had already completed the old commands and gone idle.

## 15.8 Asynchronous Event Packet Formats

The DBDMA architecture defines a special channel to allow interfaces to report low frequency asynchronous events. This shared resource alleviates the need for an additional channel to be allocated to each interface. The reporting of modem signal changes (such as DCD or CTS) is one example of a situation in which this facility might be used.

this feature is optional. In the presence of appropriate interrupt mechanisms and device status registers, it may not be needed.

The asynchronous event channel's command list is composed of a series of INPUT_LAST commands with a count of 4 bytes. When an asynchronous event occurs, the current INPUT_LAST command is executed by writing a 4-byte EventPacket to the address indicated by Command.address. This packet consists of a 16-bit eventSource field and a 16-bit eventType field, as shown in Table 349 on page 190.

Table 349. Asynchronous Event Packet Format

| Word | Bit | Function |
|---|---|---|
| Word 3 (Little Endian Byte Ordering) | 31 -0 | |
| Word 2 | 31-0 | |
| Word 1 | 31-0 | |
| Word 0 | 31-0 | |

The eventSource field identifies the source of the asynchronous event. It should contain a value that identifies the channel affiliated with the event. The eventType field identifies the nature of the event that was generated by the source. The assignment of values within this field is implementation-specific.

## 15.9 Hardwired Interrupts

**Requirements:**

**15–3.** The PCI devices described herein are allowed one interrupt each to the system. DBDMA interrupts must be shared with the other interrupt requirements of the device as described in the device chapters.

## 15.10 Examples

This section presents examples of implementations of the DBDMA architecture.

### 15.10.1 Command Queuing

This section describes how the DBDMA architecture might be used to support a circular queue structure for commands —in this example, for Ethernet packet transmission.

Table 350 illustrates a sample channel program. In this example, three commands are required for each packet: one to transmit the header, one to transmit the data, and a final one to receive the transmit status information. An interrupt is generated when the last command of the sequence finishes.

Table 350. Queued transmission commands

| No. | Command | key | i | b | w | reqCount | Address | cmdDep |
|---|---|---|---|---|---|---|---|---|
| 0 | OUTPUT_MORE | STREAM0 | NEVER | NEVER | NEVER | 100 | &Hdr1 | NA |
| 1 | OUTPUT_LAST | STREAM0 | NEVER | NEVER | NEVER | 1000 | &Buf1 | NA |
| 2 | INPUT_LAST | STREAM1 | ALWAYS | NEVER | NEVER | 4 | &Status1 | NA |
| 3 | OUTPUT_MORE | STREAM0 | NEVER | NEVER | NEVER | 100 | &Hdr2 | NA |
| 4 | OUTPUT_LAST | STREAM0 | NEVER | NEVER | NEVER | 1000 | &Buf2 | NA |
| 5 | INPUT_LAST | STREAM1 | ALWAYS | NEVER | NEVER | 4 | &Status2 | NA |
| 6 | STOP->NOP | NA | NA | NA | NA | NA | NA | NA |
| 7 | OUTPUT_MORE | STREAM0 | NEVER | NEVER | NEVER | 100 | &Hdr2 | NA |
| 8 | OUTPUT_LAST | STREAM0 | NEVER | NEVER | NEVER | 1000 | &Buf2 | NA |
| 9 | INPUT_LAST | STREAM1 | ALWAYS | NEVER | NEVER | 4 | &Status2 | NA |
| 10 | STOP | NA | NA | NA | NA | NA | NA | NA |
| 11 | - | - | - | - | - | - | - | - |
| 12 | - | - | - | - | - | - | - | - |
| 13 | NOP (Branch) | NA | NEVER | ALWAYS | NEVER | NA | NA | &Cmd0 |

In the example, commands 0 through 5 represent a request to transmit two packets. Initially these commands are followed by a STOP command because there is no further work to be done. When a new request for a packet transmission is received from a client, it can be added to the queue without waiting for completion of the first two packets. This is accomplished by appending the new commands to the list, changing the STOP to a NOP, and setting the wake bit in the ChannelControl register to 1.

Adding commands to the queue is illustrated by commands 6 through 10 in the example. Note that there are several alternatives to the insertion of the NOP command. For example, the STOP command could be overwritten with the new OUTPUT_MORE command after the remaining commands and the new STOP command are written. Software could also replace the stop with a NOP that branches to a new command group, terminated with a STOP command.

Command groups may continue to be added in this manner until the allocated descriptor blocks are exhausted. At that point, a branch back to the first descriptor can be executed, and new command groups can overwrite those which have been executed and released by hardware. (Hardware releases descriptors by writing to the Command.xferStatus field.) The branch can either take the form of a static, standalone command, as illustrated here, or can be embedded in the last data transfer command.

## 15.10.2 Ethernet Reception

**Note:** An Ethernet implementation using DBDMA is not documented herein. This example is given to demonstrate DBDMA operation.

This section illustrates one possible way in which the DBDMA architecture can be used to support Ethernet packet reception and mini-buffer allocation. In this example, memory utilization is optimized by allocating buffers that are substantially smaller than the maximum packet size. Thus, in the typical situation where the majority of packets received are quite small, less physical memory must be allocated in order to buffer a given number of packets.

As shown in Table 351, the command list is structured as a number of channel command pairs. After each input buffer has been filled, the channel checks for packet completion. If the packet is complete, a command is executed that fetches the receive status from the device's status stream and stores it in system memory. An interrupt is generated when this command finishes. Note that the first command in the command list is a STORE_QUAD. This initializes the BranchSelect register to the value required for the following conditional branches by specifying that the end of frame bit be tested.

Table 351. Ethernet receive channel commands

| No. | Command | key | i | b | w | reqCount | Address | cmdDep |
|---|---|---|---|---|---|---|---|---|
| 0 | STORE_QUAD | SYSTEM | NEVER | NEVER | NEVER | 4 | &BrSelReg | 00400040 <EOF=1> |
| 1 | INPUT_MORE | STREAM0 | NEVER | !Cond | NEVER | 256 | &Buffer0 | &Cmd3 |
| 2 | INPUT_LAST | STREAM1 | ALWAYS | NEVER | NEVER | 4 | &Status0 | NA |
| 3 | INPUT_MORE | STREAM0 | NEVER | !Cond | NEVER | 256 | &Buffer1 | &Cmd5 |
| 4 | INPUT_LAST | STREAM1 | ALWAYS | NEVER | NEVER | 4 | &Status1 | NA |
| 5 | INPUT_MORE | STREAM0 | NEVER | !Cond | NEVER | 256 | &Buffer2 | &Cmd7 |
| 6 | INPUT_LAST | STREAM1 | ALWAYS | NEVER | NEVER | 4 | &Status2 | NA |
| 7 | INPUT_MORE | STREAM0 | NEVER | !Cond | NEVER | 256 | &Buffer3 | &Cmd2 |
| 8 | INPUT_LAST | STREAM1 | ALWAYS | NEVER | NEVER | 4 | &Status3 | NA |
| 9 | STOP | NA | NA | NA | NA | NA | NA | NA |
| 10 | NOP (BRANCH) | NA | NEVER | ALWAYS | NEVER | NA | NA | &Cmd1 |

Only four input buffers are used in this example; in practice many more receive buffers would typically be allocated. The command list is structured as a circular queue of buffers. A STOP command must always be inserted after the last buffer/status command pair. This delineates the buffers that are owned by the controller from the returned buffers that are owned by the host, and prevents the controller from overwriting any filled buffers that have not been released by the host.

As buffers are filled and returned by the controller, new receive buffers are added to the queue in their place, and the position of the STOP command is advanced. This progression of the command list structure is shown in Table 352. Note that this example assumes that the first two received packets each fit within a single mini-buffer and that the third packet requires two mini-buffers.

Table 352. Ethernet receive command sequence

| No. | Initial command list | After Buffer 0 is filled | After Buffer 1 is filled | After Buffers 2 and 3 are filled |
|---|---|---|---|---|
| 0 | STORE_QUAD | STORE_QUAD | STORE_QUAD | STORE_QUAD |
| 1<br>2 | Buffer0<br>Status0 | STOP<br>NOP | Buffer5<br>Status5 | Buffer5<br>Status5 |
| 3<br>4 | Buffer 1<br>Status 1 | Buffer 1<br>Status 1 | STOP<br>NOP | Buffer 6<br>Status 6 |
| 5<br>6 | Buffer 2<br>Status 2 | Buffer 2<br>Status 2 | Buffer 2<br>Status 2 | Buffer 7<br>Status 7 |
| 7<br>8 | Buffer 3<br>Status 3 | Buffer 3<br>Status 3 | Buffer 3<br>Status 3 | STOP<br>NOP |
| 9<br>10 | STOP<br>NOP (w. Branch) | Buffer 4<br>Status 4 (w. Branch) | Buffer 4<br>Status 4 (w. Branch) | Buffer 4<br>Status 4 (w. Branch) |

## 15.10.3 Full Handshake Flow Control

The DBDMA architecture also allows large device-to-device transfers to be performed autonomously, using small memory-resident buffers for intermediate data storage.

As an example, consider the live capture of the most-recent video images on a disk device, as illustrated in Figure 10. In the most general case, the transient data transfer rates of the two devices may be arbitrarily different—the disk would have a higher transient data transfer rate, but the data transfer rate would drop to 0 during seek operations. For illustration purposes, assume that a small video sample is being saved in a circular buffer on disk.

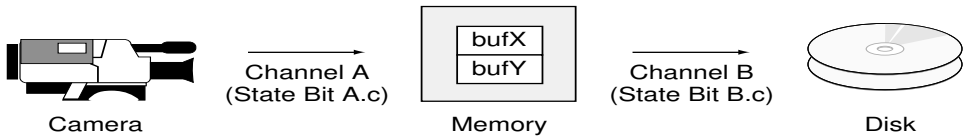


Figure 10. Flow-controlled device model

The DBDMA channels contain state bits that can influence the conditional execution of WAIT actions. Thus, channel programs can use this capability to spin-wait on externally generated events. For the data camera, flow control involves waiting for buffer-use bits to be set by the disk when a data buffer has been read from memory. For the disk, flow control involves waiting for buffer-use bits to be set by the data camera when a new data buffer has been written into memory.

The channelA program uses a STORE_QUAD command to set a channelB state bit (B.s0) after each data buffer has been written; channelA waits for its own state bit (A.s0) to be set by channelB after each data buffer has been consumed. The structure of these two channel programs is illustrated in Figure 11. In the diagram, dotted lines are used to illustrate how the STORE_QUAD command entries in one program are coupled to the WAIT actions in the other.

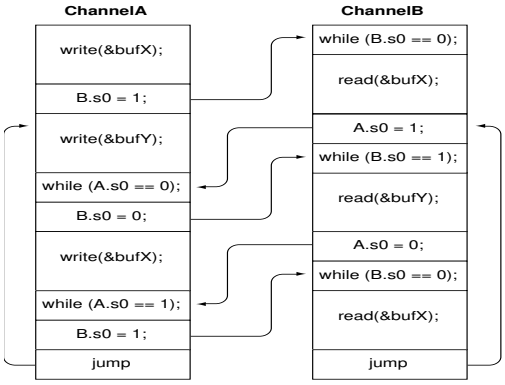


Figure 11. Synchronizing flow-controlled channel programs

# VGA Programming Model

## A.1 Introduction

IBM introduced the Video Graphics Array (VGA) display standard in 1987 to provide a higher resolution color graphics capability to its PS/2 line of personal computers. VGA BIOS was provided to control the VGA hardware, but programming directly to the hardware has become the more common approach for performance reasons. Since its introduction, VGA has been "cloned" by many manufacturers, claiming register-level compatibility. If a software developer is programming the hardware directly, 100 percent compatibility is imperative; unfortunately, this is not always the case.

The VGA function includes a video buffer, a video digital-to-analog converter (DAC), a CRT controller, a sequencer unit, a graphics controller, and an attribute controller.

*Video memory* consists of at least 256KB; its use and mapping depend on the mode selected. It is configured as four 64KB memory maps:

The *video DAC* contains the color palette that is used to convert the video data into the video signal that is sent to the display. Three analog signals (red, green, and blue) are output from the DAC.

The *CRT controller* generates horizontal and vertical synchronization signal timings, addressing for the regenerative buffer, cursor and underline timings, and refresh addressing for the video memory.

The *sequencer* generates basic memory timings for the video memory and the character clock for controlling regenerative buffer fetches. It allows the system to access memory during active display intervals by periodically inserting dedicated system microprocessor memory cycles between the display memory cycles. Map mask registers in the sequencer are available to protect entire memory maps from being changed.

The *graphics controller* is the interface between the video memory and the attribute controller during active display times, and between video memory and the system microprocessor during memory accesses.

During active display times, memory data is latched and sent to the attribute controller. In graphics modes, the memory data is converted from parallel to serial bit-plane data before being sent; in alphanumeric modes, the parallel attribute data is sent.

During system accesses of video memory, the graphics controller can perform logical operations on the memory data before it reaches video memory or the system data bus. These logical operations are composed of four logical write modes and two logical read modes. The logical operators allow enhanced operations, such as a color compare in the read mode, individual bit masking during write modes, internal 32-bit writes in a single memory cycle, and writing to the display buffer on non-byte boundaries.

The *attribute controller* takes in data from video memory through the graphics controller and formats it for display. Attribute data in alphanumeric mode and serialized bit-plane data in graphics mode are converted to an 8-bit color value. Each color value is selected from an internal color palette of 64 possible colors (except in 256-color mode). The color value is used as a pointer into the video DAC where it is converted to the analog signals that drive the display.

## A.2 VGA Modes

VGA provides alphanumeric modes for text processing and graphics modes for graphics processing. The following table describes the alphanumeric (A/N) and all points addressable (APA) graphics modes available in standard VGA monitors. Each color is selected from 256K possibilities, and gray shades from 64 possibilities. The variations within these modes are selected by setting the number of scan lines. The scan line count is set before the mode set is executed. In the 200-scan-line modes, the data for each scan line is scanned twice. This double scanning allows the 200-scan-line image to be displayed in 400 scan lines.

Table 353. VGA Video Modes

| Mode (Hex) | Type | Colors | Alpha Format | Buffer Start | Box Size | Max. Pages | Freq | Resolution (hxv Pixels) | Double Scan | Border Support |
|---|---|---|---|---|---|---|---|---|---|---|
| 0,1 | A/N | 16 | 40x25 | B8000 | 8x8 | 8 | 70Hz | 320x200 | Yes | No |
| 0*,1* | A/N | 16 | 40x25 | B8000 | 8x14 | 8 | 70Hz | 320x350 | No | No |
| 0+,1+ | A/N | 16 | 40x25 | B8000 | 9x16 | 8 | 70Hz | 360x400 | No | No |
| 2,3 | A/N | 16 | 80x25 | B8000 | 8x8 | 8 | 70Hz | 640x200 | Yes | Yes |
| 2-,3- | A/N | 16 | 80x25 | B8000 | 8x14 | 8 | 70Hz | 640x350 | No | Yes |
| 2+,3+ | A/N | 16 | 80x25 | B8000 | 9x16 | 8 | 70Hz | 720x400 | No | Yes |
| 4,5 | APA | 4 | 40x25 | B8000 | 8x8 | 1 | 70Hz | 320x200 | Yes | No |
| 6 | APA | 2 | 80x25 | B8000 | 8x8 | 1 | 70Hz | 640x200 | Yes | Yes |
| 7 | A/N | - | 80x25 | B0000 | 9x14 | 8 | 70Hz | 720x350 | No | Yes |
| 7+ | A/N | - | 80x25 | B0000 | 9x16 | 8 | 70Hz | 720x400 | No | Yes |
| D | APA | 16 | 40x25 | A0000 | 8x8 | 8 | 70Hz | 320x200 | Yes | No |
| E | APA | 16 | 80x25 | A0000 | 8x8 | 4 | 70Hz | 640x200 | Yes | Yes |
| F | APA | - | 80x25 | A0000 | 8x14 | 2 | 70Hz | 640x350 | No | Yes |
| 10 | APA | 16 | 80x25 | A0000 | 8x14 | 2 | 70Hz | 640x350 | No | Yes |
| 11 | APA | 2 | 80x30 | A0000 | 8x16 | 1 | 60Hz | 640x480 | No | Yes |
| 12 | APA | 16 | 80x30 | A0000 | 8x16 | 1 | 60Hz | 640x480 | No | Yes |
| 13 | APA | 256 | 40x25 | A0000 | 8x8 | 1 | 70Hz | 320x200 | Yes | Yes |
| **Note: * or + indicate enhanced modes** | | | | | | | | | | |

Mode 3+ is the default mode with a color display attached and mode 7 is the default mode with a monochrome display attached. The graphics controller senses whether the display is monochrome or color and initializes itself to the appropriate mode.

### A.2.1 Alphanumeric Modes

The alphanumeric modes are modes 0 through 3 and 7. The mode chart lists the variations of these modes. The data format for alphanumeric modes is the same as the data format on the IBM Color/Graphics Monitor Adapter, the IBM Monochrome Display Adapter. and the IBM Enhanced Graphics Adapter.

The video subsystem is initialized according to the selected mode and the color values are loaded into the video DAC. These color values can be changed to give a different color set to select from. Bit 3 of the attribute byte may be redefined by the Character Map Select register to act as a switch between character sets, giving the Programmer access to 512 characters at one time.

When an alphanumeric mode is selected, the character font patterns are transferred from the ROM to map 2. The system stores the character data in map 0, and the attribute data in map 1. In the alphanumeric modes, the programmer views maps 0 and 1 as a single buffer. The CRT controller generates sequential addresses, and fetches one character code byte and one attribute byte at a time. The character code and row scan count are combined to make up the address into map 2, which contains the character font. The appropriate dot patterns are then sent to the attribute controller, where color is assigned according to the attribute data.

Every display character position in the alphanumeric mode is defined by two bytes in the display buffer. Both the color/graphics and the monochrome emulation modes use the following 2-byte character/attribute format.

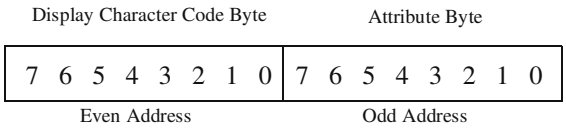


Figure 12. Character/Attribute Format

The functions of the attribute byte are defined in the following table. Bit 7 can be redefined in the Attribute Mode Control register to give 16 possible background colors; its default is to control character blinking. Bit 3 can be redefined in the Character Map Select register to select between two character fonts; its default is to control foreground color selection

.

Table 354. Attribute Byte Definitions

| Bit | Color | Function |
|---|---|---|
| 7 | B/I | Blinking or Background Intensity |
| 6 | R | Background Color |
| 5 | G | Background Color |
| 4 | B | Background Color |
| 3 | I/CS | Foreground Intensity or Character Font Select |
| 2 | R | Foreground Color |
| 1 | G | Foreground Color |
| 0 | B | Foreground Color |

The following are the color values loaded for the 16-color modes.

Table 355. Firmware Color Initialization

| IRGB | Color |
|---|---|
| 0000 | Black |
| 0001 | Blue |
| 0010 | Green |
| 0011 | Cyan |
| 0100 | Red |
| 0101 | Magenta |
| 0110 | Brown |
| 0111 | White |
| 1000 | Gray |
| 1001 | Light Blue |
| 1010 | Light Green |
| 1011 | Light Cyan |
| 1100 | Light Red |
| 1101 | Light Magenta |
| 1110 | Yellow |
| 1111 | White (High Intensity) |

Both 40-column and 80-column alphanumeric modes are supported. The features of the 40-column alphanumeric modes (all variations of modes 0 and 1) are:

- 25 rows of 40 characters
- 2,000 bytes of video memory per page
- One character byte and one attribute byte per character.

The features of the 80-column alphanumeric modes (all variations of modes 2, 3, and 7) are:

- 25 rows of 80 characters
- 4,000 bytes of video memory per page
- One character byte and one attribute byte per character.

## A.2.2 Graphics Modes

This section describes the graphics modes supported. The colors in this section are generated when the BIOS is used to set the mode. BIOS initializes the video subsystem and the DAC palette to generate these colors. If the DAC palette is changed, different colors are generated.

### A.2.2.1 320 x 200 Four Color Graphics (Modes 4 and 5)

Addressing, mapping, and data format are the same as the 320 x 200 PEL mode of the IBM Color/Graphics Monitor Adapter. The display buffer is configured at 0xB8000. Bit image data is stored in memory maps 0 and 1. The two bit planes (C0 and C1) are each formed from bits from both memory maps.

Features of this mode are:

- A maximum of 200 rows of 320 pels
- Double-scanned to display as 400 rows
- Memory-mapped graphics
- Four colors for each PEL
- Four pels per byte
- 16,000 bytes of read/write memory.

The video memory is organized into two banks of 8,000 bytes each using the following format. Address 0xB8000 contains the PEL information for the upper left corner of the display area.

The following figure shows the format for each byte.

Table 356. PEL Format, Modes 4 and 5

| Bit | Function |
|---|---|
| 7 | C1- First Display PEL |
| 6 | C0 - First Display PEL |
| 5 | C1- Second Display PEL |
| 4 | C0 - Second Display PEL |
| 3 | C1 - Third Display PEL |
| 2 | C0 - Third Display PEL |
| 1 | C1- Fourth Display PEL |
| 0 | C0 - Fourth Display PEL |

### A.2.2.2 640 x 200 Two Color Graphics (Mode 6)

Addressing, scan-line mapping, and data format are the same as the 640 x 200 PEL black and white mode of the IBM Color/Graphics Monitor Adapter. The display buffer is configured at 0xB8000. Bit image data is stored in memory map 0 and comprises a single bit plane (C0). Features of this mode are:

- A maximum of 200 rows of 640 pels
- Double scanned to display as 400 rows
- Same addressing and scan-line mapping as 320 x 200 graphics
- Two colors for each PEL
- Eight pels per byte
- 16,000 bytes of read/write memory.

The following shows the format for each byte. The bit definition for each PEL is 0 equals black and 1 equals intensified white.

Table 357. PEL Format, mode 6

| Bit | Function |
|---|---|
| 7 | First Display PEL |
| 6 | Second Display PEL |
| 5 | Third Display PEL |
| 4 | Forth Display PEL |
| 3 | Fifth Display PEL |
| 2 | Sixth Display PEL |
| 1 | Seventh Display PEL |
| 0 | Eighth Display PEL |

### A.2.2.3 640 x 350 Graphics (mode F)

This mode emulates the EGA graphics with the monochrome display and the following attributes: black, video, blinking video, and intensified video. A resolution of 640 x 350 uses 56,000 bytes of video memory to support the four attributes. This mode uses maps 0 and 2; map 0 is the video bit plane (C0), and map 2 is the intensity bit plane (C2). Both planes reside at address 0xA0000.

The two bits, one from each bit plane, define one PEL. The bit definitions are given in the following table.

Table 358. Bit Definitions C2,C0

| C2 C0 | PEL Color |
|---|---|
| 0 0 | Black |
| 01 | White |
| 1 0 | Blinking White |
| 1 1 | Intensified White |

Memory is organized with successive bytes defining successive pels. The first eight pels displayed are defined by the byte at 0xA0000, the second eight pels at 0xA0001, and so on. The most significant bit in each byte defines the first PEL for that byte.

Since both bit planes reside at address 0xA0000, the user must select the plane to update through the Map Mask register of the sequencer.

### A.2.2.4 640 x 480 Two Color Graphics (mode 11 hex)

This mode provides two color graphics with the same data format as mode 6.

The bit image data is stored in map 0 and comprises a single bit plane (C0). The video buffer starts at 0xA0000. The first byte contains the first eight pels; the second byte at 0xA0001 contains the second eight pels, and so on. The bit definition for each PEL is 0 equals black and 1 equals intensified white.