

Chapter 20: Card Services

Note that the AcknowledgeInterrupt function is called by the status change interrupt service routine. Interrupts must not be re-enabled while processing a status change interrupt. This could cause nesting of status change interrupts while processing the socket service's AcknowledgeInterrupt, a situation that socket services is unprepared to manage (because the routine is non-reentrant).

The Client Call-Back

Numerous events can occur that require a call-back to client drivers. These events are listed in table 20-11. The events in the shaded boxes were added with the PC Card 95 release.

Table 20-11. Call-Back Events Defined by Card Services

Event	Code	Source	Client(s)	Registered By
BATTERY_DEAD	01h	Hardware	Socket	RequestSocketMask
BATTERY_LOW	02h	Hardware	Socket	RequestSocketMask
CARD_INSERTION	40h	Hardware	All	RegisterClient
CARD_INSERTION [A]	40h	DeregisterMTD	MTDs	RegisterClient
CARD_INSERTION [A]	40h	RegisterClient	Requester	RegisterClient
CARD_INSERTION [A]	40h	ReleaseExclusive	All	RegisterClient
CARD_INSERTION [A]	40h	RequestExclusive	Requester	RequestExclusive
CARD_INSERTION [A]	40h	RequestExclusive	All	RegisterClient
CARD_LOCK	03h	Hardware	Socket	RequestSocketMask
CARD_READY	04h	Hardware	Socket	RequestSocketMask
CARD_REMOVAL	05h	Hardware	Socket	RequestSocketMask
CARD_REMOVAL [A]	05h	ReleaseExclusive	Socket	RequestSocketMask
CARD_REMOVAL [A]	05h	RequestExclusive	All	RegisterClient
CARD_RESET	11h	ResetFunction	Socket	RequestSocketMask
CARD_UNLOCK	06h	Hardware	Socket	RequestSocketMask
CLIENT_INFO	14h	GetClientInfo	Provider	RegisterClient
EJECTION_COMPLETE	07h	Hardware	Socket	RequestSocketMask
EJECTION_REQUEST	08h	Hardware	Socket	RequestSocketMask
ERASE_COMPLETE	81h	Queued Erase	Requester	RequestEraseQueue
EXCLUSIVE_COMPLETE	0Dh	RequestExclusive	Requester	RequestExclusive
EXCLUSIVE_REQUEST	0Eh	RequestExclusive	Socket	RequestSocketMask
INSERTION_COMPLETE	09h	Hardware	Socket	RequestSocketMask
INSERTION_REQUEST	0Ah	Hardware	Socket	RequestSocketMask
MTD_REQUEST	12h	Card Services	MTD	RegisterClient
PM_RESUME	0Bh	Card Services	Socket	RequestSocketMask

PCMCIA System Architecture

Table 20-11. Call-Back Events Defined by Card Services

Event	Code	Source	Client(s)	Registered By
PM_SUSPEND	0Ch	Card Services	Socket	RequestSocketMask
REGISTRATION_COMPLETE	82h	RegisterClient	Requester	RegisterClient
REQUEST_ATTENTION	18h	Hardware	All	RegisterClient
RESET_COMPLETE	80h	ResetFunction	Requester	ResetFunction
RESET_PHYSICAL	0Fh	ResetFunction	Socket	RegisterClient
RESET_REQUEST	10h	ResetFunction	Socket	RegisterClient
SS_UPDATED	16h	Card Services	All	RegisterClient
TIMER_EXPIRED	15h	Hardware	Requester	RegisterTimer
WRITE_PROTECT	17h	Hardware	All	RegisterClient

Configuring PC Cards During POST

The previous discussions of PC Card configuration have presumed that the cards will be installed when the operating system loads or when the PC Card is inserted sometime after the operating system has loaded and the system is running. If however, the need to load the operating system from the PC Card exists, the previously discussed approaches for configuring the cards don't work.

To perform initial program load (IPL) from a PC Card, ROM-based PCMCIA initialization code must be included with the system. This code must be able to program the HBA and parse the CIS to determine if a given card should be configured during POST (Power-On Self Test). Once the HBA has been programmed, memory cards containing a boot sector can be recognized as bootable since they will contain a BIOS Parameter Block (BPB) that permits the booting from the PC Card in the same fashion as a floppy drive.

Similarly, ATA drives can be recognized by ROM code by reading the initialization byte within the Function Identification tuple. The initialization byte specifies that the device should be configured during POST. Once the ATA drive is configured, IPL can occur from the PCMCIA ATA drive like any other ATA drive.

Note that this initialization process occurs prior to card services being installed. As a result, the a client driver will not have registered to receive status change events from the PC Card. When the operating system boots, a driver for the PC Card that is performing IPL can register with card services.

Chapter 21

The Previous Chapter

The previous chapter focused on the role of card services in the PCMCIA environment. It also reviewed each of the functions defined by the PC Card specification that apply to 16-bit PC Cards, and defined the related return codes. The call back mechanism was also described and the event and call back codes were defined.

This Chapter

This chapter discusses the three basic types of enablers: point enablers, device-specific enablers, and super enablers. The chapter also discusses the jobs performed by generic memory enablers (and MTDs) and I/O device enablers.

The Next Chapter

The next chapter discusses the problems associated with loading the operating system from a PC Card. It also defines mechanisms used to determine whether a given PC Card is a bootable device, and the firmware support required to support PC Card booting.

Overview

This chapter discusses PC Card enablers. The chapter focuses primarily on client driver enablers, but also includes a brief discussion of point enablers at the end of the chapter. Note that the terms PC Card enabler, client, client driver, and device driver are all used to describe the software that is responsible for configuring a PC Card. This chapter uses the terms enabler and client driver.

PCMCIA System Architecture

Specific types of client drivers (enablers) discussed in this chapter include:

- SRAM client drivers
- Flash client drivers and Memory Technology Drivers (MTDs)
- Generic I/O client drivers

In order to configure a PC Card, enablers must first register with card services. The primary function of an enabler is to detect and configure PC Cards that it supports. As such, the enabler must be prepared to configure its card no matter when it is installed. In order to configure cards installed after power up, the enabler registers with card services to receive a call back (i.e., a card insertion call-back) each time card services detects that a PC Cards has been installed. During registration, the enabler can also request that card services generate a call-back for each PC Card already installed, thereby calling the enabler's configuration routine.

The Card Insertion Call-Back

When card services makes a card insertion call-back it specifies the type of call-back initiated, along with the logical socket that the card was inserted into. The call-back routine then attempts to configure the card. Figure 21-1 illustrates the typical process used by an enabler to configure a card. The CIS may have to be accessed several times to obtain a combination of card-required resources that can be successfully allocated to the card (i.e., resources that are not already assigned to other devices).

The configuration process begins when card services makes the card insertion call-back to the enabler. The enabler detects which event caused the call-back and obtains information supplied by card services (e.g., which logical socket the card was inserted into).

The method of configuring a card varies depending of the type of card to be configured. The next section discusses generic memory enablers, and the following section describes the operation of a generic I/O enabler.

As discussed in the previous chapter, a variety of services are available for the PC Card enablers (i.e., card services client drivers) to configure a PC Card.

Chapter 21: Client Drivers

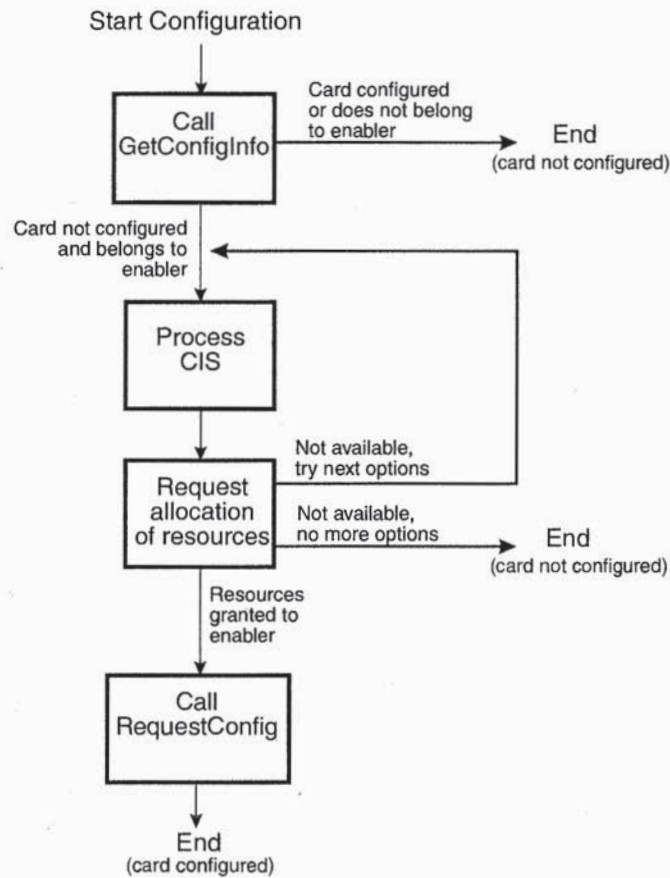


Figure 21-1. A Sample Configuration Process Used By a Card Services Client

Memory Drivers and Memory Technology Drivers

Memory client drivers provide virtual disk drive support. In short, these drivers are responsible for storing and retrieving files within the memory card. The method required to access the memory card varies depending on the type of memory devices (i.e., memory technology) implemented in the card. Since memory devices (such as flash) require various programming algorithms, each memory type must have an associated memory technology driver (MTD).

PCMCIA System Architecture

Figure 21-2 illustrates the overall software architecture specified for accessing memory cards as virtual disk drives. Notice that memory client drivers receive file access requests from the file system and must access the memory card to fulfill each request. The file system might be the standard file system used by the operating system (e.g., the DOS FAT system) or an installable system required when accessing flash memory. Card services provides bulk memory services that simplifies the memory client driver's job of accessing a specific block of memory within the memory card.

SRAM memory client drivers typically interface directly to the operating system's file manager, since there are no restrictions related to writing and reading data to or from SRAM. These client drivers are designed to access memory via the bulk memory services provided by card services. Since accessing SRAM is uniform and quite simple (byte read/write capability), the memory technology driver is incorporated into card services.

Flash memory client drivers interface directly to a flash file system. A special file system is required for flash devices due to the special requirements associated with writing to flash memory. Two major factors are:

- Write operations require first erasing a specified block of memory followed by the block write, and may take several seconds to complete.
- Flash memory also has a limited write-cycle life. That is, repetitive erasures and writes to the same memory block destroys the chips ability to retain data within that block. The maximum number of erasures and writes are specified by the manufacturer (e.g., a flash device may specify as life of as few as 10,000 writes).

Knowing the restrictions associated with accessing flash memory, the flash file system is designed specifically to provide compatible access to flash memory. For example, the flash file system distributes writes to flash memory to minimize the effects of repetitive write wear and accommodates the slow erase time.

Memory enablers (client drivers) have a formidable task to perform since a wide variety of memory card implementations exist. The enabler must also acquire a drive letter from the operating system to allocate to each card slots that a memory card might be inserted into. The following sections describe the jobs performed by SRAM and flash client drivers.

Chapter 21: Client Drivers

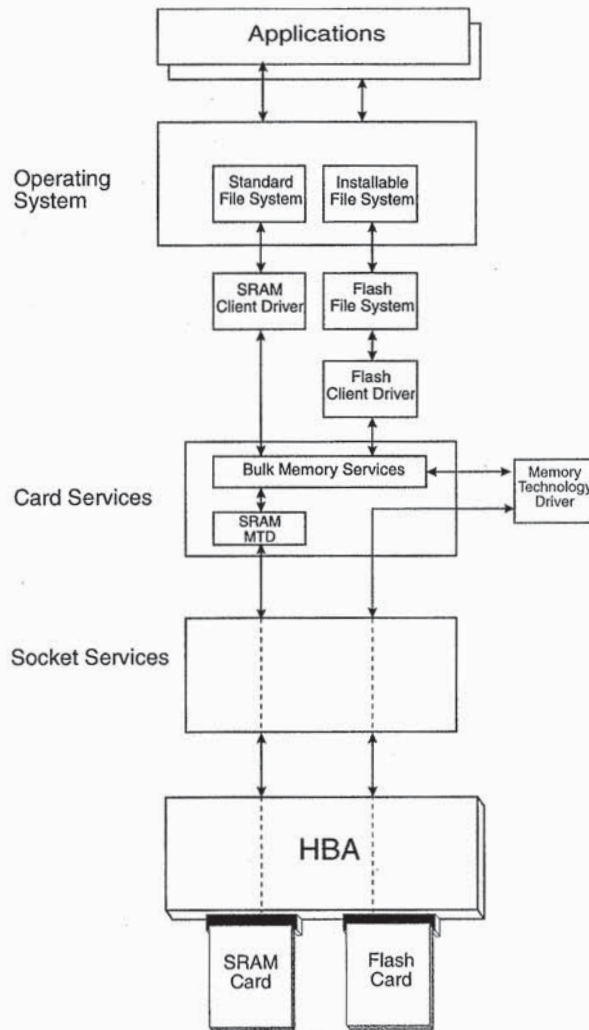


Figure 21-2. Memory Client Driver Software Environment

SRAM Client Drivers

SRAM client drivers typically load as installable device drivers via the `config.sys` file or equivalent mechanism. One of the tasks performed by the device driver is to detect the presence of card services by calling the `GetCSInfo` service. If the call returns the ASCII string "CS," then SRAM client driver recognizes that card services are installed. Note that if card services is not installed, the SRAM driver typically reports the error condition and terminates

PCMCIA System Architecture

without remaining resident in memory. When card services is detected, the driver then registers with card services.

The driver must also obtain logical drive letters needed to perform the disk emulation. Note that the drive letter is acquired when the device driver installs even though a memory card may not be installed in the system. In this case, an attempt to access files associated with the drive letter assigned to the socket will result in a drive not ready error.

SRAM Client Driver Registers with Card Services

The client driver performs the registration process by calling the RegisterClient service. The SRAM client identifies itself as a memory client, registers to receive relevant call-back events, and passes a pointer to its call-back routine. The client may also request that card services generate a card insertion call-back for each PC Card already installed in the system. The memory client receives a handle value from card services when it returns from the call. Once registered, the memory client awaits call-backs from card services, notifying it when a PC card is inserted or removed.

The SRAM Client Driver Call-Back

When card services generates the card insertion call-back, it also passes the logical socket number that the PC Card was inserted into. The memory client then attempts to configure the PC Card.

The memory client must first determine if it should attempt to configure the PC Card by determining the card type. Memory clients can use the bulk memory services to access a specific region within the PC card. To access memory, the client first calls the OpenMemory service by specifying an offset within the card's attribute or common memory address space. Card services then returns a memory handle to the client that it can use when accessing memory relative to the offset specified in the OpenMemory service. Note that if card services does not support bulk memory services, the memory client must use the RequestWindow service to specify the host system address space that it wishes to use to access PC Card memory.

Reading from or writing to PC Card memory is accomplished by calling the ReadMemory or WriteMemory services. The memory client passes the memory handle it received from the OpenMemory service and specifies the memory offset and range of addresses it wishes to access. The call will likely

Chapter 21: Client Drivers

specify location zero within the attribute memory address space. When the data is returned to the memory client it evaluates the DEVICE tuple to determine if the card contains SRAM.

Note that determining the card type can be a complicated process for memory clients. Some memory cards implement the CIS in attribute memory (required by the PC Card standard), some implement the CIS in common memory, while others do not implement a CIS at all. To complicate matters, some CIS implementations are invalid, requiring the enabler to attempt interpretation of the faulty CIS. If the card does not contain a CIS, the enabler attempts to detect the presence of the BPB (BIOS parameter block), which contains information that specifies the logical size of the disk. The BPB if present should reside at either location 0 or 512 in common memory.

If an SRAM card is detected, the call-back routine return to card services, indicating that the card was successfully configured. If the PC Card was not an SRAM card, the client returns to card services, indicating the card was not configured by the SRAM enabler.

Flash Client Drivers

Figure 21-3 illustrates the flash client driver software environment. Three types of flash client drivers are illustrated in figure 21-3. Two of the client drivers are shown interfacing to a flash file system and the other via a file translation layer. (Each file system is discussed later in this chapter.)

The flash client drivers typically load as installable device drivers via the config.sys file or equivalent mechanism. The first task performed by the device driver is to detect the presence of card services. If card services are not installed, the flash driver typically reports the error condition and terminates without remaining resident in memory. When card services is detected, the driver then registers with card services.

Flash client drivers differ from SRAM drivers in two important ways:

- Flash client drivers interface to the flash file system
- MTD client drivers must be installed to handle calls made to bulk memory services

The MTD must register prior to the flash client driver. This is necessary because the flash client driver uses the MTD to access the flash card.

PCMCIA System Architecture

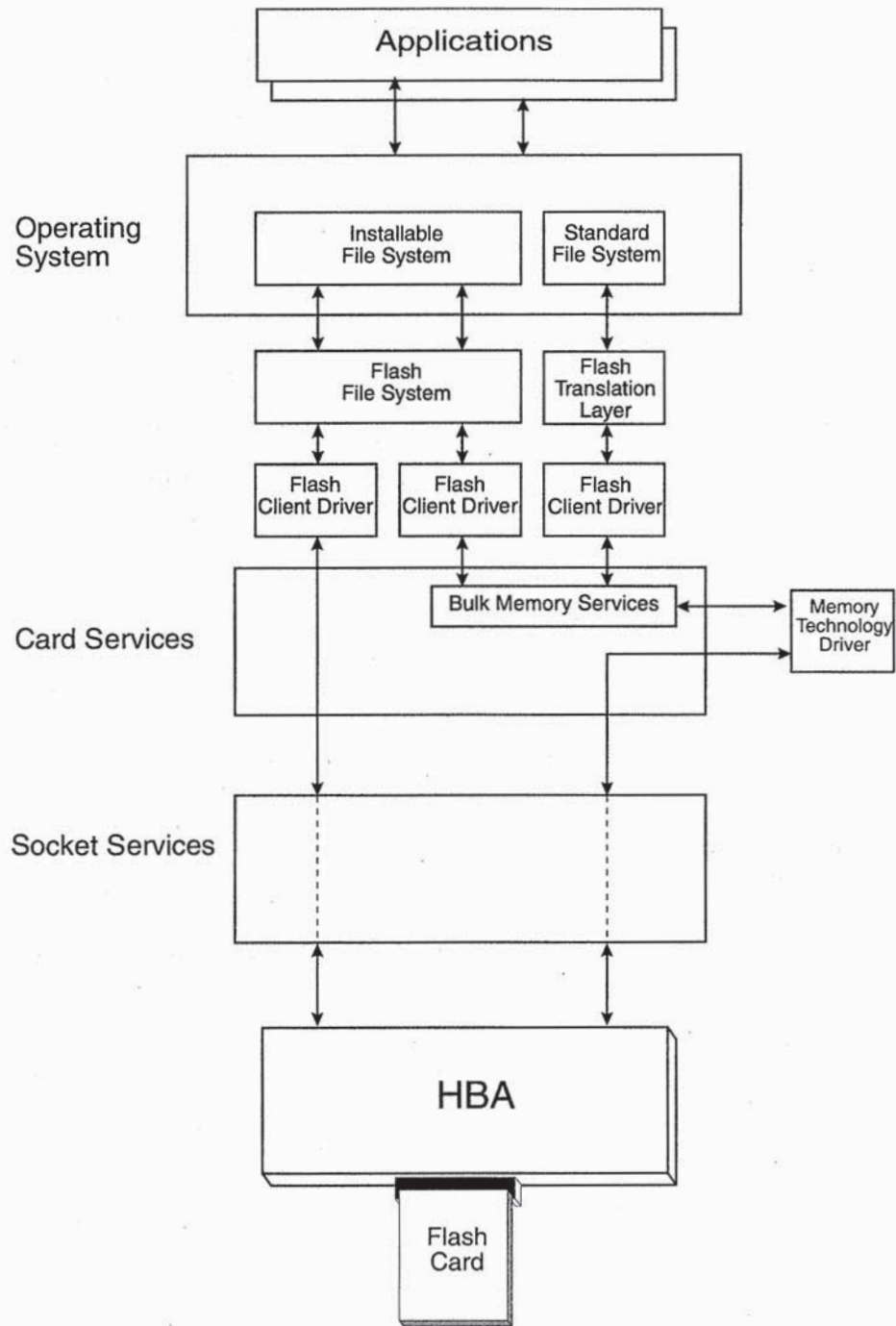


Figure 21-3. Software Environment Required for Flash Card Support

Chapter 21: Client Drivers

Like SRAM client drivers, flash client drivers must also obtain logical drive letters needed to perform the disk emulation. The drive letter is acquired when the device driver installs even though a memory card may not be installed in the system. In this case, an attempt to access files associated with the drive letter assigned to the socket will result in a drive not ready error.

The Flash File System

Two primary types of flash file system solution are provided by software vendors today. These systems are generally referred to as the flash file system (FFS) and flash translation layer (FTL) as illustrated in figure 21-3. The FFS provides file management based on variable size data blocks, while the FTL interfaces directly to the DOS file system which allocates data based on standard block sizes. The FTL system is compatible with disk utility programs such as Norton and PC Tools, whereas, the FFS-based systems are not.

MTD Registers with Card Services

The MTD registers with card services by calling the RegisterClient service. When registering the MTD specifies that it is a MTD client during , specifies relevant call-back events it want to be notified of, and passes a pointer to its call-back routine. The MTD client may also request that card services generate a card insertion call-back for each PC Card already installed in the system. The MTD client receives a handle value from card services when it returns from the call. Once registered, the MTD awaits call-backs from card services, notifying it when a PC card is inserted or removed.

The MTD Call-Back

When card services generates the card insertion call-back, it also passes the logical socket number that the PC Card was inserted into. The MTD client then determines if the PC Card contains any flash memory that it is designed to access. This can be accomplished by calling the GetFirstRegion and GetNextRegion services. These services return information (obtained from the CIS) about the card type, size, location, access time, and block erase details of the regions. If the MTD recognizes a regions of memory that it knows how to access, it then registers with card services to control access to that specific region of memory.

PCMCIA System Architecture

MTD Registers Memory Regions

To register a memory region with card services the MTD calls the RegisterMTD service. This notifies card services that the MTD has agreed to handle access to the memory regions specified. When a flash client driver requests access to this region via bulk memory services, card services will make an MTDRequest call-back to the MTD. The information specified in the call-back packet specifies the operation be requested.

Flash Client Driver Registers with Card Services

The client driver performs the registration process by calling the RegisterClient service. The flash client identifies itself as a memory client, during registration and specifies which call-back events it wishes to be notified of, and passes a pointer to its call-back routine. The client may also request that card services generate a card insertion call-back for each PC Card already installed in the system. The memory client receives a handle value from card services when it returns from the call. Once registered, the memory client awaits call-backs from card services, notifying it when a PC card is inserted or removed.

The Flash Client Driver Call-Back

When card services generates the card insertion call-back, it also passes the logical socket number that the PC Card was inserted into. The memory client then attempts to configure the PC Card.

The memory client must first determine if the card is the type that it is designed to enable. Memory clients can use the bulk memory services to access a specific region within the PC card. To access memory, the client first calls the OpenMemory service by specifying an offset within the card's attribute or common memory address space. Card services then returns a memory handle to the client for use when accessing memory starting at the offset specified in the OpenMemory service. Also when the OpenMemory service is called, card services recognized the region being opened is registered by the MTD. Note that if card services does not support bulk memory services, the memory client must use the RequestWindow service to specify the host system address space that it wishes to use to access PC Card memory.

Chapter 21: Client Drivers

Reading from or writing to PC Card memory is accomplished by calling the ReadMemory or WriteMemory services. In this instance, the memory client passes the memory handle it received from the OpenMemory service and specifies the memory offset and range of addresses it wishes to access. The call will likely specify location zero within the attribute memory address space. When the data is returned to the memory client it evaluates the DEVICE tuple to determine if the card contains flash memory.

If a flash card is detected the call-back routine returns to card services, indicating that the card was successfully configured. If the PC Card was not a flash card, the client returns to card services, indicating the card was not enabled by the flash client driver.

Accessing Flash Memory

Once the flash card has been enabled, access made to the flash card virtual drive will be fulfilled. The flash client driver receives the request from the flash file system and calls the appropriate bulk memory service. Card services recognizes that the call is to a region controlled by an MTD that previously register to access the specified memory region. Card services responds by making MTD call-backs to specify the operation being requested.

I/O Card Client Drivers

Two basic types of I/O client drivers are popular.

- Device-specific client drivers — drivers designed to detect and configure a specific PC Card. The client drivers are typically shipped by the manufacturer of a PC Card and are designed to configure this specific PC Card.
- Generic (Super) client drivers — drivers designed to detect and configure a wide range of I/O cards based on generic types, regardless of manufacturer.

Each type of I/O client driver install as device drivers when the config.sys file is executed during the system boot process. Figure 21-4 illustrates the primary actions taken by an generic I/O client driver when it initializes, registers with card services and attempts to configure PC Cards.

PCMCIA System Architecture

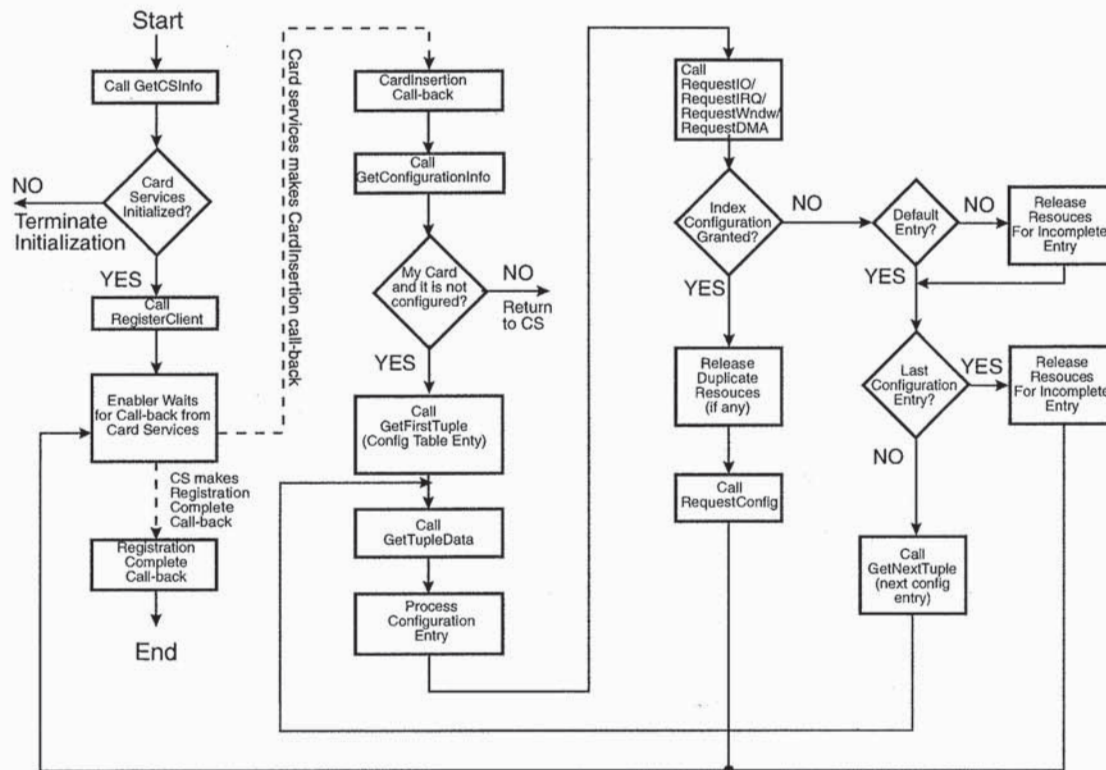


Figure 21-4. I/O Enabler Registration and PC Card Configuration Process

I/O Client Driver Registers with Card Services

The registration process begins after the client driver detects that card services has initialized. This is accomplished with the GetCardServicesInfo call. Card services returns information about card services and verifies its presence by also returning the ASCII string "CS." If card services is not initialized, the I/O client driver reports the error condition and terminates without remaining resident in memory. If card services are present the client driver calls the RegisterClient service. When the I/O client driver makes the call it:

- identifies itself as an I/O client,
- specifies which events it wants to be notified of,
- requests a card insertion call-back for each PC Card currently installed in sockets, and
- passes the entry point of it call-back routine when making the call.

Chapter 21: Client Drivers

Card services returns a client handle to the I/O client driver upon return from the RegisterClient service. The client driver then awaits card insertion call-backs. Card services generates a CardInsertion call-back for each PC Card already installed in a card socket (as requested by the client driver during registration). If all sockets are empty, card services generates a ConfigurationComplete call-back to signal the end of the configuration process. This example presumes that an I/O PC Card was installed when the system was powered on.

The I/O Client Driver Call-Back

Card services generates a CardInsertion call-back to the I/O client driver. The driver detects the call-back and evaluates the call-back packet to determine the socket into which the PC Card was inserted.

Identifying the PC Card

Next the GetConfigurationInfo service is called to determine if the PC Card has already been configured by another enabler. If already configured, the client driver returns to card services without configuring the PC Card. If the PC Card has not been configured, the client driver evaluates other data returned by the GetConfigurationInfo service to determine the type of function that is associated with the PC Card. If the function is one that the generic I/O enabler is designed to handle, the configuration process continues.

Determining Resources Requirements

Next, the client driver checks the first entry within the configuration table to determine the resources required by the card. This can be accomplished by calling the GetFirstTuple service and specifying a tuple code of 1Bh (the configuration table entry tuple code). Card services scans the CIS until it finds the first instance of tuple 1Bh and returns to the client driver. The I/O driver checks the completion status and detects that card services has located the first configuration table entry. Next, the client driver calls GetTupleData and card services returns the contents of the first configuration table entry. The tuple data is evaluated to determine the resources required by the PC Card.

PCMCIA System Architecture

Requesting the Resources

As the client driver detects a configurable resource within the configuration table entry (e.g., a range of I/O address locations), it checks with card services to determine if the resource is available for the I/O card to use. The client requests a resource by calling the respective resource request service (e.g., RequestIO). When RequestIO is called Card services receives the base I/O address and range of address locations requested. It uses these values to perform a look-up within the resource management table to determine if the resource is available. Card services indicates whether or not the resource was available in the return code.

The client driver makes requests for all resources listed within the configuration table entry and determines their availability. The configurable resources that can be acquired from the system include:

- Memory address locations — via the RequestWindow service
- I/O address locations — via the RequestIO service
- Interrupt request lines — via the RequestIRQ service
- DMA channels — via the RequestDMA service

The specific actions taken when a resource is not available depends of whether the entry is a default entry or not, as discussed in the following paragraph.

If the client driver detects that the entry is tagged as a default entry, it knows that it should attempt to acquire all resources that are specified within this entry. It should retain all resource acquired from card services even if one or more of the resources requested are not available. If the entry is not a default entry, the client driver knows that the entire set of resources specified within entry must be obtained to satisfy the configuration. If any one resource is not available, then the client driver should release any individual resources that were acquired from card services by calling the respective release resource service (e.g., ReleaseIO).

If a given entry fails to satisfy the PC Card's configuration, the client driver then proceeds to the next entry by calling the GetNextTuple service. Card services finds the next configuration table entry (tuple 1B) and the client driver calls GetTupleData and starts the resource acquisition process again.

Chapter 21: Client Drivers

Configuring the PC Card

When all resources needed for the PC Card's configuration have been acquired from the system, the client driver configures the HBA and PC Card by calling the RequestConfiguration service. In response, card services configures the HBA and PC Card. The HBA is configured by loading the appropriate HBA registers (via socket services) that satisfy the configuration being requested. This includes reconfiguring the socket interface to memory or I/O, programming the I/O window registers, and programming the IREQ# steering logic to direct the IRQ to the appropriate system IRQ line. The PC Card is configured by writing the index number of the configuration table entry (the entry that satisfied the configuration) into the configuration option register (COR) within the card's attribute memory address space.

Point Enablers

Point enablers are dedicated enablers that bypass card and socket services. These enablers are popular in environments such as DOS where limited memory address space is available for application programs. Card and socket services take a considerable amount of memory when they install. Added to this is the space required by the enabler(s) and any TSR (terminate and stay resident programs) that might be used. As a result, too little memory is left for many application programs to run.

One solution used to relieve this memory shortage, is to remove card and socket services from the system, thereby freeing up memory that is needed to run the application program. Eliminating the card and socket services prevents PC Card client drivers from performing their functions, thus PC Cards are never enabled and cannot be used. In order to use PC Cards point enablers are needed to configure the PC Cards.

In the absence of card and socket services, point enablers must communicate directly with the HBA to load the appropriate registers necessary to gain access to the PC Card. The card's CIS must be read and interpreted to identify the PC Card. If the point enabler recognizes the PC Card, it attempts to configure the card by loading the appropriate registers within the HBA to satisfy the configuration, and by writing to the configuration registers to configure the PC Card. Note that the resources used to configure the PC Card must be specified manually by the user (typically via software switches).

Chapter 22

The Previous Chapter

The previous chapter discussed the three basic types of enablers: point enablers, device-specific enablers, and generic (super) enablers. The chapter also discussed the jobs performed by generic memory enablers (and MTDs) and I/O device enablers.

This Chapter

This chapter discusses the problems associated with loading the operating system from a PC Card. It also defines mechanisms used to determine whether a given PC Card is a bootable device, and the firmware support required to support PC Card booting.

The Next Chapter

The next chapter introduces execute-in-place (XIP) support defined by the PC Card standard. The major components of an XIP environments are specified and the XIP mechanism is described.

Configuring PC Cards During POST

The previous discussions of PC Card configuration have presumed that PC Cards will be initialized either as the operating system loads or when the PC Card is inserted into a socket sometime after the operating system has loaded and the system is running. If however, the need to load the operating system from a PC Card exists, the previously discussed approaches for configuring the cards don't work.

PCMCIA System Architecture

The Problem

The normal method used in the PC environment to configure and initialize bootable devices (i.e., hard drive, video controller, and LAN adapters that support remote boot from the network) requires that the bootable device contain initialization code in a device-specific ROM. The system initialization code contained within system ROM scans the region of memory address space from location C0000h-DFFFFh to detect the presence of a device ROMs (i.e., a bootable devices). If a device ROM is detected, the system calls the initialization routine within the device ROM which is responsible for configuring the device. In this way, the bootable device is configured and can participate in loading the operating system.

To perform initial program load (IPL) from a PC Card, it too must be configured prior to beginning the boot operation. The standard method of configuring PC Cards requires the use of socket services, card services, and the PC Card's client driver. This software doesn't initialize until the operating system loads, making it unavailable for configuring a PC Card that must be used to load the operation system. Even if the PC Card contains a device ROM it cannot be detected by the system firmware during the ROM scan (because a memory window must first be programmed within the HBA to gain access to PC Card memory).

The Solution

ROM-based PCMCIA initialization code must be included with the system to support IPL from PC Cards. This firmware code must be able to program the HBA to open an attribute memory window to permit access to the CIS. Then the CIS can be evaluated to determine if the PC card is bootable, and therefore, should be configured during the POST (Power-On Self Test) sequence.

Bootable Memory Cards

The PCMCIA initialization firmware detects the presence of memory cards and configure them by opening a common memory window to provide access to the PC Cards memory array (i.e., virtual drive). The boot code being PC Card aware attempts to load the operating system from the memory card. If the memory card has been formatted and the system files reside within the

Chapter 22: Booting from PC Cards

memory card, the firmware will recognize the boot sector and load the operating system from the memory card.

Note that once the operating system loads, the memory cards will no longer be accessible unless the memory card contains a config.sys file that lists all of the PCMCIA relevant drivers. In this case, the socket services, card services, and the associated client drivers are loaded, thereby, providing access to the PC Cards after the operating system has loaded.

Bootable ATA Devices

PCMCIA initialization firmware recognizes ATA PC Cards by evaluating the function identification tuple within the CIS (table 22-1). The function identification tuple indicates the device type as shown in table 22-2. The shaded area identifies the value used by the ATA PC Card.

Table 22-1. Format of the Function Identification Tuple

Byte	Function Identification Tuple Format	
0	TPL_CODE	CISTPL_FUNCID (21H)
1	TPL_LINK	Link to next tuple (at least 2)
2	TPLFID_FUNCTION	PC Card function code
3	TPLFID_SYSINIT	System initialization bit mask

Note that function extension tuple will follow the function identification tuple that identify specific features associated with the ATA card (Refer to the chapter entitled, "An ATA PC Card Example"). The function identification tuple also includes an initialization byte that specifies whether the device should be configured during POST and whether the ATA card contains a device ROM. (See table 22-3.)

If the initialization byte indicates that the ATA card should be configured during POST but that it does not contain a device ROM, then the firmware is responsible for configuring the ATA card. Once the ATA card is configured, the operating system can boot directly from the drive.

PCMCIA System Architecture

Table 22-2. Contents of the Function Identification Byte

Code	Name	Meaning
0h	Multi-Function	PC Card has multiple functions. Examine the following function identification tuples that follow for individual functions.
1h	Memory	Memory Card (RAM, ROM, EPROM, flash, etc.).
2h	Serial Port	Serial I/O port, includes modem cards.
3h	Parallel Port	Parallel printer port, may be bi-directional.
4h	Fixed Disk	Fixed drive, may be silicon may be removable.
5h	Video Adapter	Video interface, extension tuples (type and resolutions supported).
6h	Network LAN Adapter	Local Area Network adapter.
7h	AIMS	Auto-Incrementing Mass Storage card.
8..FFh	Reserved	Unused in this release. Reserved by PCMCIA for future use.

Table 22-3. Contents of the Initialization Byte

7	6	5	4	3	2	1	0
Reserved for future use, must be set to zero (0)						ROM	POST

If the ATA drive also contains a device ROM, then firmware can map the ROM into the ROM scan region (C0000h-DFFFFh) and the standard initialization process will detect the device ROM. The ATA device ROM containing the ATA enabler and driver will be called by PCMCIA firmware. The ATA's device ROM performs the configuration process and returns to the system firmware. Once the drive is configured, IPL can occur from the PCMCIA ATA drive like any other ATA drive.

Chapter 23

The Previous Chapter

The last chapter discussed the problems associated with loading the operating system from a PC Card. It also defined mechanisms used to determine whether a given PC Card is a bootable device, and the firmware support required to support booting from PC Cards.

This Chapter

This chapter discusses the Execute-In-Place mechanism defined by PCMCIA that allows code to be executed directly from the card rather than copying files to and executing from system memory.

The Next Chapter

The next chapter introduces the ExCA (QuickSwap) specification that defines a required set of hardware and software support that is intended to improve PC Card interoperability across platforms based on the Intel X86 architecture.

The XIP Goals

Execute-In-Place (XIP) provides a mechanism for application programs to execute directly from PC Card memory. This eliminates the need to copy code from the PC Card into host memory before being executed, reducing the amount of system memory address space needed to load and execute a large application program. This is a particular concern in the DOS operating environment where memory address space is at a premium. Application programs written to support XIP could be supplied on a ROM-based PC Card or could be loaded from disk to a memory card (such as flash) and be executed directly from the PC Card.

PCMCIA System Architecture

Similar techniques, including the popular expanded memory specification (EMS), allow an application to reside in memory outside the memory address range that is addressable by DOS. Small portions (16KB pages) of these remote memory ranges are mapped into areas addressable by DOS, permitting them to be accessed. The EMS protocol defined in the Lotus/Intel/Microsoft (LIM) specification is supported by XIP and is called LXIP. Additionally, XIP defines support for applications designed to use extended memory (address space beyond 1MB) using Intel 80386 compatible addressing modes. This form of XIP is termed EXIP. A new type of XIP called SXIP (Simple XIP) is defined for systems with very limited paging mechanisms and small address space. The execution and read-only data images require no more than 64KB of address space.

The XIP Software Hierarchy

The functions performed by XIP software includes:

- Setting up XIP partitions in PC Card memory.
- Establishing directories within PC Card memory.
- Copying XIP applications into the XIP partitions.
- Mapping the application within the processor's addressable space.
- Starting the XIP application execution.
- Providing services for the XIP application so that it can manage program execution.

XIP File Management

XIP applications do not use the normal DOS File Allocation Table (FAT) or Flash File System (FFS). Instead, XIP applications use a dedicated software interface consisting of XIP utilities, XIP management software and socket services to map the PC Card memory into an XIP partition. The XIP software can only execute an XIP application from an XIP partition. An XIP partition can be set up in PC Card memory by utility programs. The PCMCIA specification details the organization and data structures required for partitions and directory entries.

Chapter 23: Execute In Place (XIP)

The XIP Loader

Once the dedicated XIP partition exists then an XIP application can be loaded into the PC Card's common memory address space within an XIP partition. The XIP directory also contained within PC Card memory is then updated to reflect the application's presence. An XIP application from the user perspective begins execution in the same way that a DOS application does (by typing the name of the executable file). In this case, however, an XIP loader is invoked when an executable XIP file is called. The XIP loader's task is to find the XIP application that resides within an XIP partition in PC Card memory. The loader searches for the application within the XIP directory, maps the application into system address space and starts the application.

The XIP Device Drivers (API and Hardware Manipulation)

Once started, the application manages program execution by making the necessary calls to the XIP driver. The PCMCIA specification defines all of the functions needed by the application. The XIP device-driver functionality is split between a high-level driver (XIP.SYS) and a low-level driver (PCMCIA.SYS). The high-level driver is implemented as an installable device driver and provides all the services needed by the XIP application. This provides the XIP application's API. The low-level driver provide services for the high-level driver when it needs to access the memory-mapping hardware within the HBA. It can be installed as an installable device driver or included in the system's BIOS routines.

The intent of the split driver approach is to remove the details of the hardware interface from the high-level driver, making it easy to implement a generic XIP driver that can be used with any XIP-capable system. The system manufacturer then need only concentrate on developing the low-level driver used to manipulate the hardware (the same as the related socket service functions).

PCMCIA System Architecture

LXIP

LXIP is compatible with the LIM 4.0 specification. This protocol requires that four separate 16KB blocks of contiguous memory address space, called page frames, be mapped into the processor's memory address space. Each of these four page frames must permit access within the PC Card's XIP application, which is also organized into 16KB blocks, called pages. An LXIP application is aware of this organization and interacts with the LXIP manager to access PC Card memory via the page frames.

A PCMCIA host bus adapter designed to support the LXIP capability must have the ability to map these four 16KB address ranges independently. The LXIP manager accepts requests from the XIP application and sets up access to PC Card memory via the socket services interface and the HBA.

EXIP

EXIP specifies the ability of applications to execute directly from PC Card memory when the memory card is mapped into the processor's extended address space (above 1MB). The EXIP manager determines where PC Card memory will be allocated in extended memory and programs the HBA to map the card into extended memory.

SXIP

SXIP applications are quite small and cannot exceed more than 64KB of address space. In this respect they are similar to .com programs that execute in a single x86 memory segment. The entire program image is directly mapped into the processor's address space and no remapping or paging is performed.

Part Five

ExCA(QuickSwap)

Chapter 24

The Previous Chapter

The previous chapter described the Execute-In-Place (XIP) functionality provided by PCMCIA that allows code to be executed directly from files stored on PC Cards. Three types of XIP were defined: one for small applications (SXIP), one based on expanded memory concepts (LXIP) and the other for applications using extended memory (EXIP).

This Chapter

This chapter introduces the ExCA (QuickSwap) specification that defines a required set of hardware and software support, intended to improve PC Card interoperability across platforms based on Intel x86 architecture.

The Next Chapter

The next chapter provides a sample PCMCIA host bus adapter. The adapter documented is the Cirrus Logic PD6722 designed for use in x86 PC-based systems.

The ExCA Goal

The Intel ExCA (Exchangeable Card Architecture) specification provides specific HBA, PC Card, and software requirements for systems implementing DOS-based Intel x86 compatible systems. By defining minimum hardware and software requirements for these systems, Intel hopes to ensure PC Card compatibility across x86 systems implementing the ExCA standard.

The need for such a standard stems from the flexibility incorporated into the PCMCIA specification. The standard was designed to provide latitude for designers who are developing PCMCIA solutions over a wide range of PC and

PCMCIA System Architecture

non-PC platforms. However, this latitude, while achieving its goal, also creates a greater possibility of compatibility problems.

ExCA Scope

In most respects, the ExCA specification defines a subset of the features within the PCMCIA standards, narrowing down the implementation possibilities and reducing the risk of PC Cards and systems being developed that are incompatible with one another. Additionally, ExCA defines some system characteristics not specified in the PCMCIA standard. The ExCA specification describes the minimum capabilities of the following items:

- The ExCA Host Bus Adapter
- Socket Services
- Card Services
- PC Cards (both memory-only and memory and I/O)

ExCA also encompasses a three phase compliance test, including socket hardware functional testing, system software functional testing and system integration testing.

This chapter highlights the ExCA specification's features. Refer to the ExCA specification for complete details.

ExCA Host Bus Adapter Requirements

Host bus adapter requirements fall into the following categories:

- Address Mapping (memory and I/O)
- Interrupt Support
- System Power
- PC Card Insertion and Removal
- Event WakeUp (i.e. ring indicate when system is in sleep mode)

Address Mapping (memory and I/O)

Specific requirements exist for ExCA compliant host bus adapters to ensure that address windowing capability provides the features needed in DOS-based operating environments. Address mapping features are described for

Chapter 24: ExCA (QuickSwap)

ExCA compliant sockets for both memory sockets and Memory or I/O sockets.

Memory Address Mapping

Each socket must include a minimum of four memory windows that can be acquired and used by a socket. This requirement provides support for expanded memory (L-XIP) in which four separate 16KB address ranges must be acquired from system memory and mapped to the PC Card. Support must also exist for each socket to provide a fifth window, thereby allowing access to attribute memory when necessary.

Each memory window must support system address capabilities for both real mode (within the first 1MB of memory address space) and protected mode (above 1MB of memory address space) operation. Furthermore each ExCA memory address window must have the following capabilities and characteristics:

- windows are mappable anywhere between 256KB to 16MB (in host space)
- minimum window size of 4KB
- maximum window size of 256KB (real mode)
- maximum window size of 8MB (protected mode)
- window size can be any 4KB increment (4, 8, 12, 16, 20 KB) or may be a power of two size (4, 8, 16, 32, 64 KB)

Consistent with the PCMCIA specification, memory windows are not allowed to overlap in system address space, unless use of the address range is time multiplexed.

I/O Address Mapping

ExCA requires that at least two I/O windows be implemented per socket. Characteristics of ExCA I/O windows include:

- minimum window size of 1 byte
- maximum windows size of 256 bytes
- window size must be power of two (1, 2, 4, 8, 16, 32 bytes)

Note that no remapping of the system I/O address is required. Addresses are directly mapped from system address locations to the same locations on the PC Card.

PCMCIA System Architecture

ExCA does not support overlapping I/O windows without time multiplexing them, as is required for overlapping memory windows. This means that no support need exist for the INPACK# signal on ExCA compliant adapters.

Interrupt Support

ExCA adapters generate a status change interrupt for all card status change events defined in the PCMCIA specification and they redirect or steer PC Card interrupts to system IRQ lines as required.

Status Change Interrupt

ExCA adapters generate a single status change interrupt for card events from all adapter sockets. Software must have the ability to globally select which type of card events generate a status change interrupt. Additionally, individual events can be masked at the socket, providing selection of specific events that generate a status change interrupt on a per socket basis. Support must also exist for enabling and disabling the status change interrupt under software control.

The adapter captures all status change events reported by each socket so that software can determine which socket encountered the status change event. The actual state of the status change signals from each socket can also be read directly from the adapter.

Status change events from I/O cards are reported when an I/O card asserts its status change pin. Status change information must be read directly from the I/O card's configuration register (pin replacement register).

PC Card Interrupts

A PC Card interrupt must be steerable to any available system interrupt. Availability depends on the host system implementation as listed in table 24-1.

ExCA compliant systems must ensure that at least one interrupt is available for standard communications and local area networks (LANs). In other words, the system must supply at least one interrupt request line from each bullet list that follows.

Chapter 24: ExCA (QuickSwap)

Table 24-1. Interrupts Potentially Available For Use By PC Cards

Systems with One Interrupt Controller	Systems with Two Interrupt Controllers	
IRQ 2	IRQ 2/9	IRQ 10
IRQ 3	IRQ 3	IRQ 11
IRQ 4	IRQ 4	IRQ 12
IRQ 5	IRQ 5	IRQ 14
IRQ 7	IRQ 7	IRQ 15

Standard Communications Interrupts (Serial Port)

- IRQ 3
- IRQ 4

Standard LAN Interrupts

- IRQ 5
- IRQ 7
- IRQ 10
- IRQ 11
- IRQ 15

Note that ExCA recommends that all interrupts listed in table 24-1 be supported by the adapter. However, a given implementation may choose to use only a subset since the system design likely uses some of the interrupts.

Interrupt sharing support is system dependent. Systems based on ISA host buses do not support interrupt sharing, while systems based on Micro Channel and EISA can share interrupts. Micro Channel and EISA devices use level sensitive interrupt triggering to support sharing, thus cards that support only the PCMCIA specified pulse-mode interrupts will not behave according to the level sensitive triggering protocol. ExCA compliant adapters must support level-mode interrupts from the PC Card, while pulse-mode support is optional.

PCMCIA System Architecture

System Power Requirements

ExCA systems must supply minimum power requirements as indicated in the ExCA specification. PCMCIA compliant PC Cards requiring more power than specified by the ExCA specification may not operate correctly when installed in sockets that are ExCA compliant. Additionally, ExCA compliant systems need not provide separate programmable voltages for Vpp1 and Vpp2. Refer to the ExCA specification for actual power requirements. The voltage supply combinations that must be provided at the socket include those listed in table 24-2.

Table 24-2. ExCA Voltage Requirements

	Vcc	Vpp1	Vpp2
required	0v	0v	0v
required	5v	5v	5v
required	5v	12v	12v
optional	5v	0v	0v

PC Card Insertion/Removal

The ExCA specification defines the sequence of events, interface signal status, Vcc and Vpp levels and critical timing delays for PC Card insertion and removal. The ExCA specification supports both cold socket insertion (recommended) and warm insertion (not recommended). Hot socket insertion of PC Cards is not supported by the ExCA. Table 24-3 defines the difference between cold, warm and hot PCMCIA sockets.

Table 24-3. State of Socket When PC Card is Inserted

Socket State	Vcc and Vpp	Address State at Signal Contact	Data State at Signal Contact	Control Signal State at Signal Contact
Cold Socket	Off	High Z or 0v	High Z or 0v	High Z or 0v
Warm Socket	On	High Z or 0v	High Z or 0v	High Z or 0v
Hot Socket	On	Driven Active	Driven Active	Driven Active

Chapter 24: ExCA (QuickSwap)

Card Insertion

The ExCA specification defines the sequence of events and minimum time duration for these events when a card is inserted into a socket. The sequence of events is listed below.

1. Card inserted into socket (both CD1# and CD2# asserted)
2. Adapter applies Vcc
3. Adapter asserts reset to PC Card
4. Adapter removes reset and PC Card begins initialization
5. Initialization completes within 20 ms or else deasserts READY
6. Client driver polls READY to detect when PC Card is ready to be accessed.

Card Removal

The ExCA specification also defines the sequence of events that are recommended when the PC Card is removed from the socket as listed below. Note that when a PC Card is removed from the system, the socket interface may be active.

1. Adapter detects card removal (CD1# and/or CD2# deasserted)
2. Adapter ceases to drive active signals to the interface (address, data and control signals go to high impedance state or 0v)
3. Vcc removed from the socket (not required if warm socket insertion is supported)

Note that the adapter detects that a card is being removed before any of the other interface or power pins lose contact with the socket (because the Card Detect pins are shortest). Next, the adapter releases the interface by tri-stating the address, data and control lines, (which are the intermediate length signal pins), and finally removes power to the Vcc pins (which are the longest pins). As the PC Card is removed it is still in contact with the signals pins and power pins long after they are disabled by the adapter.

ExCA Socket Services

The ExCA specification defines a minimum subset of socket services functions that are required for ExCA compliance. Table 24-4 lists the socket services functions and notes those that are required versus optional.

PCMCIA System Architecture

Table 24-4. Socket Services Functions Required/Optional for ExCA Compliant Systems

Function	Required ?
GET_ADP_CNT	Yes
GET_SS_INFO	Yes
INQ_ADAPTER	Yes
GET_ADAPTER	Yes
SET_ADAPTER	Yes
INQ_WINDOW	Yes
GET_WINDOW	Yes
SET_WINDOW	Yes
GET_PAGE	Yes
SET_PAGE	Yes
INQ_SOCKET	Yes
GET_SOCKET	Yes
SET_SOCKET	Yes
GET_STATUS	Yes
RESET_SOCKET	Yes
INQ_EDC	No
GET_EDC	No
SET_EDC	No
START_EDC	No
PAUSE_EDC	No
RESUME_EDC	No
STOP_EDC	No
READ_EDC	No
GET_VENDOR_INFO	No
ACK_INTERRUPT	Yes
PRIOR_HANDLER	No
SS_ADDR	No
ACCESS_OFFSETS	No
VEND_SPECIFIC	No
CARD_SERVICES	Yes

Note that the EDC and vendor specific functions are optional for ExCA compliant socket services. The implementation and definition of the required socket services functions are compliant with the PCMCIA socket services standard.

Chapter 24: ExCA (QuickSwap)

ExCA Card Services

ExCA compliant systems must support card services; but, like socket services not every card services function is required. Table 24-5 lists the card services functions that are required.

Table 24-5. Card Services Functions Required/Optional For ExCA Compliance

Function	Required ?
Client Services Functions	
GetCardServicesInfo	Partial
RegisterClient	Yes
DeregisterClient	Yes
GetStatus	Yes
ResetCard	Yes
SetEvenMask	Yes
GetEvenMask	Yes
Resource Management Functions	
RequestIO	Yes
ReleaseIO	Yes
RequestIRQ	Yes
ReleaseIRQ	Yes
RequestWindow	Yes
ReleaseWindow	Yes
ModifyWindow	Yes
MapMemPage	Yes
RequestSocketMask	Yes
ReleaseSocketMask	Yes
RequestConfiguration	Yes
GetConfigurationInfo	Yes
ModifyConfiguration	Yes
ReleaseConfiguration	Yes
Bulk Memory Services Functions	
OpenMemory	No
ReadMemory	No
WriteMemory	No
CopyMemory	No
RegisterEraseQueue	No
CheckEraseQueue	No

Function	Required ?
DeregisterEraseQueue	No
CloseMemory	No
Client Utilities Functions	
GetFirstTuple	Yes
GetNextTuple	Yes
GetTupleData	Yes
GetFirstRegion	No
GetNextRegion	No
GetFirstPartition	No
GetNextPartition	No
Advanced Client Services Functions	
ReturnSSEntry	Yes
MapLogSocket	Yes
MapPhySocket	Yes
MapLogWindow	Yes
MapPhyWindow	Yes
RegisterMTD	No
RegisterTimer	Yes
SetRegion	No
ValidateCIS	Yes
RequestExclusive	Yes
ReleaseExclusive	Yes
GetFirstClient	Yes
GetNextClient	Yes
GetClientInfo	Yes
AddSocketServices	No
ReplaceSocket Services	No
VendorSpecific	No
AdjustResourceInfo	Yes
AccessConfigurationRegister	No

PCMCIA System Architecture

ExCA PC Cards

ExCA recommends which tuples an ExCA compliant PC Card should implement. Table 24-6 below lists the recommended tuples for memory and I/O PC Cards. The lower portion of the table lists three tuples that might contain information needed by system initialization code or peripheral installation software for determining if a PC Card should be installed and configured during POST (Power-On Self Test) prior to loading the operating system. This capability is needed primarily for those devices that must be used to load and install the operating system.

Table 24-6. Tuples Recommended by the ExCA Specification

Tuples Recommended by ExCA	Memory Cards ?	I/O Cards ?
Device Information	Yes	Yes
Level 1 Version/Product Information	Yes	Yes
Configuration	Yes	Yes
Configuration Table Entry	Yes	Yes
JEDEC Device ID	Yes	No
Device Geometry Information (flash)	Yes	No
Recommended for bootable PC Cards		
Card Manufacturer ID	Yes	Yes
Function ID	Yes	Yes
Function Extension	Yes	Yes

PC Card Event WakeUp

Systems implementing power conservation modes, such as suspend or sleep, may want to wake the system up if some critical event occurs at the PC Card. Events, such as a call to a modem, could be used to wake the system up and return to normal full power operation so that the event can be processed. Currently the PCMCIA specification (release 2.1) does not define an event wakeup procedure, and in its absence, ExCA defines the following optional definition for event wakeup.

Two events can cause event wakeup in an ExCA compliant system:

- Ring Indication from a modem or fax
- remote power up from a LAN card

Chapter 24: ExCA (QuickSwap)

ExCA compliant HBAs and PC Cards use socket pin 63 (Status Change) for event wakeup, replacing either the READY, Write Protect or Battery Voltage Change status change indication on the PC Card. The PC Card indicates its support of event wakeup via the CIS.

The Configuration Table Entry tuple identifies the card's capability for using event wakeup via pin 63. The configuration entry tuple contains a miscellaneous features field that can be used to specify which status change indicators are supported by the card and is used to indicate which status change event that the event wakeup mechanism uses. The host bus adapter is programmed to direct the status change indication to the power management interrupt, which requests that the system return to full power operation.

PCMCIA System Architecture

Part Six

An Example HBA

Chapter 25

The Previous Chapter

The previous chapter introduces the ExCA (QuickSwap) specification that defines a required set of hardware and software support, intended to improve PC Card interoperability across platforms based on Intel x86 architecture.

This Chapter

This chapter provides an overview of a sample PCMCIA host bus adapter (The Cirrus Logic CL-PD6722) used in Intel x86 implementations for either an original PC or ISA compatible host bus.

Introduction to the CL-PD6722

This chapter is intended as a brief look at an actual PCMCIA host bus adapter. The Cirrus Logic CL-PD6722 was chosen as the example adapter for several reasons. First, the Intel 82365 PCMCIA adapter chip is currently implemented in more systems than any other, and the CL-PD6722 is register compatible with the Intel chip, with a few minor exceptions. The second reason is that it includes considerably more functionality than the Intel chip.

The CL-PD6722 controls two PCMCIA sockets via a single 208-pin PQFP. Features of the CL-PD6722 include the following:

- PCMCIA 2.1 and JEIDA 4.1 Compliant
- Intel 82365SL (Step A) compatible register set, ExCA compliant
- ISA host bus interface
- Dual socket interface
- Automatic Low-Power Dynamic Mode
- Programmable Suspend Mode for power conservation
- Five programmable memory windows per socket
- Two programmable I/O windows per socket
- 8-bit or 16-bit host bus interface

PCMCIA System Architecture

- ATA disk interface support for small form-factor drives
- DMA support
- Mixed-Voltage operation (3.3v or 5v) operation

Socket Power Control

The CL-PD6722 uses the PowerGood signal from the system's power supply as its reset. When PowerGood transitions from low to high the CL-PD6722 leaves the reset state and begins operation. Power to the socket is controlled by chip outputs that go to power switching devices. As shown in figure 25-1, the CL-PC6722 has four output signals per socket that control power to the socket as follows:

- Vcc_5 - when asserted 5v is applied to socket Vcc
- Vcc_3 - when asserted 3.3v is applied to socket Vcc
- Vpp_Vcc when asserted Vcc is applied to socket Vpp1.
- Vpp_PGM when asserted the programming voltage (12v) is applied to socket Vpp1.

Internal registers determine which of these signals will be asserted and when.

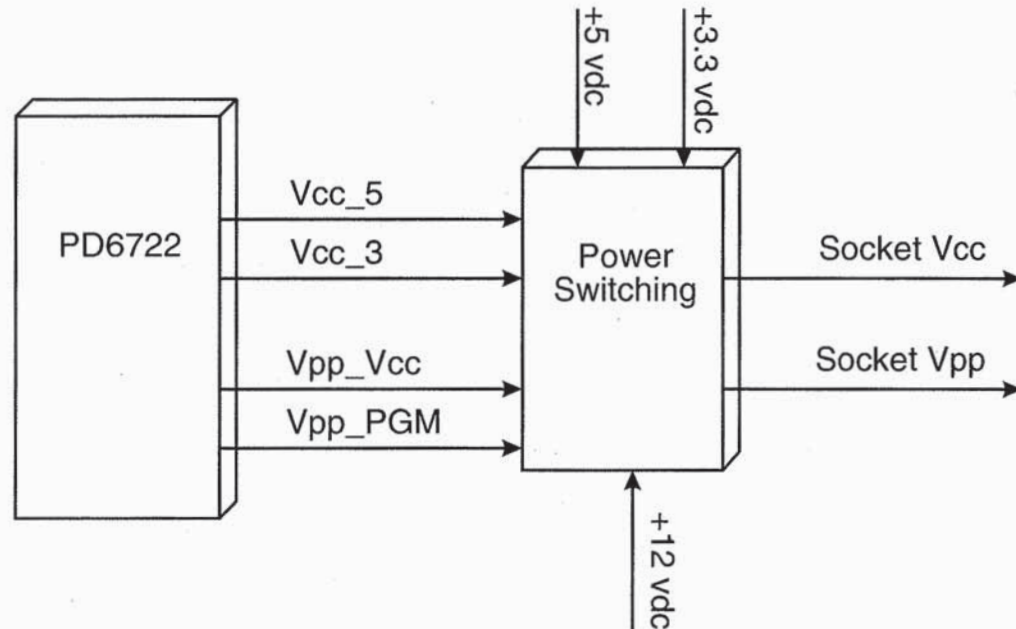


Figure 25-1. CL-PD6722 Socket Power Control Signals

Chapter 25: An Example HBA —The CL-PD6722

Vcc Control

In PCMCIA 2.1 compliant systems, Vcc to the socket must always be 5v, but can be switched to 3.3v if the PC Card indicates the ability to operate at 3.3v via the CIS. If 3.3v operation is supported, software will write to Miscellaneous Control 1 register, specifying that Vcc should be switched to 3.3v. (Note that the CL-PD6722 provides limited support for Vcc sensing and can be used in new designs that incorporate the low voltage connectors. Contact Cirrus Logic for details.)

The CL-PD6722 supports two methods of applying Vcc to the socket:

- Vcc control via the client driver, card service and socket services software chain when a card is detected.
- Automatic Vcc control via the CL-PD6722 controller

When a card is inserted into the socket, the -CD pins are asserted and the adapter detects the card's presence. When autopower mode is not selected, the adapter waits to be commanded by the software before applying Vcc to the socket. Software must set bit four in the Power Control register (Refer to figure 25-2) to enable power to the socket. The adapter responds by asserting the Vcc_5 signal.

If bit five is set during system initialization, the adapter automatically supplies Vcc to the adapter (asserts Vcc_5) when it detects the presence of a card. Vcc is automatically removed from the card when the card is removed. Note that power is removed based on timing parameters specified in the ExCA specification.

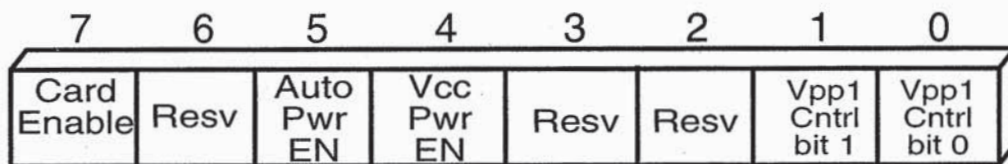


Figure 25-2. The Power Control Register

PCMCIA System Architecture

Vpp1 Control

Bits zero and one of the Power Control register determine whether Vcc (5v or 3.3 v), 12v, or zero volts is applied to the Vpp1 pin. (Refer to table 25-1)

Table 25-1. Socket Vpp Control

Bit 1	Bit 0	VPP_PGM	VPP_VCC	Socket Vpp1
0	0	Deasserted	Deasserted	zero volts
0	1	Deasserted	Asserted	selected Vcc (3.3v or 5v)
1	0	Asserted	Deasserted	+12v
1	1	Deasserted	Deasserted	zero volts

PC Card Data Transfers

The adapter monitors ISA host bus activity to see if the bus cycle is intended for it or a PC Card installed in one of its sockets. Figure 25-3 shows the signals and functional blocks involved in transferring bus cycles to the target PC Card. Note that figure 25-3 shows a single socket interface to simplify the illustration. In reality, the socket signals shown are duplicated for the second socket.

The adapter must decode the address when an ISA bus cycle is run to determine if either a local access is being made to one of its registers or whether the access is to a PC Card. PC Card accesses are determined via the window address registers. If an ISA access is made to an address location that falls within the address window programmed for a the PC Card, then the adapter knows that the PC Card is being accessed and starts a data transfer either to or from the card depending on the state of the ISA read/write command lines. In essence, the HBA decodes addresses like other ISA adapters. The HBA performs the decode to determine if the transaction is for it (an HBA register) or one of its sockets.

The CL-PD6722 uses a First In First Out serial memory (FIFO) to store up to four write operations. When a write occurs from the host ISA bus, the CL-PD6722 stores the write in the FIFO and completes the operation in zero ISA wait states. The adapter then runs the socket access to the target PC Card to complete the write transfer. In this way, write operations to PC Cards al-

Chapter 25: An Example HBA —The CL-PD6722

ways complete at zero waits states until the FIFO fills up. Note that the FIFO is bypassed on read transfers.

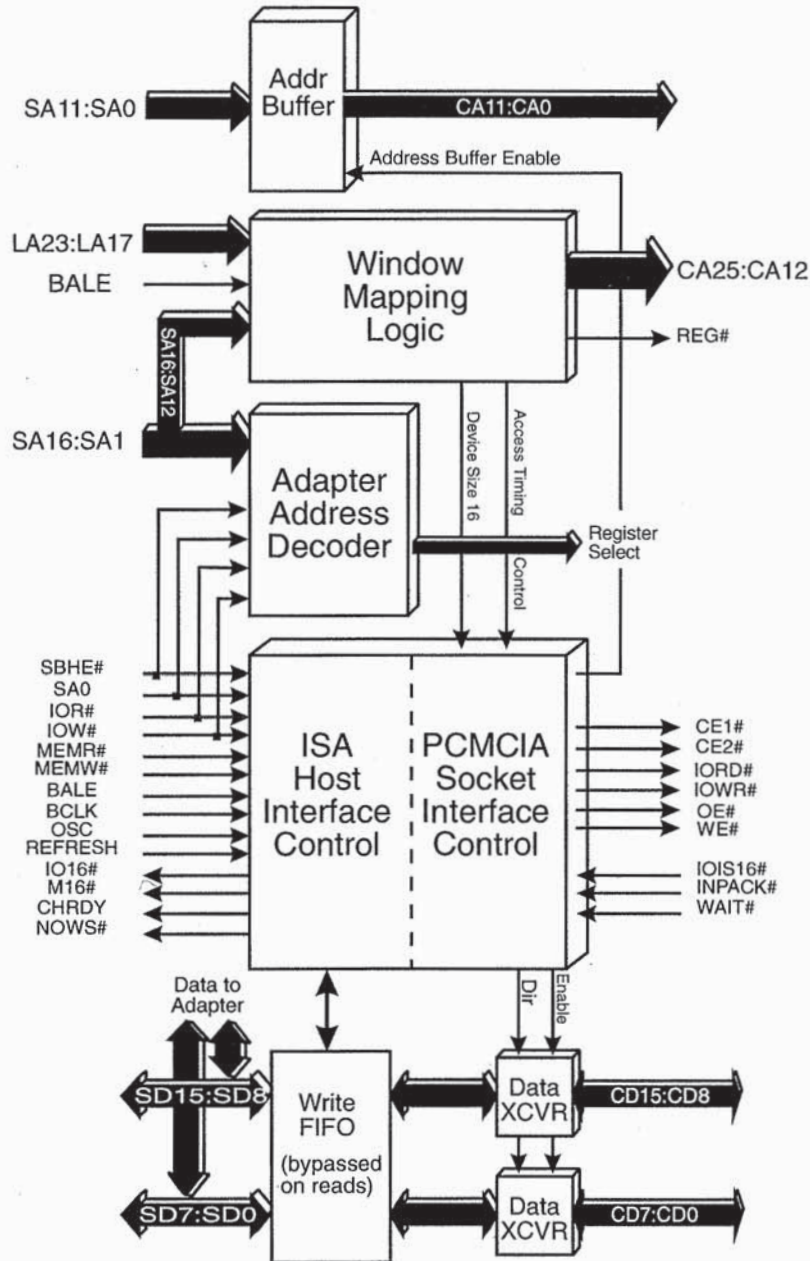


Figure 25-3. Basic Functional Blocks Used During Data Transfers

PCMCIA System Architecture

The CL-PD6722 contains two timing register sets each consisting of three registers that control transfer timing:

- Setup Timing register
- Command Timing register
- Recovery Timing register

These registers provide very flexible transaction timing when accessing PC Cards. Refer to the CL-PD6722 data book for details regarding these registers.

Address Window Mapping

The CL-PD6722 has seven window address registers for each socket: Five memory window registers and two I/O window register. Associated with each register is the transfer speed of the devices that respond within the window.

Memory Interface

The memory window register is comprised of six 8-bit registers containing the following information (refer to figure 25-4):

- **Lower byte of window start address** (LA19:LA17; SA16:SA12). Note that address line 12 is the smallest address used to define a memory address window. This supports the ExCA's requirement that windows start on 4KB boundaries. The lower 12-bits of the address (SA11:SA0) go directly to the socket via a buffer.
- **Upper portion of window start address** (LA23:LA20). The window start address reflects the maximum address capability of the ISA host bus (16MB).
- **Lower byte of window stop (end) address** (LA19:LA17; SA16:SA12). Note that memory windows must also end on even 4KB boundaries, making the smallest memory window 4KB.
- **Upper portion of window stop (end) address** (LA23:LA20)
- **Lower byte of window offset** (CA19:CA12). Note that the offset is comprised of the Card Address value that is added to the ISA address, permitting the card address to appear anywhere within the PC Card's 64MB of address space.
- **Upper portion of window offset** (CA25:CA20).

Chapter 25: An Example HBA —The CL-PD6722

The address register also contains bits that determine characteristics about the specified range of addresses. These characteristics include:

- **Data Size** - Specifies whether access should be made to devices based on 8-bit or 16-bit addressing mode (depends on host bus size).
- **Access Time** (Timer Select) - The CL-PD6722 incorporates two timing register sets that determine the cycle time of the devices that are mapped into the address window.
- **Type of Window** (-REG) - Determines whether the window is used to access attribute memory or common memory.
- **Write Protect** (WP) - specifies whether the memory within the window address range should be write protected. Writes to address within the window are inhibited if WP is set.

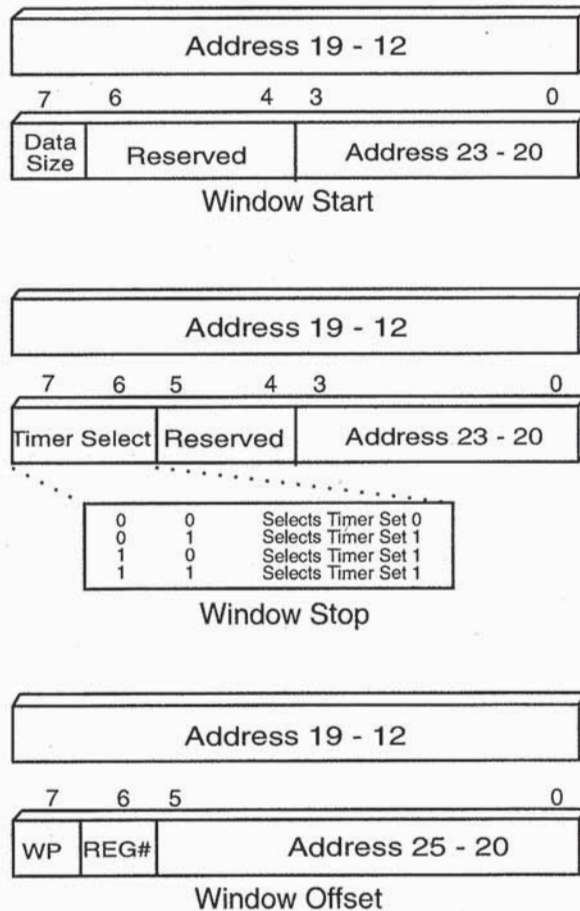


Figure 25-4. Registers Comprising a Single Memory Address Window

PCMCIA System Architecture

I/O Interface

I/O devices are mapped through the I/O window address registers. Each socket contains two I/O window registers each comprised of seven one byte registers as follows (refer to figure 25-5):

- Upper byte of window start address (SA15:SA8)
- Lower byte of window start address (SA7:SA0)
- Upper byte of window stop (end) address (SA15:SA8)
- Lower byte of window stop address (SA7:SA0)
- Upper byte of window offset register (CA15:CA8)
- Lower byte of window offset register (CA7:CA1)
- Control bits for both I/O windows

Note that the I/O start address can begin and end on any byte boundary and can be any length. ExCA specifies constraints regarding I/O address window size and start addresses that compliant software should observe. Since the ISA host bus supports a maximum of 64KB of I/O address space, only 15 address bits are used. The offset capability allows software to map two devices at the same system address space and offset or remap the system addresses to separate locations with the PC Cards I/O address space.

Note that the characteristics of both I/O windows is controlled via the I/O window control register. The characteristics include:

- Data Size (data size and -IOIS16) - An I/O device can be either an 8-bit or 16-bit device. The size can be programmed via the data size bit or can be dynamically determined by the PC Card via the -IOIS16 signal.
- Cycle Timing (Timing Select) - the access timing of the devices responding within the window is determined by the value of a timing register set. The timing select bit determines which timer set should be used.

Chapter 25: An Example HBA —The CL-PD6722

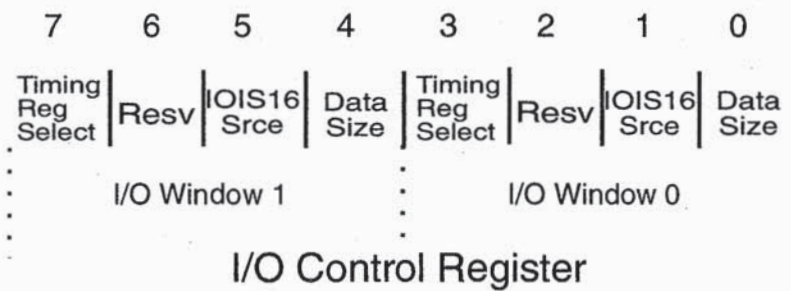
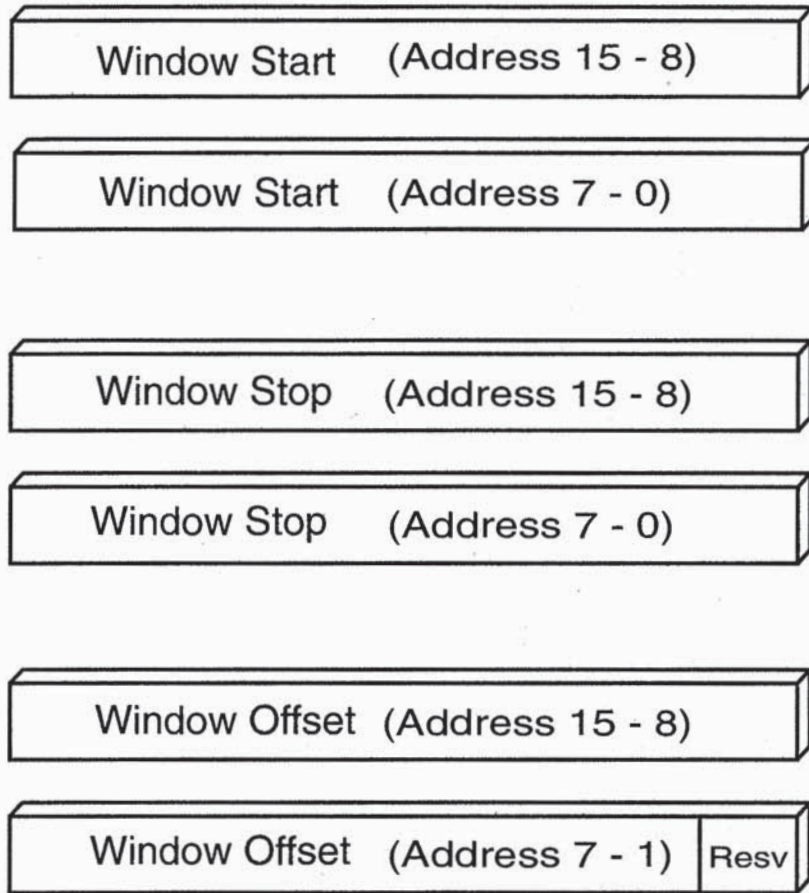


Figure 25-5. Register Comprising a Single I/O Address Window

PCMCIA System Architecture

Status Change Reporting

Status change interrupts are typically reported via a system interrupt whenever a status change event occurs. A single status change interrupt is used to report status changes for all sockets. Status change events that can result in a status change interrupt include:

For Memory Cards:

- Battery Dead Detection
- Battery Low Warning
- Change in Ready/Busy status
- Card Detect Change

For I/O Cards:

- Status Change Pin is asserted - The I/O card's configuration registers must be read to determine which of the previously mentioned status changes have occurred.

The CL-PD6722 reports a status change (also called management) interrupt over the one of the system IRQ lines specified in the Management Interrupt Configuration register (refer to figure 25-6). The upper four bits of the register determine which IRQ line the status change should be reported over, while the lower four bits determine which of the status change events should result in an interrupt being reported. These lower four bits act as a global mask to eliminate one or more of the status change events from being reported by the adapter.

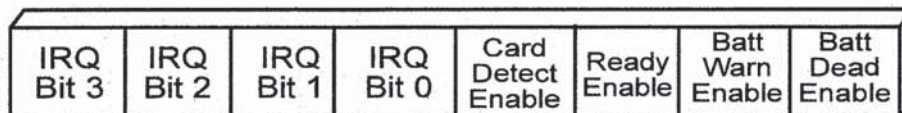


Figure 25-6. Management Interrupt Configuration Register

System software having been notified of a status change must determine which status change caused the interrupt. The Card Status Change register indicates the source of the status change. (Refer to figure 25-7.)

Chapter 25: An Example HBA —The CL-PD6722

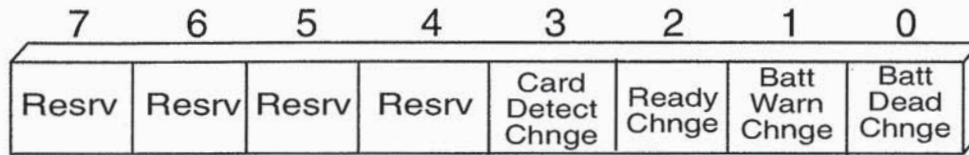


Figure 25-7. Card Status Change Register

The actual state of the socket interface pins can also be observed by software on a socket by socket basis when a memory interface is used. The interface status register provides the capability as shown in figure 25-8. When an I/O interface is defined, the PC Card must be interrogated directly to determine the state of status change indicators.

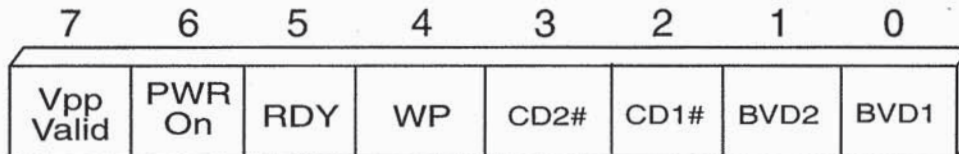


Figure 25-8. Interface Status Register

Interrupt Steering

When a card interrupt occurs, the adapter generates an IRQ to the system. The IRQ line to which the card interrupt is directed is controlled via the interrupt and general control register. (Refer to figure 25-9.) The lower four bits determine which IRQ line the interrupt is steered to. Note that this register is also used to enable management interrupt generation via bit four, and if the card uses interrupts the card type bit (five) will indicate an I/O card type.

Bit six of the register is set and reset to control reset to the PC Card. Bit seven is used when the I/O device is either a FAX, Modem, or network interface card (NIC). This pin is set when the status change pin from the PC Card is used to wake the system up due to external activity that requires system attention. The bit is defined as Ring Indicate (RI) since it is commonly used by FAX or modem cards to notify the system of an incoming call.

PCMCIA System Architecture

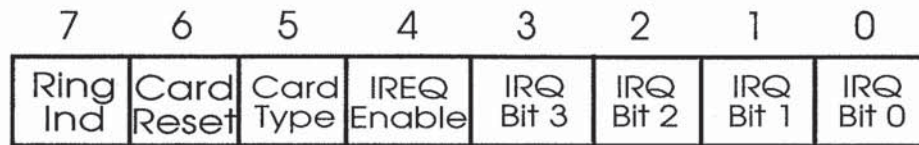


Figure 25-9. Interrupt and General Control Register

Refer to the CL-PD6722 data book for additional details regarding which interrupt pins are supported and how interrupts are reported.

The ATA Socket Interface

Figure 25-10 illustrates the socket interface when configured for ATA. Many of the signals used by the memory and I/O interfaces are no connections (NC) when the socket is configured for ATA. This interface is intended for manufacturers that want to use a PCMCIA socket to support their internal IDE drives. When used as an imbedded connector, the CL-PD6722 can be programmed to operate in the ATA mode, making the socket compatible with the ATA interface. This interface also provides a slight increase in performance when compared to the standard I/O interface approach described in chapter nine.

ATA Registers

The PCMCIA host bus adapter accesses ATA devices using two register groups. The groups are defined as:

- Command Block Registers - used to send commands to the drive, transfer data between the host and drive and return drive status to the host.
- Control Block Registers - used for drive control and returning alternate status information to the host.

The ATA host bus adapter accesses registers within each group by asserting the -CS0 and -CS1 signals. These signals identify which register block is being accessed, while address lines A2, A1 and A0 select the target register within the block. The binary value of A2:A0 should not be thought of as consecutive byte accesses, but rather as a binary code allowing selection of either 8-bit or 16-bit registers. For example, when CS0 is asserted the command register block is selected and address lines A2:A0 determine which of the eight register is being accessed. Register zero is the 16-bit data register selected with a

Chapter 25: An Example HBA —The CL-PD6722

binary code of zero. The next register is the 8-bit error/feature register. (Refer to table 25-2). It is beyond the scope of this book to discuss the definition and use of the ATA registers. Refer to the ANSI ATA specification and the ATA standard within the PCMCIA specification for details regarding register definition and commands.

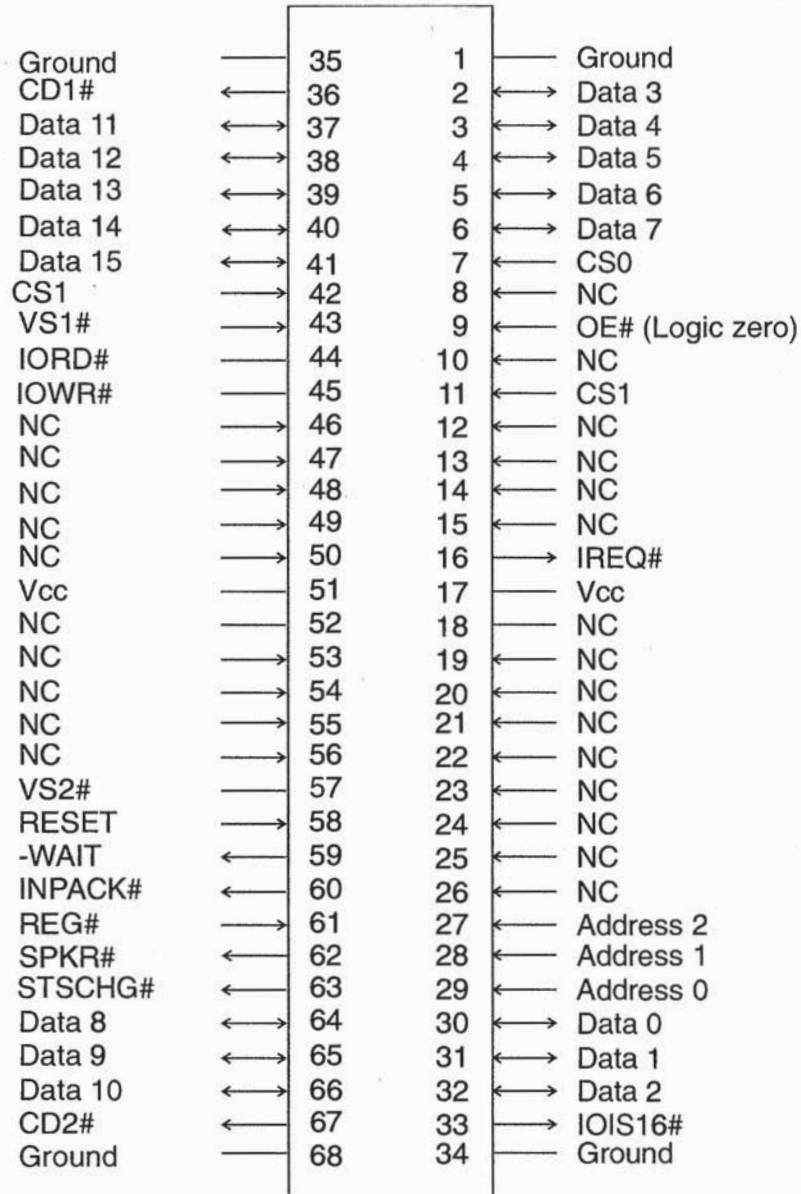


Figure 25-10. ATA Socket Interface

PCMCIA System Architecture

Table 25-2. Example Addressing Scheme Used by ATA Cards

ATA Command Register Block (-CS1 asserted)		
Register # (A2:A0)	Read Function (-IOR asserted)	Write Function (-IOWR asserted)
0	Data Register (16-bit register)	Data Register (16-bit register)
1	Error Register (8 bits)	Features (8 bits)
2	Sector Count (8 bits)	Sector Count (8 bits)
3	Sector Number (8 bits)	Sector Number (8 bits)
4	Cylinder Low (8 bits)	Cylinder Low (8 bits)
5	Cylinder High (4 bits)	Cylinder High (4 bits)
6	Head Number (3 bits)	Head Number (3 bits)
7	Status Information (8 bits)	Command Register (8 bits)

DMA Support

The CL-PD6722 also supports DMA transfers between an I/O Card and memory. This capability is achieved through a special DMA-type PCMCIA interface cycle. This cycle is defined such that conflicts with standard PCMCIA memory or I/O cycles is avoided. These cycles are distinguished from normal I/O cycles by the -REG signal being high during an I/O cycle. This is an undefined condition in the PCMCIA 2.1 specification.

A register within the adapter controls the DMA function. The signal used by the PC Card to request a DMA transfer is programmable. When the controller sees the DMA request from the PC Card, it then requests a DMA transfer from the ISA bus's DMA controller by asserting a DMA request on the ISA bus. The CL-PD6722 uses the IRQ 9 and 10 lines to report a DMA request. If configured for DMA these IRQ lines cannot be used. Refer to the CL-PD6722 data book for details.

Appendices

Appendix A: SRAM CIS Example

The following example is the attribute memory address map showing the CIS tuples implemented within a typical SRAM card. This listing includes page number where the tuple description can be found.

Offset/Adr (hex)	Data (hex)	Description and interpretation	Page Ref
0/0	01	Device Info Tuple	353
1/2	03	Link to next tuple	353
2/4	61	Device type = 6 (SRAM) Device Speed=1 (250ns)	353
3/6	7C	Unit Size=128K Number of Units=16 Total size = 2MB	355
4/8	FF	Termination Byte	353
5/A	15	Level 1 Version/Product Information Tuple	355
6/C	20	Link to next tuple	356
7/E	04	Major Version number=4	356
8/10	01	Minor Version number=0 (PC Card 95 release)	356
9/12	58	X (The remaining entries within the tuple are ASCII codes)	356
A/14	59	Y	356
B/16	5A	Z	356
C/18	00	End Manufacturing ID	356
D/1A	32	2	356
E/1C	4D	M	356
F/1E	42	B	356
10/20	20	.	356

PCMCIA System Architecture

Offset/Adr (hex)	Data (hex)	Description and interpretation	Page Ref
11/22	53	S	356
12/24	52	R	356
13/26	41	A	356
14/28	4D	M	356
15/2A	00	End Card Description Information	356
16/2C	53	S	356
17/2E	52	R	356
18/30	30	0	356
19/32	30	0	356
1A/34	30	0	356
1B/36	30	0	356
1C/38	31	1	356
1D/3A	00	End Model Information 1	356
1E/3C	53	S	356
1F/3E	52	R	356
20/40	30	0	356
21/42	30	0	356
22/44	30	0	356
23/46	30	0	356
24/48	32	2	356
25/4A	00	End Model Information 2	356
26/4C	FF	Termination Byte	356
27/4E	10	Checksum Tuple	356
28/50	05	Link to next tuple	357
29/52	D9	Offset fm Checksum tuple (27h) to checksum start address. D9h (low byte); FFh (high byte)=FFD9h + 0027h (tuple address)=0000h	
2A/54	FF		357
2B/56	27	Number of CIS locations to be checksummed from start address = 27h (low byte);	
2C/58	00	00h (high byte)=0027h	357
2D/5A	DE	Checksum Value=DEh	357
2E/5C	FF	Termination Tuple	357

Appendix A:SRAM CIS Example

Device Information Tuple

Table -1 shows the format of the device information tuple. Shaded areas show portions of the tuple definition used by the SRAM card in the example.

The SRAM CIS listing includes a link value of 03h, indicating only two bytes are used for device info, followed by the termination byte.

Table A -1. Device Information Tuple Format

Byte	Device Information Tuple Format	
0	TPL_CODE	CISTPL_DEVICE (01h)
1	TPL_LINK	Link to next tuple (03h)
		Device Info 1 (2 or more bytes)
		Device Info 2 (2 or more bytes)
...		Device Info n (2 or more bytes)
m		FFh termination byte (marks end of device info field)

The SRAM's device information tuple contains information for a single block of memory, therefore, only one device info block (Device Info 1) is defined. Device info 1 is comprised of two bytes in this example:

- Device Type and Speed Byte
- Device Size Byte

Device Type and Speed Byte

Refer to table -2. The first byte describes the device speed, whether the write protect switch affects this address range, and the device type. Note that the device type code is only used to describe devices that use a fixed memory address range, and not for dynamically relocatable devices. Relocatable devices use the configuration entry tuples to describe the memory address ranges supported.

The device type and speed byte contains a 61h value, equating to the values shown below. Note that extended speed information can be used in lieu of the standard speed definitions. This capability permits speed definitions that might be supported by host bus adapters capable of supporting a wide range of programmable transfer rates.

PCMCIA System Architecture

Table A -2. Memory Device Type and Speed Information

Byte	7	6	5	4	3	2	1	0
0	Device Type Code = 6				WPS=0	Device Speed Codes = 1		
1	Extended Device Speed (if Device Speed Code equals 7h, otherwise omitted)							
2 .. m-1	Additional Extended Device Speed (if bit 7 of Extended Device Speed is 1, otherwise omitted)							
m .. n	Extended Device Type (if Device Type Code equals Eh, otherwise omitted)							

Table A -3. Device Speed Codes

Code	Name	Meaning
0h	DSPEED_NULL	Use when device type = null
1h	DSPEED_250NS	250 nsec
2h	DSPEED_200NS	200 nsec
3h	DSPEED_150NS	150 nsec
4h	DSPEED_100NS	100 nsec
5h-6h		(Reserved)
7h	DSPEED_EXT	Use extended speed byte.

Table A -4. Device Type Codes

Code	Name	Meaning
0	DTYPE_NULL	No device. Generally used to designate a hole in the address space. If used, speed field should be set to 0h
1	DTYPE_ROM	Masked ROM
2	DTYPE_OTPROM	One-time programmable PROM
3	DTYPE_EPROM	UV EPROM
4	DTYPE_EEPROM	EEPROM
5	DTYPE_FLASH	Flash EPROM
6	DTYPE_SRAM	Static RAM (JEIDA has Nonvolatile RAM)
7	DTYPE_DRAM	Dynamic RAM (JEIDA has Volatile RAM)
8-Ch		Reserved
Dh	DTYPE_FUNCSPEC*	Function-specific memory address range. Includes memory-mapped I/O registers, dual-ported memory, communication buffers, etc., not intended to be used as general-purpose memory.
Eh	DTYPE_EXTEND	Extended type follows.
Fh		Reserved

Appendix A:SRAM CIS Example

Device Size Byte

The SRAM's device size byte entry contains 7Ch. This represents a three bit "unit size code" of 4h, and the number of address units value of 0Fh. One is added to the number of address units value to obtain the actual number of units. Refer to table -5 for byte format.

Table A -5. Device Size Definition

7	6	5	4	3	2	1	0
# of address units(Fh) + 1= 10h (16d)					Unit Size Code = 4h		

(16 units x 128KB unit size = 2MB)

Code	Unit Size	Max Size
0	512 bytes	16 K
1	2 K	64 K
2	8 K	256 K
3	32 K	1 M
4	128 K	4 M
5	512 K	16 M
6	2 M	64 M
7	Reserved	Reserved

Level 1 Version / Product Information Tuple

Table -6 shows the format and contents of the Level 1 Version/Product Information tuple. This tuple provides the PCMCIA compliance level supported by the PC Card and includes manufacturer defined product information. The tuple includes three fields:

- The major version byte indicating PCMCIA version information.
- The minor version byte indicating compliance with a given PCMCIA release.
- A variable length field comprised of one or more strings of ASCII characters specified by the manufacturer. A value of 00h demarks each ASCII string.

PCMCIA System Architecture

Table A -6. Level 1 Version/ Product Information Tuple Format

Byte	Level 1 Version/Product Information Tuple Format	
0	TPL_CODE	CISTPL_VERS_1 (15h).
1	TPL_LINK	Link to next tuple (20h).
2	TPLLV1_MAJOR	Major version number (04h).
3	TPLLV1_MINOR	Minor version number (01h) for Release 2.0 and 2.01
4	TPLLV1_INFO	Product information string: name of the manufacturer, terminated by 00h. Additional product information, in text; terminated by 00H. Suggested use: lot number. Additional product information, in text; terminated by 00h. Suggested use: define special programming conditions.
n		FFh: termination byte (marks end of list).

Checksum Tuple

The Checksum tuple is included with this particular SRAM card for additional reliability. In this example, the CIS checksum region is defined as offset 0 (beginning of CIS) and the number of bytes included in the checksum is 27 (byte 0 to 26). Refer to table -7 for the tuple format. Note that this tuple contains three fields:

- Relative start address of the memory block within the CIS to be checked. The relative address is specified as an offset value (contents of this field) added to the offset of the checksum tuple code (the address/2). In this example the beginning of the CIS. ($FFD9h + 0027 = 0000h$)
- Length of the block to be checked. The length is specified as an offset value. The checksum is performed by summing the even bytes in the address range. The last location in the range can be expressed as "target address + 2 * length - 1".
- Checksum value to be tested.

Appendix A:SRAM CIS Example

Table A -7. Checksum Tuple Format

Byte	CheckSum Tuple Format	
0	TPL_CODE	CISTPL_VERS_1 (15h).
1	TPL_LINK	Link to next tuple (05h).
2..3	TPLCKS_ADDR	Offset to region to be checked ,LSB first. (FFD9h)
4..5	TPLCKS_LEN	Length of region to be checked, LSB first. (0027h)
6	TPLCKS_CS	Checksum value of the region

Termination Tuple

The Termination tuple identifies the end of the current tuple list. The termination tuple consists only of the tuple code FFh. This tuple should be the last tuple in a linked list, but does not necessarily indicate the end of the entire string of tuples within the PC Card. Whether processing software stops or continues processing tuples upon encountering the termination tuple depends on the absence or presence of other link-specific tuples in the string as stated below:

- If a no-link tuple is contained in the tuple list, then this is the only tuple list and tuple processing ends.
- If a long-link tuple is contained in the tuple list, then process the secondary tuple list beginning at the address specified by the long-link tuple.
- When processing a secondary tuple list, if no long-link tuple is contained in the tuple list, then no more tuples exist.
- If there is no link-specific tuple contained in the primary CIS tuple list, then tuple processing should continue at location zero in common memory. In other words, a long-link tuple to common memory is implied when there is no link tuple in the primary CIS.

In this example, link-specific tuples were not included, causing parsing software to continue tuple processing at location zero within common memory address space.

Appendix B: Flash Memory CIS Example

The following is an example of a flash memory tuple chain. The reference page number indicates where the tuple description can be found.

Offset/Adr (HEX)	Data (HEX)	Description and Interpretation	Page Ref
0/0	01	Device Info Tuple	361
1/2	03	Link to next tuple	362
2/4	53	Device type = 5 (FLASH); Device Speed = 3 (150ns)	362
3/6	26	Unit Size = 2MB, Number of Units = 5, Total size = 10MB	364
4/8	FF	Tuple Termination Byte	362
5/A	1E	Device Geometry Tuple	364
6/C	06	Link to next tuple	365
7/E	02	Internal bus width of card = 2 bytes (release 1.0 and 2.0 cards)	366
8/10	11	Erase geometry block size $(2^{(n-1)}) = 2^{(11h-1)} = 2^{(16)} = 64K$	366
9/12	01	Read geometry block size $(2^{(n-1)}) = 2^{(1h-1)} = 2^{(0)} = 1$	366
A/14	01	Write geometry block size $(2^{(n-1)}) = 2^{(1h-1)} = 2^{(0)} = 1$	366
B/16	03	Partition size $(2^{(p-1)}) = 2^{(3h-1)} = 2^{(2)} = 4$	366
C/18	01	Interleave size $(2^{(q-1)}) = 2^{(1h-1)} = 2^{(0)} = 1$	366

PCMCIA System Architecture

Offset/Adr (HEX)	Data (HEX)	Description and Interpretation	Page Ref
D/1A	18	JEDEC Identifier tuple	367
E/1C	02	Link to next tuple	367
F/1E	98	Flash Designs JEDEC-ID	
10/20	B9	46F006 JEDEC-ID	
11/22	15	Level 1 Version/Product Information Tuple	367
12/24	26	Link to next tuple	368
13/26	04	Major Version number = 4	368
14/28	01	Minor Version number = 1 (Release 2.0 or 2.1)	368
15/2A	46	F (ASCII string "FLASH DESIGNS")	368
16/2C	76	L	368
17/2E	41	A	368
18/30	53	S	368
19/32	48	H	368
1A/34	20	SPACE	368
1B/36	44	D	368
1C/38	45	E	368
1D/3A	53	S	368
1E/3C	49	I	368
1F/3E	47	G	368
20/40	4E	N	368
21/42	53	S	368
22/44	00	<end manufacturer name>	368
23/46	31	1 (ASCII string "10MB FLASH")	368
24/48	30	0	368
25/4A	4D	M	368
26/4C	42	B	368
27/4E	20	space	368
28/50	46	F	368
29/52	4C	L	368

Appendix B: Flash Memory CIS Example

Offset/Adr (HEX)	Data (HEX)	Description and Interpretation	Page Ref
2A/54	41	A	368
2B/56	53	S	368
2C/58	48	H	368
2D/5A	56	V (ASCII string "VERSION 02")	368
2E/5C	45	E	368
2F/5E	52	R	368
30/60	53	S	368
31/62	49	I	368
32/64	4F	O	368
33/66	4E	N	368
34/68	20	space	368
35/6A	30	0	368
36/6C	32	2	368
37/6E	00	<end version information>	368
38/70	FF	Tuple termination byte	368
39/72	1A	Configuration Tuple	368
3A/74	06	Link to next tuple	369
3B/76	01	Size of fields	369
3C/78	00	Index of last configuration entry within configuration table	370
3D/7A	00	Attribute memory address where configuration registers are mapped (location 4000h)	370
3E/7C	40		
3F/7E	03	Configuration register presence mask (configuration option & status registers)	371
40/80	FF	Tuple termination byte	368
41/82	FF	Termination Tuple (End of tuple string)	371

PCMCIA System Architecture

Device Information Tuple

Table B-1 shows the format of the Device Information tuple. Shaded areas show portions of the tuple definition used by the flash card in the example.

The flash CIS listing includes a link value of 03h, indicating only two bytes are used for device info, followed by the termination byte.

Table B-1. Device Information Tuple Format

Byte	Device Information Tuple Format
0	TPL_CODE CISTPL_DEVICE (01h)
1	TPL_LINK Link to next tuple
	Device Info 1 (2 or more bytes)
	Device Info 2 (2 or more bytes)
...	Device Info n (2 or more bytes)
n	FFh termination byte (marks end of device info tuple)

The Flash's device information tuple contains information for a single block of memory, therefore only one device info block (Device Info 1) is defined. Device info 1 is comprised of two bytes in this example:

- Device Type and Speed Byte (53h)
- Device Size Byte (26h)

Device Type and Speed Byte

Refer to table B-2. The first byte describes the device speed, whether the write protect switch affects this address range, and the device type. Note that the device type code is only used to describe devices that use a fixed memory address range, and not for dynamically relocatable devices. Relocatable devices use the configuration entry tuples to describe the memory address ranges supported.

The device type and speed byte contains a 53h, equating to the values shown in table B-2. Note that extended speed information can be used in lieu of the standard speed definitions. This capability permits speed

Appendix B: Flash Memory CIS Example

definitions that might be supported by host bus adapters capable of supporting a wide range of programmable transfer rates.

The device speed information contained in the tuple is specified as a code. Refer to table B-3

Table B-2. Device Information Entry

Byte	7	6	5	4	3	2	1	0
0	Device Type Code = 5				WPS=0		Device Speed Codes = 3	
1	Extended Device Speed (if Device Speed Code equals 7h, otherwise omitted)							
2 .. m-1	Additional Extended Device Speed (only if bit 7 of Extended Device Speed=1)							
m .. n	Extended Device Type (if Device Type Code equals Eh, otherwise omitted)							

Table B-3. Device Speed Codes

Code	Name	Meaning
0h	DSPEED_NULL	Use when device type = null
1h	DSPEED_250NS	250 nsec
2h	DSPEED_200NS	200 nsec
3h	DSPEED_150NS	150 nsec
4h	DSPEED_100NS	100 nsec
5h-6h		(Reserved)
7h	DSPEED_EXT	Use extended speed byte.

Table B-4. Device Type Codes

Code	Name	Meaning
0	DTYPE_NULL	No device. Generally used to designate a hole in the address space. If used, speed field should be set to 0h.
1	DTYPE_ROM	Masked ROM
2	DTYPE_OTPROM	One-time programmable PROM
3	DTYPE_EPROM	UV EPROM
4	DTYPE_EEPROM	EEPROM
5	DTYPE_FLASH	Flash EPROM
6	DTYPE_SRAM	Static RAM (JEIDA has Nonvolatile RAM)
7	DTYPE_DRAM	Dynamic RAM (JEIDA has Volatile RAM)
8-Ch		Reserved
Dh	DTYPE_FUNCSPEC*	Function-specific memory address range. Includes memory-mapped I/O registers, dual-ported memory, communication buffers, etc., that are not intended to be used as general-purpose memory.
Eh	DTYPE_EXTEND	Extended type follows.
Fh		Reserved

PCMCIA System Architecture

Device Size Byte

The flash's device size byte entry contains 26h. This represents a three bit "unit size code" of 6h, and the number of address units value of 04h. One is added to the number of address units value to obtain the actual number of units. This equates to a unit size of 2MB times 5 unit, or 10MB. Refer to table B-5.

Table B-5. Device Information Size Byte Format

Bits	7	6	5	4	3	2	1	0
Data Value	0	0	1	0	0	1	1	0
Interpretation	# of address units (04h) + 1= 05h (5d)					Unit Size Code = 6h		

(5 units x 2MB unit size = 10MB)

Table B-6. Unit Size Codes

Code	Unit Size	Max Size (32 units)
0	512 bytes	16 K
1	2 K	64 K
2	8 K	256 K
3	32 K	1 M
4	128 K	4 M
5	512 K	16 M
6	2 M	64 M
7	Reserved	Reserved

Device Geometry Tuple

The device geometry tuple provides the erase, read, and write characteristics of the flash device. This tuple consists of multiple entries for each device identified in the device information tuple. Refer to table B-7. In this example, a single device (a 150ns, 10MB flash card) was defined in the device information tuple. Therefore, a single device geometry field is defined within the device geometry tuple (as indicated by the shaded area in the table).

Note that for multiple device cards (i.e. SRAM/Flash card), multiple device information entries are continued within the device information tuple. The device geometry tuple must contain a device information

Appendix B: Flash Memory CIS Example

entry corresponding to each device information entry in the device information tuple. Device geometry entries must exist even if the device geometry information is not relevant (as in the case of SRAM).

Table B-7. Device Geometry Tuple Format

Byte	Device Geometry Tuple Format
0	TPL_CODE CISTPL_DEVICEGEO (1EH)
1	TPL_LINK Link to next tuple (6H)
2 .. 7	Device geometry for first device info entry (6 bytes)
8 .. D	Device geometry for second device info entry (6 bytes)
..	Device geometry for remain device info entries (6 bytes)

Device Geometry Information

The device geometry information consists of six fields that define the characteristics of the memory array or arrays within the memory card. These entries include:

- Internal data bus width within the card.
- Minimum erase block size.
- Minimum read block size.
- Minimum write block size
- Hardware interleaving factor used by the card.

Table B-8 defines how each of these values are expressed in each of the one byte fields. Table B-9 shows the resultant characteristics of the example flash card. Note that the erase, read and write block size must be multiplied by the bus size value and interleave factor to obtain the overall geometric characteristics of the card.

PCMCIA System Architecture

Table B-8. Device Geometry Information Fields Definition

Byte	Device Geometry Fields Definition	
1	DGTPL_BUS	Internal card data bus width. This entry = n, where the bus width is $2^{(n-1)}$ bytes. n=2 for release 1.0 & 2.0 cards.
2	DGTPL_EBS	Minimum Erase Block Size of memory arrays. This entry=n, where the minimum EBS is $2^{(n-1)}$ address increments for bus-width accesses.
3	DGTPL_RBS	Minimum Read Block Size of memory arrays or segments. This entry=n, where the minimum RBS is $2^{(n-1)}$ address increments for bus-width accesses.
4	DGTPL_WBS	Minimum Write Block Size of memory array segments. This entry=n, where the minimum WBS is $2^{(n-1)}$ address increments for bus-width accesses.
5	DGTPL_PART	Minimum size or granularity into which memory array segments can be partitioned. This entry=p, where the minimum partition granularity is $2^{(p-1)}$ erase blocks. P=1 where array partitioning on erase block boundaries is allowed.
6	DGTPL_HWIL	Value = q, where card architectures employ a multiple of $2^{(q-1)}$ times interleaving of the entire memory array or subsystems with the above characteristics. Non-interleaved cards have values of q=1. The value q = 00h is not allowed.

Table B-9. Interpretation of the Device Geometry Information Fields for Sample Flash Card

Device Geometry Field Description	Calculated Value from Field Entry	Data Bus Width	Interleave Factor	Resultant Geometry
Bus Width	2 bytes	NA	NA	NA
Hardware Interleave	1	NA	NA	NA
Erase Geometry	64k	x 2 bytes	x 1	= 128 kb
Read Geometry	1	x 2 bytes	x 1	= 2 bytes
Write Geometry	1	x 2 bytes	x 1	= 2 bytes

Appendix B: Flash Memory CIS Example

JEDEC Identifier Tuple

The JEDEC Identifier tuple is an optional tuple used by programmable devices. This tuple must have a JEDEC identifier entry for each device specified in the device information tuple, whether or not a given device is programmable. In this way, a one-to-one correspondence is maintained between the device information tuples and the JEDEC identifier entries. If a given device is not programmable, then the corresponding JEDEC identifier entry for that device will contain 00h.

The basic structure of the JEDEC Identifier tuple is shown in table B-10. Only the shaded fields are used for the flash card example. Refer to table B-11 for a description of the contents of each JEDEC identifier.

Table B-10. JEDEC Identifier Tuple Format

Byte	JEDEC Identifier Tuple Format
0	TPL_CODE CISTPL_JEDEC_C (18H)
1	TPL_LINK Link to next tuple (2H)
2..7	JEDEC identifier for first device info entry (2 bytes)
8..D	JEDEC identifier for second device info entry (2 bytes)
..	JEDEC identifier for remaining device info entries (2 bytes)

Table B-11. JEDEC Identifier Entry Definition

Byte	7	6	5	4	3	2	1	0
0	parity	Device manufacturer ID assigned by JEDEC (odd parity)						
1	Manufacturer-specific data specifying device type, programming info, etc.							

Level 1 Version / Product Information Tuple

Table B-12 illustrates the format and contents of the level 1 version/product information tuple. This tuple provides the PCMCIA compliance level supported by the PC Card and includes manufacturer defined product information. The tuple includes three data fields:

- The major version byte indicating PCMCIA version information.
- The minor version byte indicating compliance with a given PCMCIA release.

PCMCIA System Architecture

- A variable length field comprised of one or more strings of ASCII characters specified by the manufacturer. A value of 00h demarks each ASCII string. The tuple is terminated by FFh.

Table B-12. Level 1 Version/Product Information Tuple Format

Byte	Level 1 Version/Product Information Tuple Format	
0	TPL_CODE	CISTPL_VERS_1 (15H)
1	TPL_LINK	Link to next tuple
2	TPLLV1_MAJOR	Major version number (04H)
3	TPLLV1_MINOR	Minor version number (01h) for Release 2.0 and 2.01
4	TPLLV1_INFO	Product information string: name of the manufacturer, terminated by 00h. Additional product information, in text; terminated by 00h. Suggested use: lot number. Additional product information, in text; terminated by 00h. Suggested use: programming conditions.
n		FFH: termination byte (marks end of list).

Configuration Tuple

The configuration tuple identifies the number of configuration registers implemented and their location in attribute memory. The configuration tuple consists of six data entries. Table B-13 shows the actual format of the configuration tuple. Note that the entries used in the flash example are shaded.

- Size of fields—specifies the number of bytes in the "configuration registers base address" field, in the "configuration presence mask" field, and in the "reserved field."
- Index number of the last entry in the configuration table.
- Configuration registers base address in attribute memory.
- Configuration presence mask—identifies the configuration registers that are implemented.
- Reserved Field.
- Subtuple information—containing additional card configuration information.

Appendix B: Flash Memory CIS Example

Table B-13. Configuration Tuple Format

Byte	Configuration Tuple Format	
0	TPL_CODE	Configuration tuple code (CISTPL_CONFIG, 1Ah)
1	TPL_LINK	Link to next tuple (n-1; minimum 1)
2	TPCC_SZ	Size of Fields Byte
3	TPCC_LAST	Index Number of the last entry in the Card Configuration Table
4..	TPCC_RADR	Configuration Registers Base Address in attribute memory space. 1,2,3, or 4 bytes depending upon the size field in TPCC_LAST
..	TPCC_RMSK	Configuration Registers Presence Mask. 1 to 16 bytes as indicated by the count in TPCC_SZ.
..	TPCC_RSVD	Reserved area 0 - 3 bytes. Must be 0 bytes until defined.
q+1..r	TPCC_SBTPL	The rest of the tuple is reserved for subtuples containing optional information related to the card's configuration.

Size of fields

The size of fields entry describes the number of bytes used in the TPCC_RADR, TPCC_RMSK and TPCC_RFSZ fields as shown in table B-14. In the flash card example, the size of fields entry has a value of 01h, indicating the following values:

- TPCC_RASZ — a one must be added to the hex value in this field to determine the number of bytes in TPCC_RADR used to specify the configuration registers base address. In this example, the TPCC_RADR entry consists of two bytes.
- TPCC_RMSZ — a one must be added to the hex value in this field to determine the number of bytes in TPCC_RMSK used to indicate which of the option registers have been implemented. In this example, the TPCC_RMSK entry consists of one byte.
- TPCC_RFSZ — the number of bytes reserved for future use (either 0,1,2 or 3). Must be zero for release 2.0 compliance.

PCMCIA System Architecture

Table B-14. Size of Fields Byte

Bits	7	6	5	4	3	2	1	0
Data Value	0	0	0	0	0	0	0	1
Field Definition	TPCC_RFSZ (RESR Size=0)		TPCC_RMSZ (Size of TPCC_RMSK=0)			TPCC_RASZ (Size of TPCC_RADR=1)		

Index Number of Last Configuration Entry

This entry contains the index number of the last configuration entry of the card's configuration table and a reserved field as shown in table B-15. Since no configuration table is used in this example, the "last index" value is zero. Bits six and seven are reserved future use and must be set to zero.

Table B-15. Last Configuration Index

Bits	7	6	5	4	3	2	1	0
Data Value	0	0	0	0	0	0	0	1
Field Definition	Reserved for future use (Resr bits=0)		The index number of the final entry in the Card Configuration Table when scanning the CIS from address zero (Last Index = 0)					

Configuration Registers Base Address Entry

The entry consists of either 1,2,3 or 4 bytes as specified by the "TPCC_RASZ" field of the "Size of Fields" entry. In this example, the "TPCC_RASZ" field indicates this entry consists of two bytes as shown in the shaded area of table B-16. The resulting address is attribute memory location 4000h (or 32,768d).

Table B-16. Configuration Register Base Address Entry

Bits	Configuration Register Base Address Entry
Field	Base Address Bits 7:0 (00H)
Definition	Base Address Bits 15:8 (40H)
	Base Address Bits 23:16
	Base Address Bits 25:24

Appendix B: Flash Memory CIS Example

Configuration Presence Mask

The presence mask entry consists of a variable number of fields as determined by the TPCC_RMSZ field within the size of fields tuple entry. The presence mask is a bit map of configuration registers that can be implemented. The presence mask entry can contain a maximum of sixteen one byte fields (TPCC_RMSZ = 4 bits), and the eight bits in each field represents a configuration register; therefore, 128 configuration registers can be identified. The format of the presence mask fields is shown in figure B-17. In this example, the presence mask entry consists of a single byte (indicated by shading).

Currently, only four registers are specified by the PCMCIA standard. Each of these registers is numbered as follows:

- Register 0 = Configuration Option Register
- Register 1 = Card Configuration and Status Register
- Register 2 = Pin Replacement Register
- Register 3 = Socket and Copy Register

The value 03h specified in the flash card example indicates that the "configuration option register" and "card configuration and status register" have been implemented in this card. Refer to Table B-17.

Table B-17. Configuration Register Presence Mask Entry Format

Bits	Configuration Register Presence Mask Entry Format
Field Definition	Configuration Registers 7:0 (03H)
	Configuration Registers 15:8
	Configuration Registers 23:16
	Configuration Registers 31:24
	Configuration Registers 39:32
	Configuration Registers 47:40
	Configuration Registers 55:48

	Configuration Registers 127:120

PCMCIA System Architecture

Termination Tuple

The termination tuple consists only of the tuple code FFh. This tuple should be the last tuple in the linked list. The action taken when encountering an end of list tuple depends on which form of link tuple, if any, was previously encountered in the tuple:

- If a no-link tuple is contained in the tuple list, then this is the only tuple list.
- If a long-link tuple is contained in the tuple list, then process the secondary tuple list beginning at the address specified by the long-link.
- when processing a secondary tuple list, If no long-link tuple is contained in the tuple list, then no more tuples exist.
- If no link-specific tuples are contained in the primary CIS tuple list, then tuple processing should continue at location zero in common memory. In other words, a long-link tuple to common memory is implied when there is no link-specific tuple in the primary CIS. Tuple processing continues only if the link-target tuple is found at location zero in common memory.

Appendix C: FAX/Modem Tuple Example

Offset/Addr (hex)	Data	Description and interpretation	Page Ref
0/0	01	Device Info Tuple	379
1/2	02	Link = 2h	379
2/4	00	Not a memory device	379
3/6	FF	Termination byte	379
4/8	15	Level 1 Version/Product Information Tuple	379
5/A	24	Link = 24h	380
6/C	04	Major Version number = 4	380
7/E	01	Minor Version number = 1 (Release 2.0 or 2.1)	380
8/10	58	X (The remaining entries within the tuple are ASCII codes)	380
9/12	59	Y	380
A/14	5A	Z	380
B/16	00	<End manufacturers name>	380
C/18	32	2	380
D/1A	2E	.	380
E/1C	34	4	380
F/1E	2F	/	380
10/20	39	9	380
11/22	2E	.	380
12/24	36	6	380
13/26	20		380
14/28	44	D	380

PCMCIA System Architecture

Offset/Addr (hex)	Data	Description and interpretation	Page Ref
15/2A	41	A	380
16/2C	54	T	380
17/2E	41	A	380
18/30	2F	/	380
19/32	46	F	380
1A/34	41	A	380
1B/36	58	X	380
1C/38	20		380
1D/3A	4D	M	380
1E/3C	4F	O	380
1F/3E	44	D	380
20/40	45	E	380
21/42	4D	M	380
22/44	00	<End product name>	380
23/46	30	0	380
24/48	30	0	380
25/4A	31	1	380
26/4C	00	<End lot number>	380
27/4E	41	A	380
28/50	00	<End version information>	380
29/52	FF	Termination byte (End tuple)	380
2A/54	20	Manufacturer Identification Tuple	380
2B/56	04	Link to next tuple	381
2C/58	AA	Manufacturer AAh (low byte); 00 (high byte) = AA00h	381
2D5A	00		
2E/5C	96	Product code = 96h	381
2F/5E	00	Revision Information = 0	381
30/60	21	Function Identification Tuple	381
31/62	02	Link to next tuple	382
32/64	02	Function code 2 = serial interface	382
33/66	00	Initialization byte = no init during POST and no ROM	382
34/68	22	Function extension tuple	383
35/6A	04	Link to next tuple	384
36/6C	00	Tuple function extension type = 0	385

Appendix C: FAX/Modem Tuple Example

Offset/Addr (hex)	Data	Description and interpretation	Page Ref
37/6E	01	UART type = 1 (16450 UART)	385
38/70	0F	UART capabilities = even, odd parity; mark, space, 1 stop and 7 bit characters	385
39/72	1C		
3A/74	22	Modem function extension tuple	385
3B/76	09	Link to next tuple	385
3C/78	05	Tuple function extension type = 5	385
3D/7A	1F	Flow control methods = 31 Trans, RTS/CTS, XON/XOFF	385
3E/7C	3F	DCE command buffer size = 63	385
3F/7E	00	DCE to DTE buffer size = 000300h (768d)	385
40/80	03		
41/82	00		
42/84	00	DTE to DCE buffer size = 000300h (768d)	385
43/86	03		
44/88	00		
45/8A	22		
46/8C	09	Link to next tuple	385
47/8E	06	Tuple function extension type = 6	385
48/90	1F	Flow control methods = 31 Trans, RTS/CTS, XON/XOFF	385
49/92	3F	DCE command buffer size = 63	385
4A/94	00	DCE to DTE buffer size = 000400h (16384d)	385
4B/96	40		
4C/98	00		
4D/9A	00	DTE to DCE buffer size = 000400h (16384d)	385
4E/9C	04		
4F/9E	00		
50/A0	22		
51/A2	0D	Link to next tuple	385
52/A4	02	Tuple function extension type = 2	385
53/A6	01	Max DTE to UART BPS (v 75) = 256	385
54/A8	00		
55/AA	3F	Modulation Standards = 003Fh (V.22BIS, V.22, Bell212A, V23,	

PCMCIA System Architecture

Offset/Addr (hex)	Data	Description and interpretation	Page Ref
56/AC	00	V21 and Bell 103)	385
57/AE	03	Error Correction Detection protocols = 03h (V.42 MNP)	385
58/B0	03	Data compression protocols = 03h (MNP5, V.42BIS)	385
59/B2	1F	Command protocols = 1Fh (V.25BIS, MNP AT, AT 3-1)	385
5A/B4	07	Escape mechanisms = 7	385
5B/B6	00	Data encryption = 0	385
5C/B8	01	Misc. features = 1 (caller ID)	385
5D/BA	B5	Country Code = 181	385
5E/BC	FF	Country Code = 255	385
5F/BE	22	Modem function extension tuple	385
60/C0	08	Link to next tuple	385
61/C2	13	Tuple function extension type = 13	385
62/C4	01	Max. DTE to UARTBPS (v/75) = 256	385
63/C6	00		
64/C8	07	Modulations Standards supported = V.29, V.27ter, V.21	385
65/CA	00	Data encryption = 0	385
66/CC	00	FAX feature selection = 0	385
67/CE	00		
68/D0	B5	Country Code = 181	385
69/D2	22	Modem function extension tuple	385
6A/D4	08	Link to next tuple	385
6B/D6	23	Tuple function extension type	385
6C/D8	01	Max. DTE to UART BPS (v/75) = 256	385
6D/DA	00		
6E/DC	07	Modulation standards supported = v.29, V.27ter, V.21	385
6F/DE	00	Data encryption = 0	385
70/E0	00	FAX feature selection = 0	385
71/E2	00		
72/E4	B5	Country Code = 181	385
73/E6	1A	Configuration Table Tuple	385
74/E8	05	Link to next tuple	386
75/EA	01	2-byte base address register; 1-byte configuration mask	386
76/EC	24	Index number of last configuration table entry = 24h	387

Appendix C: FAX/Modem Tuple Example

Offset/Addr (hex)	Data	Description and interpretation	Page Ref
77/EE	00	base address of configuration registers = 0200h	18
78/F0	02		
79/F2	17	Configuration mask = Config Option, Status and Pin Replacement	388
7A/F4	1B	Configuration Table Entry Tuple	389
7B/F6	11	Link to next tuple	390
7C/F8	E0	Config Index = E0h (Interface byte used, default entry, Index = 20h)	390
7D/FA	41	Interface description byte = Mem/IO interface, READY active, BVD inactive, WP inactive, MWait inactive.	391
7E/FC	9D	Feature selection byte = (Power, timing, I/O, INTR, Misc. defined)	393
7F/FE	78	Power description byte (Istatic, Iavg, Ipeak, Ipwrn)	395
80/100	75	Istatic = 80ma	397
81/102	7D	Iavg = 90ma	397
82/104	06	Ipeak = 100ma	397
83/106	15	Ipwrn = 13ma	397
84/108	E7	Timing description byte (READY scaling factor=10d, no wait timing, and no reserved speed)	398
85/10A	5F	5.0(mantissa) * 10ms (exponent) * 10 (scaling factor) = 500 ms max READY Delay	399
86/10C	AA	I/O description byte (10 address lines, 8-bit device, includes range)	400
87/10E	60	Length size descriptor = 1 address and 1 length	402
88/110	F8	Start address = 03F8h	403
89/112	03		
8A/114	07	Length of address block = 0-7 (8 bytes)	403
8B/116	24	Interrupt descriptor byte = IRQ4, level mode	403
8C/118	28	Misc. description byte = Audio Feedback present	403
8D/11A	1B	Configuration Table Entry	404
8E/11C	08	Link to next tuple	404
8F/11E	21	Configuration Index = 21h, no interface description byte	404
90/110	18	Feature selection byte	404

PCMCIA System Architecture

Offset/Addr (hex)	Data	Description and interpretation	Page Ref
91/122	AA	I/O description byte (10 address lines, 8 bit device, includes range)	404
92/124	60	Length size descriptor = 1 address and 1 length	404
93/126	F8	Start address = 02F8h	404
94/128	02		404
95/12A	07	Length of address block = 0-7 (8 bytes)	404
96/12C	23	Interrupt description byte = IRQ3, level mode	404
97/12E	1B	Configuration Table Entry	404
98/130	08	Link to next tuple	404
99/132	22	Configuration Index = 22h, no interface description byte	404
9A/134	18	Feature selection byte	404
9B/136	AA	I/O description byte (10 address lines, 8 bit device, includes range)	404
9C/138	60	Length size descriptor = 1 address and 1 length	404
9D/13A	E8	Start address = 03E8h	404
9E/13C	03		404
9F/13E	07	Length of address block = 0-7 (8 bytes)	404
A0/140	24	Interrupt description byte = IRQ4, level mode	404
A1/142	1B	Configuration Table Entry	404
A2/144	08	Link to next tuple	404
A3/146	23	Configuration Index = 23h, no interface description byte	404
A4/148	18	Feature selection byte	404
A5/14A	AA	I/O description byte (10 address lines, 8 bit device, includes range)	404
A6/14C	60	Length size descriptor = 1 address and 1 length	404
A7/14E	E8	Start address = 02E8h	404
A8/150	02		404
A9/152	07	Length of address block = 0-7 (8 bytes)	404
AA/154	23	Interrupt description byte = IRQ3, level mode	404
AB/156	1B	Configuration Table Entry	404
AC/158	06	Link to next tuple	404

Appendix C: FAX/Modem Tuple Example

Offset/Addr (hex)	Data	Description and interpretation	Page Ref
AD/15A	24	Configuration Index = 24h, no interface description byte	404
AE/15C	18	Feature selection byte	404
AF/15E	23	I/O description byte (3 address lines, no range)	404
B0/160	30	Interrupt description byte (use IRQ mask)	404
B1/162	BC	Permissible IRQ lines = IRQ2, 3, 4, 5, 7, 9, 10 or 15	
B2/164	86		404
B3/166	14	No Link Tuple	404
B4/168	00	Link to next tuple	404
B5/16A	FF	Termination Tuple (end of tuple list)	404

Device Information Tuple

The device information tuple must be the first tuple of any release 2.0 compliant system and must be located at attribute memory address location zero. Device information provided in this tuple applies only to memory devices. When an I/O only card is used, the device information field will be only one byte long and contain a zero. Table C-1 shows the format of the device information tuple. Shaded area show portions of the tuple definition used by the FAX/modem card in this example. The FAX/modem CIS listing includes a link value of 02h, indicating only one byte for device information, followed by the termination byte.

Table C-1. Device Information Tuple Format

Byte	Device Information Tuple Format	
0	TPL_CODE	CISTPL_DEVICE (01h)
1	TPL_LINK	Link to next tuple (02h)
		Device Info 1 (00h = null - not a memory device)
		Device Info 2 (2 or more bytes)
...		Device Info n (2 or more bytes)
n		FFh termination byte (marks end of device info field)

PCMCIA System Architecture

Level 1 Version / Product Information Tuple

Table C-2 shows the format and contents of the level 1 version/product information tuple. This tuple provides the PCMCIA compliance level supported by the PC Card and includes manufacturer defined product information. The tuple includes three fields:

- The major version byte indicating PCMCIA version information.
- The minor version byte indicating compliance with a given PCMCIA release.
- A variable length field comprised of one or more strings of ASCII characters specified by the manufacturer. A value of 00h demarks each ASCII string.

Table C-2. Level 1 Version/Product Information Tuple Format

Byte	Level 1 Version/Product Information Tuple Format	
0	TPL_CODE	CISTPL_VERS_1 (15h).
1	TPL_LINK	Link to next tuple (24h).
2	TPLLV1_MAJOR	Major version number (04h).
3	TPLLV1_MINOR	Minor version number (01h) for Release 2.0 and 2.01
4	TPLLV1_INFO	Product information string; name of the manufacturer, terminated by 00h. Additional product information, in text; terminated by 00h. In this example: <ul style="list-style-type: none">• Product name• Lot number• Version
n		FFh termination byte (marks end of list).

Manufacturer Identification Tuple

This tuple provides information about the PC Card manufacturer and consists of two fields:

- Manufacturer ID
- Manufacturer specific card ID information

The format of the Manufacturer Identification tuple is shown in Table C-3.

Appendix C: FAX/Modem Tuple Example

Table C-3. Manufacturer Identification Tuple

Byte	Manufacturer Identification Tuple	
0	TPL_CODE	CISTPL_MANFID (20h)
1	TPL_LINK	Link to next tuple (04h)
2..3	TPLMID_MANF	PCMCIA PC Card manufacturer code
4..5	TPLMID_CARD	Manufacturer information (Card Number and/or Revision)

Manufacturer ID Field (TPLMID_MANF)

The value stored in this two-byte field is assigned by PCMCIA with ID codes starting at 0100h and ending at FFFFh. The first 256 codes, 0000h to 00FFh, are reserved for manufacturers that already have an eight-bit JEDEC ID code from the Electronics Industry Association (EIA). This eight-bit ID code may be used as part of the PCMCIA ID. In this case, the JEDEC ID is used as the least-significant eight bits of the PCMCIA code, with the most significant eight bits all zeros.

Manufacturer Card ID Field (TPLMID_CARD)

This two-byte field is designated for manufacturer information regarding the PC Card. The first byte is typically used to identify the card and the second byte for card revision information. The FAX/modem listing defines the first byte as a product code (28h) and the second as revision information (00h).

Function Identification Tuple

This tuple identifies individual functions within the PC Card and specifies whether the function should be automatically configured during system initialization. The tuple contains two fields:

- Function Code byte
- System Initialization Bit Mask

If a PC Card contains multiple functions, this tuple must be repeated for each function. In this case, an initial function identification tuple must be used to specify the PC Card as a multifunction card, followed by a

PCMCIA System Architecture

separate function identification tuple for each function. For a given function, if additional function-specific information is available, function extension tuples will follow the function identification tuple for that function. Refer to table C-4 for the tuple's basic format.

Table C-4. Function Identification Tuple Format

Byte	Function Identification Tuple Format	
0	TPL_CODE	CISTPL_FUNCID (21h)
1	TPL_LINK	Link to next tuple (02h)
2	TPLFID_FUNCTION	PC Card function code (02h)
3	TPLFID_SYSINIT	System initialization bit mask (00h)

Function Code Byte (TPLFID_FUNCTION)

This field contains a code that identifies the basic function of the PC Card. In this example, the FAX/modem is a serial device (code 02h). Table C-5 lists the functions supported by PCMCIA.

Table C-5. PC Card Function Codes

Code	Name	Meaning
0	Multi-Function	PC card has multiple functions. Function identification tuples for each individual function must follow this tuple.
1	Memory	Memory card (RAM, ROM, EPROM, flash, etc.).
2	Serial Port	Serial I/O port (includes modem cards).
3	Parallel Port	Parallel printer port (may be bi-directional).
4	Fixed Disk	Fixed drive (may be silicon or removable).
5	Video Adapter	Video interface extension tuples (type and resolutions).
6	NetworkLAN Adapter	Local Area Network adapter.
7	AIMS	Auto-Incrementing Mass Storage card.
8..FFh	Reserved	Unused in release 2.x. Reserved by PCMCIA for future use.

Appendix C: FAX/Modem Tuple Example

System Initialization Byte (TPLFID_SYSINIT)

This field contains two bits that permits a PC Card to perform initial program load (IPL):

- POST bit — specifies whether a given function should be configured during system initialization
- ROM bit — indicates whether the PC Card contains an expansion ROM

The format of the system initialization byte is shown in table C-6. Note that for the FAX/modem both bits are zero, indicating no requirement for configuration during system initialization and no expansion ROM is included on the card.

Table C-6. Initialization Byte Format

7	6	5	4	3	2	1	0
Reserved for future use, must be set to zero (0)						ROM	POST

Function Extension Tuple

Not all classes of devices have function extension tuples defined. Working groups within PCMCIA that are concerned with specific PC card functions define function extensions. Relevant to the FAX/modem example, the function extensions for the serial port have been defined by PCMCIA. These extensions include support for the serial port itself (UART), data modems, facsimile modems, and voice modems.

The extension tuples for modem support include the features normally seen in application software. The extension tuples provide information for use by application software and play no role in the PC Card's configuration. The types of information included in the extensions include the various features supported by the modem such as: communication protocols, error correction, command support, and data compression support.

Table C-7 shows the common format of the function extension tuples. Each function extension tuple has the same tuple code (22h), a link field and two function-specific fields:

PCMCIA System Architecture

- Function Extension Type Code Field — this field identifies the specific function extension defined by this tuple.
- Function-specific information — this field contains data that is specific to a given extension type.

Table C-7. Function Extension Tuple Format

Byte	Function Extension Tuple Format	
0	TPL_CODE	CISTPL_FUNCE (22H)
1	TPL_LINK	Link to next tuple
2	TPL_TYPE	Function Extension Type Code (see table C-8)
3..n	TPLFE_DATA	Function-specific information

Function Extension Type Code (TPL_TYPE)

A separate function extension tuple is used for each type of extension defined. The particular type of function extension is defined in the function extension type code entry (TPL_TYPE). TPL_TYPE consists of two fields:

- Subfunction ID — this is the function extension type ID code
- Subfunction Descriptor — this identifies the EIA/TIA modem service class

Table C-8 lists the modem extensions and their associated function codes. This example includes serial port, data modem and facsimile modem extensions (those used in the example are shown in the shaded areas). The subfunction descriptor specifies a numeric value related to the EIA/TIA class of service supported by the modem. The FAX/modem in this example specifies a subfunction descriptor for the facsimile modem for class 1 and class 2 support.

Appendix C: FAX/Modem Tuple Example

Table C-8. Modem Function Extensions

7	6	5	4	3	2	1	0
Subfunction Descriptor				Subfunction ID			
Code		Extension Descriptor		Code		Extension Function	
1-15		Describes the EIA/TIA Service Class specified in the numeric value of the code		0		Describes serial port interface. (UART)	
		<ul style="list-style-type: none"> • The extension descriptor is used in conjunction with the FAX modem services function (code 3h). • In addition to the FAX modem services, the descriptor code adds the following: <ul style="list-style-type: none"> 13h = class 1 fax command support 23h = class 2 fax command support 		1		Describes capabilities of the modem interface common to all modem services.	
				2		Describes data modem services.	
				3		Describes facsimile modem services.	
				4		Describes voice encoding services	
				5		Describes capabilities of the data modem interface.	
				6		Describes capabilities of the facsimile modem interface.	
				7		Describes capabilities of the voice modem interface.	
				8		Describes serial port interface for data modem services.	
				9		Describes serial port interface for facsimile modem services.	
				10		Describes serial port interface for voice modem services.	
				11-15		Reserved for future standardization, set to zero.	

Note: The sequence of function extension tuples in the FAX/Modem example is as follows:
 Code 0 - Modem Interface Description. (Default for Data & FAX, 8 & 9 not used)
 Code 5, 6 - Data & FAX modem inter-face capabilities.
 Code 2, 3, 3 - Data Modem Services & FAX Modem Services (class 1 & class 2)

Function-Specific Data (TPLFE_DATA)

Definition of the function-specific data depends on the subfunction ID or mode extension type. The structure of these data fields are detailed in the PCMCIA specification and are not repeated here. A review of the tuple list gives a good idea of the information specified in each type of function extension tuples.

Configuration Tuple

The Configuration tuple identifies the number of configuration registers implemented and their location in attribute memory. The configuration tuple consists of six data entries. Table C-9 shows the actual format of

PCMCIA System Architecture

the configuration tuple. Note that the data entries used in the FAX/Modem example are shaded.

- Size of fields—specifies the number of bytes in the configuration registers base address field, in the configuration presence mask field, and in the reserved field.
- Index number of the last entry in the configuration table.
- Configuration registers base address in attribute memory.
- Configuration presence mask—identifies the configuration registers implemented.
- Reserved Field.
- Subtuple information—containing additional card configuration information.

Table C-9. Configuration Tuple Format

Byte	7	6	5	4	3	2	1	0
0	TPL_CODE		Configuration tuple code (CISTPL_CONFIG, 1Ah)					
1	TPL_LINK		Link to next tuple (05h)					
2	TPCC_SZ		Size of Fields Byte (01h)					
3	TPCC_LAST		Index Number of the last entry in the Card Configuration Table (24h)					
4..	TPCC_RADR		Configuration Registers Base Address in Reg Space. 1,2,3, or 4 bytes depending upon the size field in TPCC_LAST (0200h)					
..	TPCC_RMSK		Configuration Registers Present Mask. 1 to 16 bytes as indicated by the count in TPCC_SZ. (17h)					
..	TPCC_RSVD		Reserved area 0-3 bytes. Must be 0 bytes until defined.					
q+1..r	TPCC_SBTPL		The rest of the tuple is reserved for subtuples containing standardized optional additional information related to the Card Configuration.					

Size of fields

The size of fields entry describes the number of bytes used in the TPCC_RADR, TPCC_RMSK and TPCC_RFSZ fields as shown in table C-10. In the Flash card example, the size of fields entry has a value of 01h, resulting in the following values:

Appendix C: FAX/Modem Tuple Example

- TPCC_RASZ — a one must be added to the hex value in this field to determine the number of bytes in TPCC_RADR used to specify the configuration registers base address. In this example, the TPCC_RADR entry consists of two bytes.
- TPCC_RMSZ — a one must be added to the hex value in this field to determine the number of bytes in TPCC_RMSK used to indicate which of the option registers have been implemented. In this example, the TPCC_RMSK entry consists of one byte.
- TPCC_RFSZ — the number of bytes reserved for future use (either 0,1,2 or 3). Must be zero for release 2.0 compliance.

Table C-10. Size of Fields Byte

Bits	7	6	5	4	3	2	1	0
Data Value	0	0	0	0	0	0	0	1
Field Definition	TPCC_RFSZ (RESR Size=0)		TPCC_RMSZ (Size of TPCC_RMSK=0)			TPCC_RASZ (Size of TPCC_RADR=1)		

Index Number of Last Configuration Entry

This entry contains the index number of the last configuration entry of the card's configuration table and a reserved field as shown in table C-11. The value contained in the "last index" field in this example is 24h. Bits six and seven are reserved future use and must be set to zero.

Table C-11. Last Configuration Index

Bytes/Bits	7	6	5	4	3	2	1	0
Data Value	0	0	1	0	0	1	0	0
Field Definition	Reserved for future use (Resr bits=0)		The index number of the final entry in the Card Configuration Table when scanning the CIS from address zero (Last Index = 24h)					

Configuration Registers Base Address Entry

This entry consists of either 1,2,3 or 4 bytes as specified by the "TPCC_RASZ" field of the "Size of Fields" entry. In this example, the "TPCC_RASZ" field indicates this entry consists of two bytes as shown

PCMCIA System Architecture

in the shaded area of table C-11. The resulting address is attribute memory location 0200h (or 512d).

Table C-12. Configuration Register Base Address

Bytes/Bits	7	6	5	4	3	2	1	0
Field	Base Address Bits 7:0 (00H)							
Definition	Base Address Bits 15:8 (02H)							
	Base Address Bits 23:16							
	Base Address Bits 25:24							

Configuration Presence Mask

The "presence mask" entry consists of a variable number of fields as determined by the TPCC_RMSZ field within the "size of fields" tuple entry. The presence mask is a bit map of configuration registers that can be implemented. The presence mask entry can contain a maximum of sixteen one byte fields (TPCC_RMSZ = 4 bits), and the eight bits in each field represents a configuration register; therefore, 128 configuration registers can be identified. The format of the presence mask fields is shown in table C-13. In this example, the presence mask entry consists of a single byte (indicated by shading).

Currently, only four registers are specified by the PCMCIA standard. Each of these registers is numbered as follows:

- Register 0 = Configuration Option Register
- Register 1 = Card Configuration and Status Register
- Register 2 = Pin Replacement Register
- Register 3 = Socket and Copy Register

The value 17h specified in the FAX/Modem card example means that the configuration option register, card configuration, and status register, pin replacement register and a manufacturer specific register 4 have been implemented in this card.

Appendix C: FAX/Modem Tuple Example

Table C-13. Configuration Register Mask

Bytes/Bits	7	6	5	4	3	2	1	0
Field	Configuration Registers 7:0 (17H)							
Definition	Configuration Registers 15:8							
	Configuration Registers 23:16							
	Configuration Registers 31:24							
	Configuration Registers 39:32							
	Configuration Registers 47:40							
							
	Configuration Registers 127:120							

Configuration Table Entry Tuple

Configuration table entry tuples comprise the configuration table within the CIS. This table provides the configuration options available for the PC Card, with each configuration table entry containing a different combination of options. These configuration table entries are scanned in sequence by PC Card enabling software in an attempt to find a configuration that can be satisfied with available system resources.

Enabling software reads the configuration table entries one at a time to determine the configurable resources that the card requires. After each configuration table entry is read, the enabling software checks available system resources to see if the resources requested are available. If available, enabling software configures the host bus adapter and PC Card. If, however, the configuration cannot be satisfied, enabling software proceeds to the next configuration table entry to obtain alternate configuration options. This process continues until the PC Card's configuration is satisfied. If the configuration cannot be satisfied with available resources, the card cannot be enabled by the enabling software.

Typically, the first configuration entry tuple within the configuration table is specified as the default. This tuple details the desired configuration for the PC Card. Since this tuple is the default, any configuration parameters that are successfully acquired from card services will be retained. Subsequent configuration entries include other permissible configuration combinations.

Refer to table C-14 for the following discussion. The configuration table entry tuple contains up to twelve entries. The number of entries depends on the number of configuration parameters that must be specified

PCMCIA System Architecture

for the additional PC Card options the designer needs to specify. The shaded areas of table C-14 show the entries used by the FAX/Modem in the first configuration table entry tuple. Note that in this example, the link value is 11h. The other entries are detailed below.

Table C-14. Configuration Table Entry Tuple

Byte	7	6	5	4	3	2	1	0
	TFL_CODE		Configuration Entry tuple code (CISTPL_CFTABLE_ENTRY, 18h)					
1	TFL_LINK		Link to next tuple					
2	TFCE_INDX		Configuration Table Index Byte					
-	TFCE_IF		Interface-definition byte (This field is present only when the interface bit of the Configuration-Table Index Byte is set)					
-	TFCE_FS		Feature-selection field indicates which optional fields are present					
-	TFCE_PD		Power-description structure					
-	TFCE_TD		Timing-description structure					
-	TFCE_IO		I/O-description structure					
-	TFCE_IR		Interrupt-request-description structure					
-	TFCE_MS		Memory-space-description structure					
-	TFCE_MI		Miscellaneous-information structure					
..n	TFCE_ST		Additional information about the configuration in subtuple format					

Configuration Table Index Byte

Refer to table C-15. The index byte consists of three fields:

- Configuration Index
- Default bit
- Interface bit

Table C-15. Configuration Index Entry

7	6	5	4	3	2	1	0
Interface	Default	Configuration Entry Number (Index)					

Configuration Index

Each configuration table entry contains a unique index number for identification purposes. The index number of the configuration table entry tuple that satisfies the configuration tells the PC Card which con-

Appendix C: FAX/Modem Tuple Example

figuration options were selected during the configuration process. To enable the configuration described in the tuple, the index number is written to the configuration option register .

Default Bit

This bit specifies whether or not this particular configuration table entry provides default values. In the FAX/Modem example, the first entry is a default entry. If the enabler is able to acquire all configuration parameters specified by a default entry (i.e. obtain the resources required by the card) then the card is configured with the based on the system resources specified within the entry. In the event that some, but not all resources were successfully acquired from the system, the enabler retains those that were granted. The enabler then proceeds to the next entry and attempts to complete the configuration based on alternative parameters specified in the next entry.

When the default bit of the entry is not set, the default conditions are those specified by the last entry encountered that had its default bit set. If one or more of the resources specified by a non-default entry, then the enabler must release all resources specified by the non-default entry. Note that all entries in the FAX/Modem example are non-default entries, except the first entry.

Interface Bit

The interface bit is set in the first entry of this example, specifying that an interface configuration byte follows this byte. If this bit is a zero, then the interface configuration byte is not present within the tuple. (All subsequent entries have the interface bit cleared.) When no interface byte exists, the interface is presumed to be a standard memory only interface, with no requirement to support the wait signal.

Interface Description Byte

This byte describes the type of interface the card requires and specifies some associated features. Refer to table C-16. The fields within the Interface-definition byte are:

PCMCIA System Architecture

- The interface type field
- The card status reporting fields, consisting of:
 - - Battery Voltage Detection Active field
 - - Write Protect Active field
 - - Ready/Busy Active field
- Memory Cycle Wait signal required field

Table C-16. Entry Interface Description Field

Bits	7	6	5	4	3	2	1	0
Value	0	1	0	0	0	0	0	1
Function	M Wait not Re- q'd	RdyBsy Active	WP Inactive	BVDs Inactive	Interface Type			

The Interface Type Field

The four-bit interface type field defines the PC Card's interface type. Notice that the FAX/Modem has an interface type field value of 1h. The 16 possible values and their associated meanings are:

- 0h Memory Only Interface — The status reporting bit fields are not valid for this interface type.
- 1h Memory or I/O Interface — All other bits within this entry are meaningless when this interface is selected.
- 2h-3h Reserved for future standardization.
- 4h-7h Custom Interfaces (0-3) corresponding to the definition of the CCSTPL_CIF subtuples in the Configuration Tuple. The custom interface number is the relative position of the CCSTPL_CIF subtuple used by this configuration in the set of CCSTPL_CIF subtuples within the Configuration Tuple for this card.
- 8h-Fh Reserved for future standardization.

The Card Status Reporting Fields

When an I/O device such as the FAX/Modem is used the status reporting signals, which are part of the memory interface pinout, are not available when the memory or I/O interface is used. These status signals are:

Appendix C: FAX/Modem Tuple Example

- Battery Voltage Detection
- Write-Protect switch position
- Ready/Busy status

Since the FAX/Modem reports ready/busy status, the I/O device must report status via the Pin Replacement Register within the CIS in lieu of the signal pins. The Rdy/Bsy Active bit is set to indicate that the Pin Replacement Register is used to report ready/busy status.

Memory Cycle Wait Signal Required Field

This single bit field specifies that the PC Card requires wait support for the memory device accesses.

Feature-selection field

This byte indicates which additional fields are present within the tuple. The FAX/Modem has a value of 9Dh in the feature selection byte as shown in table C-17. Definition of each of the feature selection byte's fields is detailed in table C-18. Note the definition used by the FAX/Modem is indicated by shading.

Table C-17. Feature Selection Byte

Bits	7	6	5	4	3	2	1	0
Value= 9d	1	0	0	1	1	1	0	1
Option Fields	Misc	Mem Space		IRQ	IO Space	Timing	Power	

PCMCIA System Architecture

Table C-18. Feature Selection Byte Field Definition

Power	The power supply requirements and load characteristics for this configuration are indicated. There may be 0,1,2, or 3 fields following representing Vcc, Vpp, or both Vpp1 and Vpp2 in that order. The coding is as follows:	
	Code	Description
	0	No power-description structures. Use the default
	1	Vcc power-description-structure only.
	2	Vcc and Vpp (Vpp1=Vpp2) power-description-structures.
3	Vcc, Vpp1 and Vpp2 power-description-structures.	
Timing	0	When the default bit is set in this tuple, or no configuration entry tuple has been scanned with its default bit set, then no timing is specified. RDY/BSY may indicate busy indefinitely. WAIT will be active from 0 to 12 microseconds.
	1	A timing description structure is present following the power description structure.
IO Space	0	When the default bit is set in the tuple, or no configuration-entry tuple has been scanned with its default bit set, then no I/O space is used. Otherwise, the I/O space requirement is specified by the most-recently scanned configuration entry tuple with its default bet set.
	1	An I/O description structure is present following the timing-description structure.
IRQ	0	When the default bit is set in this tuple, or no configuration-entry tuple has been scanned with its default bit set, then no Interrupt is used. Otherwise, the Interrupt request requirement is specified by the most recent scanned configuration entry tuple with its default bit set.
	1	An Interrupt request description structure is present following the I/O space description structure.

Appendix C: FAX/Modem Tuple Example

Table C-18. Feature Selection Byte Field Definition (Continued)

MemSpace	Memory address space mapping requirements for this configuration. There may be 0,2,4, or n bytes of information following the interrupt request structure. The coding follows:	
	Code	Description
	0	When the default bit is set in this tuple, or no configuration entry tuple has been scanned with its default bit set, then no configuration dependent, memory address space is used. Otherwise, the memory address space requirement is specified by the most recently scanned configuration entry tuple with its default bit set.
	1	Single 2-byte length specified.
	2	Length (2 bytes) and card address (2 bytes) specified.
	3	A memory space selection byte followed by table memory space descriptors (length determined by selection byte)
Misc	0	When the default bit is set in this tuple, or no Configuration-Entry tuple has been scanned with its default bit set, then the miscellaneous fields are interpreted to be all zero. Otherwise, the miscellaneous fields are specified by the most-recently scanned configuration entry tuple with its default bit set.
	1	A miscellaneous-fields structure is present following the memory space description structure.

Power-Description Structure

The feature selection byte determines which power structure(s) will be defined within the configuration table entry tuple (only a Vcc power description structure in this example). Additionally, each power-description structure defines a variable number of power parameters that will be specified. The power description structure consists of:

- A Power Parameter Selection byte
- Power Parameter Definition byte(s)

PCMCIA System Architecture

Power Parameter Selection byte

The power parameter selection byte specifies which parameters are to be described within the power-description structure. Table C-19 lists the power parameters specified by the FAX/Modem (78h). Note that current parameters but no voltage parameters are defined. Definition of the parameter selection fields is stated in Table C-20.

Table C-19. Power Description Structure Parameter Selection Byte

	7	6	5	4	3	2	1	0
value	0	1	1	1	1	0	0	0
Power Def.	RFU (0)	PDwn I	Peak I	Avg I	Static I	Max V	Min V	Nom V

Table C-20. Power Selection Parameter Definition

Nom V	Nominal operating supply voltage. In the absence of other information the nominal operating voltage has a tolerance of $\pm 5\%$.
Min V	Minimum operating supply voltage.
Max V	Maximum operating supply voltage.
Static I	Continuous supply current required.
Avg I	Maximum current required averaged over 1 second.
Peak I	Maximum current required averaged over 10 milliseconds.
PDwn I	Power-down supply current required.
RFU	Reserved for future standardization.

Power Parameter Definition Bytes

Values for each of the power parameters is determined by codes placed in the mantissa and exponent fields of the power parameter definition byte. Table C-21 shows each of the definition bytes along with the corresponding values for each of the four parameters specified by the power parameter selection byte. The actual values are calculated by multiplying the mantissa and exponent together. The values for the mantissa are given in table C-22 and values for the exponent are given in table C-23.

Appendix C: FAX/Modem Tuple Example

Note that bit seven in each of the parameter definition bytes indicates whether power parameter extension bytes will follow each of the definition bytes. The FAX/Modem card does not implement power extensions. However, for reference purposes table C-24 shows the values and definition provided by the extensions.

Table 3-21. Power Parameter Definition for FAX/Modem

Byte	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	0	1
Istatic	EXT	Mantissa = Eh (8)				Exponent = 5h (10ma)		
2	0	1	1	1	1	1	0	1
Iavg	EXT	Mantissa = Fh (9)				Exponent = 5h (10ma)		
3	0	0	0	0	0	1	1	0
Ipeak	EXT	Mantissa = 0h (1)				Exponent = 6h (100ma)		
4	0	0	0	1	0	1	0	1
Iprdn	EXT	Mantissa = 2h (1.3)				Exponent = 5h (10ma)		

- * The extension bytes may be continued indefinitely until the first byte which contains a 0 or 7, which is the final byte of the extension

Table C-22. Mantissa Values for Power Definition

The Mantissa Values (hex)	
Mantissa	Value
0	1
1*	1.2
2*	1.3
3*	1.5
4	2
5*	2.5
6	3
7*	3.5
8	4
9*	4.5
A	5
B*	5.5
C	6
D	7
E	8
F	9

- * These values are not permitted when the EXT bit is set.

PCMCIA System Architecture

Table C-23. Exponent Values for Power Definition

The Exponent of the Current and Voltage Values are given below:		
Exponent	Current Scale	Voltage Scale
0	100 nA	10 μ V
1	1 μ A	100 μ V
2	10 μ A	1 mV
3	100 μ A	10 mV
4	1 mA	100 mV
5	10 mA	1 V
6	100 mA	10 V
7	1 A	100 V

Table C-24. Power Descriptor Extension Byte

Extension (Bit 7)	Extension Values and Definition (Bits 6:0)	
	0..63h	Binary value for the next two decimal digits to the right of the current value.
	64..7Ch	Reserved
	7Dh	No connection (i.e. high impedance) permitted during sleep or power-down only (Must be last extension byte).
	7Eh	Zero value required (must be only extension byte).
	7Fh	No connection (i.e. high impedance) is required (must be only extension byte).
	Extension bytes may be concatenated indefinitely. The final extension byte contains a 0 in bit 7.	

Timing Description Structure

The Timing Description structure allows the card designer to specify:

- maximum time interval the wait signal will be asserted
- maximum time interval that the card will remain in the busy state
- reserved-time definition

The timing description structure defines up to four bytes used to define the timing parameters. Refer to table C-25. The first byte (byte 0), called the timing scale factor byte, has two purposes: to determine which of the timing parameters are to be defined and if so, what scaling factor is to be applied to the timing descriptor for that parameter. The three bytes that follow the timing scale factor byte are the actual timing descriptors for the three parameters. Only the Ready/Busy timing parameter is implemented by FAX/Modem (the bytes used are shaded).

Appendix C: FAX/Modem Tuple Example

Table C-25. Timing Parameters

Byte	7	6	5	4	3	2	1	0
0	Reserved Scale = 7h (no reserved speed)			Ready/Busy Scale = 1h (scaling factor of 10d)			Wait Scale = 3h (no wait used)	
1	Wait Signal Timing Descriptor							
2	Ready/Busy Timing Descriptor (5Fh)							
3	Reserved-Speed Timing Descriptor							

Timing Scale Factor Byte

Each field within the timing scale factor byte defines how the timing parameter values that follow are to be interpreted. Refer to table C-26.

Table C-26. Timing Scale Factors

WAIT Scale	This field is the power of 10 scaling factor to be applied to the MAX WAIT timing parameter byte which follows. The value 3 indicates that the WAIT signal is not used and the MAX WAIT Speed is not present following this byte.
RdyBsy Scale	This field is the power of 10 scaling factor to be applied to the MAX time the card will be in the Busy State. A value of 7 indicates that Ready/Busy is not used and no maximum time is present.
Res'd Scale	This field is the power 10 scaling factor which is to be applied to a reserved-time definition. A value of 7 indicates that no reserved-speed bytes follows.

Ready/Busy Timing Description Byte

The FAX/Modem has a value of 5Fh in the ready/busy timing description byte. The format of the timing description byte is shown in table C-27. The mantissa and exponent values are speed codes, referring to the values in table C-28. The ready/busy timing parameter is calculated as follows:

$$5.0 \text{ (mantissa)} * 10\text{ms (exponent)} * 10 \text{ (scaling factor)} = 500 \text{ ms}$$

Table C-27. Timing Description Byte

Byte	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	1
Istatic	NA	Mantissa = Bh (5.0d)				Exponent = 7h (10ms)		

PCMCIA System Architecture

Table C-28. Extended Device Speed Codes

Mantissa		Exponent	
Code	Meaning	Code	Meaning
0h	Reserved	0h	1 ns
1h	1.0	1h	10 ns
2h	1.2	2h	100 ns
3h	1.3	3h	1 μ s
4h	1.5	4h	10 μ s
5h	2.0	5h	100 μ s
6h	2.5	6h	1 ms
7h	3.0	7h	10 ms
8h	3.5		
9h	4.0		
Ah	4.5		
Bh	5.0		
Ch	5.5		
Dh	6.0		
Eh	7.0		
Fh	8.0		

I/O-Description Structure

The I/O-description structure consists of the following entries as shown in table C-29:

- I/O Address Decode Requirements byte
- I/O Address Range Descriptor Byte (defines the number of address ranges included within the structure, and characterizes the number of bytes used to define each of the I/O address ranges that follow)
- Address Range Descriptions (up to 16 entries, each defining a range of I/O addresses that the card uses)

Appendix C: FAX/Modem Tuple Example

Table C-29. I/O Description Structure

Byte	7	6	5	4	3	2	1	0
0	I/O Address Decode Requirements Byte							
1	I/O Address Range Descriptor Byte							
2..n	Address Range Description (number of bytes specified by I/O Address Range Descriptor Byte)							
...	Address Range Description (Up to 16 address range descriptions can be defined.)							

I/O Address Decode Requirements Byte

This byte contains four fields as shown in table C-30. The table also provides a definition for each field. The FAX/Modem contains a value of AAh in this field, representing the following:

- Ten I/O Address lines are used by the card's address decoder.
- The card is an 8-bit device.
- One I/O address range (followed by one I/O address descriptor).

I/O Address Range Descriptor Byte

This byte determines the number of address ranges that the card requires and determines the number of bytes used in each address range description that follows. Three fields specify this information:

- Number of I/O Address Ranges field - specifies the number of I/O address blocks that the card requires. For each range specified, an address range description entry follows.
- Address Size field - specifies the number of bytes used in the I/O address descriptions to specify the starting address range.
- Length Size field - specifies the number of bytes used in the I/O address descriptions to specify the length of the address range.

The FAX/Modem contains a value of 60h in this field, telling software responsible for reading the card's I/O address requirements how to interpret the address range descriptions. Refer to table C-31. The number of I/O address ranges is one, so only one I/O address range description follows this byte. Within this I/O address description, the starting address is specified by two bytes, while a single byte specifies the length of the range.

PCMCIA System Architecture

Table C-30. I/O Address Decode Description

Bits	7	6	5	4	3	2	1	0
value (AAh)	1	0	1	0	1	0	1	0
Def.	Range	Bus16	Bus 8	I/O Address Lines = 10				

I/O Address Lines Field (Total number of address lines used by card decoder)	
0	When the I/OAddress Lines field is zero, the card will respond to all addresses presented to it. The system is entirely responsible for when the card is selected, and at what addresses the card is selected. The system must assign to the card a portion of the address space which is at least as large as the number of bytes indicated in the length field of the following range entry. The Base Address for the I/O space (assigned to the card by the system) must begin on a 2*n address boundary such that 2*n is greater than the number of bytes indicated in the length field.
1-26	When IOAddrLines is non-zero, the card performs address decoding to determine when it is selected. In this case, the card and the system share the determination of when a card is actually selected. The card must indicate in IOAddrLines the highest address line (plus 1) which it decodes to determine when it has been selected. The card provides a list of ranges of addresses for which it is selected within the I/O space that it decodes. The system and the card then share the task of determining when the card is selected. The system controls when the CE# pins are asserted during I/O cycles, and the card determines to which addresses it will respond when it is enabled by those CE# signals. The card returns the INPACK# signal to the system whenever the card can recognize the I/O address on the bus.
27-31	Reserved

Bus16	Bus8	Description
0	0	Reserved
0	1	Card registers accessible over data lines D7:D0 only.
1	0	16-bit registers accessible over data lines D15:D0 only (no 8-bit accesses to 16-bit registers are supported) 8-bit card registers accessible to odd bytes may take place over D15:D8 or D7:D0.
1	1	Same as previous combination except 8-bit access are supported to 16-bit registers.

Range	When this bit is a "one", the range of addresses to which the card responds follows this byte, in the I/O address range descriptor byte". If this bit is a "zero", the card responds to all addresses and uses all I/O address lines to distinguish among its I/O ports. In this case, the amount of address space which should be allocated to the card is indicated by the number of address lines decoded by the card (e.g. 4 lines means 16 addresses). No I/O address range descriptor byte follows.
-------	--

Appendix C: FAX/Modem Tuple Example

Table C-31. I/O Address Requirements

Byte	7	6	5	4	3	2	1	0
Value	0	1	1	0	0	0	0	0
1	Length Size (1 byte)		Address Size (2 bytes)		Number of I/O Address Ranges (value +1= 1 address range)			

Address Range Description

The actual I/O addresses specified by the FAX/Modem for the first configuration table entry are 3F8-3FFh as indicated in table C-32.

Table C-32. I/O Address Range Description

Start of I/O Address (3F8h) — This field is 0, 1, 2 or 4 bytes long (2 bytes in this example). Address bits in bytes which are not present are zeros								
Data	7	6	5	4	3	2	1	0
F8h	1	1	1	1	1	0	0	0
03h	0	0	0	0	0	0	1	1
Length of I/O Address (field value + 1) — This field is 1, 2 or 4 bytes long (1 byte in this example). Length bits in bytes which are not present are zeros								
Data	7	6	5	4	3	2	1	0
07h+1	0	0	0	0	0	1	1	1

Interrupt Request Description Structure

The interrupt request description consists of either a single byte or three bytes depending on the value of bit 4 (mask) in the first byte. In the FAX/Modem example, the mask bit is a zero, meaning that the mask registers (bytes 1 & 2) will not be used. Refer to table C-33. The shaded area shows the values used in this example.

Table C-33. Interrupt Request Description Structure

Byte	7	6	5	4	3	2	1	0
0	Share 0	Pulse 0	Level 1	Mask 0	IRQn Line 0-15 (4h)			
1	IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0
2	IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8

PCMCIA System Architecture

Miscellaneous Information Structure

The miscellaneous information structure defines additional features that a given PC Card might support. These features are defined in table C-34.

Table C-34. Miscellaneous Features Field

Byte	7	6	5	4	3	2	1	0
0	EXT	Resrvd (0)	Pwr Down	Read Only	Audio	Max Twin Cards		

Max Twins	This field indicates that cards which support installation of identical cards in the system be differentiated from each other in a sequential manner. For example, first twin is card 0, second is card 1, and so on. This allows the cards to share I/O ports and interrupts in a manner consistent with some peripherals commonly used in PC computers, such as ATA drives. The max-twins field specifies the maximum number of other identical cards which can be configured identically to this card. This permits more than one card to be installed in host which responds to the identical I/O addresses. The host allows the cards to distinguish among themselves by writing their "Copy" numbers (e.g. 0, for the first card, 1 for the second, etc.) into the copy field of the Socket and Copy Register in the Card Configuration Registers.
Audio	This bit indicates that the card allows the BVD2 signal to be used as Audio Waveform for the speaker. This operation is controlled by the Audio Enable Bit in the Card Control and Status Configuration Register.
Read Only	This bit indicates that the card contains a data-storage medium which is read-only for this configuration. There may be other configurations for which the storage medium is read/write.
Pwr Down	This bit indicates that the card supports a power-down mode controlled by the power-down bit in the Control and Status Register.
Resrvd (0)	These bits are reserved for future definition and must be 0.
EXT	An extension follows this byte. A series of extension bytes, which will be defined by PCMCIA, is terminated when an extension byte is encountered which does not have the EXT bit set.

The remaining tuple entries provide options for the I/O range and IRQ lines. If these resources requested in the first configuration table entry are not available, then the next configuration table entry is checked. This continues until a resource combination is satisfied. Note that the last configuration table entry requests any 8-byte range of I/O addresses and a large variety of possible IRQ lines, thereby increasing the chances that the card can be configured.

Appendix D: ATA Disk CIS Example

The following section includes a sample CIS for an ATA PC Card. This sample CIS is from the Maxtor MobileMax 105 MB ATA Drive. Notice that the CIS includes four configuration table entry tuples to support all four addressing modes.

Offset/Addr (hex)	Data (hex)	Description and Interpretation
0/0	01	Device Info Tuple
1/2	04	Link to next tuple
2/4	DF	Device type = D (Function Specific Memory Device) Device Speed=7 (ext.)
3/6	4A	Mantissa = 9 (4.0) Exponent = 2 (100ns) - Speed = 400ns
4/8	01	Unit Size=2KB Number of Units=1 - Total size = 2KB
5/A	FF	Termination Byte
6/C	15	Level 1 Version/Product Information Tuple
7/E	11	Link to next tuple
8/10	04	Major Version number=4
9/12	01	Minor Version number=1 (Release 2.0 or 2.1)
A/14	4D	M (The remaining entries within the tuple are ASCII codes)
B/16	61	a
C/18	78	x
D/1A	74	t
E/1C	6F	o
F/1E	72	r
10/20	00	end of manufacturer name string

PCMCIA System Architecture

Offset/Addr (hex)	Data (hex)	Description and Interpretation
11/22	4D	M
12/24	78	X
13/26	4C	L
14/28	31	1
15/2A	30	0
16/2C	35	5
17/2E	00	00 End Model Information String
18/30	FF	Tuple end byte
19/32	1A	Configuration Tuple
1A/34	05	Link to next tuple
1B/36	01	Configuration register address size = 2 bytes
1C/38	03	Last configuration entry number = 3
1D/3A	00	Configuration Registers base address = 00 (LSB) 02 (MSB)
1E/3C	02	base address = 200h
1F/3E	0F	configuration registers at location 200, 202, 204, and 206
20/40	1B	Configuration Table Entry Tuple
21/42	10	Link to next tuple
22/44	C0	Config entry = 0; interface byte follows; default set
23/46	C0	Interface = memory; wait & rdy/bsy active; bvd & wp inactive
24/48	A5	Feature Selection = Power, Timing, Memory address range, misc entries
25/4A	7F	Power Description = nom v, min v, static i, avg i, peak i, and pwrdsn i
26/4C	55	nom v = 5v
27/4E	4D	min v = 4.5 v
28/50	D5	max v = 5.25v
29/52	19	
2A/54	26	static i = 400ma
2B/56	26	avg i = 400ma
2C/58	6E	peak i = 700ma
2D/5A	54	pwrdsn i = 5ma
2E/5C	FF	no extended wait, no rdy/bsy, or reserved defined
2F/5E	08	2KB memory address range starting at address 0
30/60	00	
31/62	20	Support for powerdown bit in configuration status register
32/64	1B	Configuration Table Entry tuple
33/66	12	Link to next tuple

Appendix D: ATA Disk CIS Example

Offset/Addr (hex)	Data (hex)	Description and Interpretation
34/68	C1	Configuration entry = 1; interface described; default set
35/6A	41	Interface = I/O; rdy/bsy active; wait, wp and bvd not active
36/6C	9D	Feature selection = power, timing, I/O addr range, IRQ and misc. entries
37/6E	7F	Power descrip = nom v, min v, max v, static i, avg i, peak i, & pwrdsn i
38/70	55	nom v = 5v
39/72	4D	min v = 4.5 v
3A/74	D5	max v = 5.25v
3B/76	19	
3C/78	26	static i = 400ma
3D/7A	26	avg i = 400ma
3E/7C	6E	peak i = 700ma
3F/7E	54	pwrdsn i = 5ma
40/80	FF	no extended wait, no rdy/bsy, or reserved defined
41/82	64	I/O addr range = 16 contiguous bytes, 8 or 16-bit mode
42/84	F0	IRQ shared, pulse or level mode supported
43/86	FF	All IRQs supported = IRQ15: IRQ0
44/88	FF	
45/8A	20	Support for powerdown bit in configuration status register
46/8C	1B	Configuration Table Entry tuple
47/8E	0C	Link to next tuple
48/90	82	Configuration entry = 2; interface described; default not set
49/92	41	Interface = I/O; rdy/bsy active; wait, wp and bvd not active
4A/94	18	Feature selection = I/O address range and IRQ entries
4B/96	EA	10 addr lines decoded, 8 or 16 bit mode, subranges follow
4C/98	61	2 address ranges; 2 byte addresses; 1byte length
4D/9A	F0	first address range = F0 (LSB), 01 (MSB) = 01F0h
4E/9C	01	
4F/9E	07	Length of first address range = 8 bytes
50/A0	F6	Second address range = F6 (LSB), 03 (MSB) = 03F6h
51/A2	03	
52/A4	01	Length of second address range = 2 bytes
53/A6	EE	IRQ shared, pulse or level, IRQ 14
54/A8	1B	Configuration Table Entry tuple
55/AA	0C	Link to next tuple
56/AC	83	Configuration entry = 2; interface described; default not set

PCMCIA System Architecture

Offset/Addr (hex)	Data (hex)	Description and Interpretation
57/AE	41	Interface = I/O; rdy/bsy active; wait, wp and bvd not active
58/B0	18	Feature selection = I/O address range and IRQ entries
59/B2	EA	10 addr lines decoded, 8 or 16 bit mode, subranges follow
5A/B4	61	2 address ranges; 2 byte addresses; 1byte length
5B/B6	70	first address range = 70 (LSB), 01 (MSB) = 0170h
5C/B8	01	
5D/BA	07	Length of first address range = 8 bytes
5E/BC	76	Second address range = 76 (LSB), 03 (MSB) = 0376h
5F/BE	03	
60/C0	01	Length of second address range = 2 bytes
61/C2	EE	IRQ shared, pulse or level, IRQ 14
62/C4	21	Function ID Tuple
63/C6	02	Link to next tuple
64/C8	04	Device type 4= Fixed Disk Drive
65/CA	01	Initialization byte = attempt configuration at Power-On Self Test (POST)
66/CC	22	Disk Drive Function Extension Tuple
67/CE	02	Link to next tuple
68/D0	01	Disk device interface
69/D2	01	ATA interface
6A/D4	14	No-link tuple
6B/D6	00	link to next tuple
6C/D8	FF	End of Tuple String

Device Information Tuple

The device information tuple must be the first tuple of any release 2.0 compliant system and must be located at attribute memory address location zero. Device information provided in this tuple applies only to memory devices. Normally when an I/O only card is used, the device information field will be only one byte long and contain a zero. In this case, however, the PC Card supports memory-mapped I/O. Table D-1 shows the format of the device information tuple. Shaded area show portions of the tuple definition used by the ATA card in this example.

Appendix D: ATA Disk CIS Example

The CIS listing includes a link value of 04h, indicating that the extended speed byte is used.

Table D-1. Device Information Tuple Format

Byte	Device Information Tuple Format	
0	TPL_CODE	CISTPL_DEVICE (01H)
1	TPL_LINK	Link to next tuple
		Device Info 1 (00h = no valid device information)
		Device Info 2 (2 or more bytes)
...		Device Info n (2 or more bytes)
n		FFH (marks end of device info field)

The ATA card's Device Information tuple contains information for a single block of memory, therefore, only one device info block (Device Info 1) is defined. Device info 1 is comprised of three bytes in this example:

- Device Type and Speed Byte
- Extended Speed Byte
- Device Size Byte

Device Type and Speed Byte

Refer to table D-2. The first byte describes the device speed, whether the write protect switch affects this address range, and the device type. Note that the device type code is only used to describe devices that use a fixed memory address range, and not for dynamically relocatable devices. Relocatable devices use the configuration entry tuples to describe the memory address ranges supported.

The device type and speed byte contains a DFh value equating to the values shown below. Note that extended speed information is used in lieu of the standard speed definitions (refer to table D-3), since the devices being accessed within the memory address range are registers.

The extended speed byte contains a codes for a mantissa and exponent value. The ATA card's CIS in this example contains a value of 4Ah, equating to a mantissa of 4.0 and an exponent of 100ns or a device speed of 400ns.

PCMCIA System Architecture

Table D-2. Memory Device Type and Speed Information

Byte	7	6	5	4	3	2	1	0
0	Device Type Code = D				WPS=1	Device Speed Codes = 7		
1	Extended Device Speed (if Device Speed Code equals 7h, otherwise omitted)							
2 .. m-1	Additional Extended Device Speed (if bit 7 of Extended Device Speed is 1, otherwise omitted)							
m .. n	Extended Device Type (if Device Type Code equals Eh, otherwise omitted)							

Table D-3. Device Speed Codes

Code	Name	Meaning
0h	DSPEED_NULL	Use when device type = null
1h	DSPEED_250NS	250 nsec
2h	DSPEED_200NS	200 nsec
3h	DSPEED_150NS	150 nsec
4h	DSPEED_100NS	100 nsec
5h-6h		(Reserved)
7h	DSPEED_EXT	Use extended speed byte.

Table D-4. Device Type Codes

Code	Name	Meaning
0	DTYPE_NULL	No device. Generally used to designate a hole in the address space. If used, speed field should be set to 0H
1	DTYPE_ROM	Masked ROM
2	DTYPE_OTPROM	One-time programmable PROM
3	DTYPE_EPROM	UV EPROM
4	DTYPE_EEPROM	EEPROM
5	DTYPE_FLASH	Flash EPROM
6	DTYPE_SRAM	Static RAM (JEIDA has Nonvolatile RAM)
7	DTYPE_DRAM	Dynamic RAM (JEIDA has Volatile RAM)
8-Ch		Reserved
Dh	DTYPE_FUNCSPEC*	Function-specific memory address range. Includes memory-mapped I/O registers, dual-ported memory, communication buffers, etc., which are not intended to be used as general-purpose memory.
Eh	DTYPE_EXTEND	Extended type follows.
Fh		Reserved

Appendix D: ATA Disk CIS Example

Device Size Byte

The ATA device size byte entry contains 01h. This represents a three bit "unit size code" of 4h, and the number of address units value of 0Fh. One is added to the number of address units value to obtain the actual number of units. Refer to table D-5 for byte format.

Table D-5. Device Size Definition

7	6	5	4	3	2	1	0
# of address units(0h) + 1= 1h					Unit Size Code = 1h		

(1 unit x 2KB unit size = 2KB)

Code	Units	Max Size
0	512 bytes	16 K
1	2 K	64 K
2	8 K	256 K
3	32 K	1 M
4	128 K	4 M
5	512 K	16 M
6	2 M	64 M
7	Reserved	Reserved

Level 1 Version / Product Information Tuple

Table D-6 shows the format and contents of the level 1 version/product information tuple. This tuple provides the PCMCIA compliance level supported by the PC Card and includes manufacturer defined product information. The tuple includes three fields:

- The major version byte indicating PCMCIA version information.
- The minor version byte indicating compliance with a given PCMCIA release.
- A variable length field comprised of one or more strings of ASCII characters specified by the manufacturer. A value of 00h demarks each ASCII string.

PCMCIA System Architecture

Table D-6. Level 1 Version/Product Information Tuple Format

Byte	Level 1 Version/Product Information Tuple Format	
0	TPL_CODE	CISTPL_VERS_1 (15h).
1	TPL_LINK	Link to next tuple (24h).
2	TPLLV1_MAJOR	Major version number (04h).
3	TPLLV1_MINOR	Minor version number (01h) for Release 2.0 and 2.01
4	TPLLV1_INFO	Product information string: name of the manufacturer, terminated by 00h. Additional product information, in text; terminated by 00h. In this example: <ul style="list-style-type: none">• Product name• Model Information
n		FFh: marks end of list.

Configuration Tuple

The Configuration tuple identifies the number of configuration registers implemented and their location in attribute memory. The configuration tuple consists of six data entries as follows. Table D-7 shows the actual format of the configuration tuple. Note that the tuples used in the ATA disk example are shaded.

- Size of fields—specifies the number of bytes in the "configuration registers base address" field, in the "configuration presence mask" field, and in the "reserved field"
- Index number of the last entry in the configuration table
- Configuration registers base address in attribute memory
- Configuration presence mask—identifies the configuration registers implemented
- Reserved Field
- Subtuple information—containing additional card configuration information

Appendix D: ATA Disk CIS Example

Table D-7. Configuration Tuple Format

Byte	7	6	5	4	3	2	1	0
0	TPL_CODE		Configuration tuple code (CISTPL_CONFIG, 1AH)					
1	TPL_LINK		Link to next tuple (n-1; minimum 1)					
2	TPCC_SZ		Size of Fields Byte					
3	TPCC_LAST		Index Number of the last entry in the Card Configuration Table					
4..	TPCC_RADR		Configuration Registers Base Address in Reg Space. 1,2,3, or 4 bytes depending upon the size field in TPCC_LAST					
..	TPCC_RMSK		Configuration Registers Present Mask. 1 to 16 bytes as indicated by the count in TPCC_SZ.					
..	TPCC_RSVD		Reserved area 0-3 bytes. Must be 0 bytes until defined.					
q+1..r	TPCC_SBTPL		The rest of the tuple is reserved for subtuples containing standardized optional additional information related to the Card Configuration.					

Size of fields

The size of fields entry describes the number of bytes used in the TPCC_RADR, TPCC_RMSK and TPCC_RFSZ fields as shown in table D-8. In the Flash card example, the size of fields entry has a value of 01h, resulting in the following values:

- TPCC_RASZ — a one must be added to the hex value in this field to determine the number of bytes in TPCC_RADR used to specify the configuration registers base address. In this example, the TPCC_RADR entry consists of two bytes.
- TPCC_RMSZ — a one must be added to the hex value in this field to determine the number of bytes in TPCC_RMSK used to indicate which of the option registers have been implemented. In this example, the TPCC_RMSK entry consists of one byte.
- TPCC_RFSZ — the number of bytes reserved for future use (either 0,1,2 or 3). Must be zero for release 2.0 compliance.

PCMCIA System Architecture

Table D-8. Size of Fields Byte

Bits	7	6	5	4	3	2	1	0
Data Value	0	0	0	0	0	0	0	1
Field Definition	TPCC_RFSZ (RESR Size=0)		TPCC_RMSZ (Size of TPCC_RMSK=0)			TPCC_RASZ (Size of TPCC_RADR=1)		

Index Number of Last Configuration Entry

This entry contains the index number of the last configuration entry of the card's configuration table and a reserved field as shown in table D-9. The value contained in the "last index" field in this example is 03h. Bits six and seven are reserved future use and must be set to zero.

Table D-9. Last Configuration Index

Bytes/Bits	7	6	5	4	3	2	1	0
Data Value	0	0	0	0	0	0	1	1
Field Definition	Reserved for future use (Resr bits=0)		The index number of the final entry in the Card Configuration Table when scanning the CIS from address zero (Last Index = 03d)					

Configuration Registers Base Address Entry

This entry consists of either 1,2,3 or 4 bytes as specified by the "TPCC_RASZ" field of the "Size of Fields" entry. In this example, the "TPCC_RASZ" field indicates this entry consists of two bytes as shown in the shaded area of table D-10. The resulting address is 0200h or attribute memory location 512d.

Table D-10. Configuration Register Base Address

Bytes/Bits	7	6	5	4	3	2	1	0
Field Definition	Base Address Bits 7:0 (00h)							
	Base Address Bits 15:8 (02h)							
	Base Address Bits 23:16							
	Base Address Bits 25:24							

Appendix D: ATA Disk CIS Example

Configuration Presence Mask

The "presence mask" entry consists of a variable number of fields as determined by the TPCC_RMSZ field within the "size of fields" tuple entry. The presence mask is a bit map of configuration registers that can be implemented. The presence mask entry can contain a maximum of eight one byte fields (TPCC_RMSZ = 3 bits), and the eight bits in each field represents a configuration register; therefore, 64 configuration registers can be identified. The format of the presence mask fields is shown in figure D-11. In this example, the presence mask entry consists of a single byte (indicated by shading).

Currently, only four registers are specified by the PCMCIA standard. Each of these registers is numbered as follows:

Register 0 = Configuration Option Register

Register 1 = Card Configuration and Status Register

Register 2 = Pin Replacement Register

Register 3 = Socket and Copy Register

The value 0Fh specified in the ATA CIS means that the "configuration option register", "card configuration and status register", "pin replacement register" and "Socket and Copy register" have been implemented in this card.

Table D-11. Configuration Register Mask

Bytes/Bits	7	6	5	4	3	2	1	0
Field	Configuration Registers 7:0 (0Fh)							
Definition	Configuration Registers 15:8							
	Configuration Registers 23:16							
	Configuration Registers 31:24							
	Configuration Registers 39:32							
	Configuration Registers 47:40							
	Configuration Registers 55:48							
	Configuration Registers 63:56							

PCMCIA System Architecture

Configuration Table

The configuration table contains four entries each of which describes a different combination of resource options required by the ATA card.

Function Identification Tuple

This tuple identifies a PC card's function and specifies whether the function should be automatically configured during system initialization. The tuple contains two fields:

- Function Code byte.
- System Initialization Bit Mask.

Refer to table D-12 for the tuple's basic format.

Table D-12. Function Identification Tuple Format

Byte	Function Identification Tuple Format	
0	TPL_CODE	CISTPL_FUNCID (21H)
1	TPL_LINK	Link to next tuple (at least 2)
2	TPLFID_FUNCTION	PC Card function code
3	TPLFID_SYSINIT	System initialization bit mask

Function Code Byte (TPLFID_FUNCTION)

This field contains a code that identifies the basic function of the PC card. In this example, the ATA card is a Fixed Disk (code 04h). Table D-13 lists the functions supported by PCMCIA.

Appendix D: ATA Disk CIS Example

Table D-13. PC Card Function Codes

Code	Name	Meaning
0h	Multi-Function	PC Card has multiple functions. Examine the following function identification tuples that follow for individual functions.
1h	Memory	Memory Card (RAM, ROM, EPROM, flash, etc.).
2h	Serial Port	Serial I/O port, includes modem cards.
3h	Parallel Port	Parallel printer port, may be bi-directional.
4h	Fixed Disk	Fixed drive, may be silicon may be removable.
5h	Video Adapter	Video interface, extension tuples (type and resolutions).
6h	Network LAN Adapter	Local Area Network adapter.
7h	AIMS	Auto-Incrementing Mass Storage card.
8..FFh	Reserved	Unused in this release. Reserved by PCMCIA for future use.

System Initialization Byte (TPLFID_SYSINIT)

This field contains two bits that permit a PC card to perform initial program load (IPL):

- POST bit — specifies whether a given function should be configured during system initialization.
- ROM bit — indicates whether the PC card contains an expansion ROM.

The format of the System Initialization byte is shown in table D-14. Note that for an ATA card, the POST bit is usually set to one, indicating the card should be configured during system initialization. A designer may or may not choose to incorporate an expansion ROM on the card.

Table D-14. Initialization Byte

7	6	5	4	3	2	1	0
Reserved for future use, must be set to zero (0)						ROM	POST

PCMCIA System Architecture

Function Extension Tuple

Two extension tuples have been defined for ATA drives. One identifies the interface type and another specifies additional PC Card ATA features. Table D-15 shows the common format of the Function Extension tuples. Each Function Extension tuple has the same tuple code (22h), a link field and two function-specific fields:

- Function Extension Type Code field — this field identifies the specific function extension defined by this tuple.
- Function-specific information — this field contains data that is specific to a given extension type.

Table D-15. Function Extension Tuple Format

Byte	Function Extension Tuple Format	
0	TPL_CODE	CISTPL_FUNCE (22H)
1	TPL_LINK	Link to next tuple
2	TPL_TYPE	Function Extension Type Code
3..n	TPLFE_DATA	Function-specific information

Appendix E: Metaformat Layers 2, 3, and 4

The following table lists and describes the tuples defined for the Layer 2: Data Recording Format, of the PC Card Metaformat.

Code	Name	Description
23h	CISTPL_SWIL	Software interleaving — This tuple allows software interleaving of data within a partition on the card. This tuple indicates the software interleaving factor.
40h	CISTPL_VERS_2	Level-2 version tuple — This tuple indicates the compliance of the level 2 tuples within the card and provides information regarding the general organization of the PC Card.
41h	CISTPL_FORMAT	Data recording format for Common Memory — This tuple provides information about how the card is organized and accessed for use as a virtual disk drive. This tuple includes information defining: <ul style="list-style-type: none">• Whether access is memory-like (byte accessible) or disk-like (accessed in blocks of address space).• The error correction method employed and length.• Byte address of first data byte in this partition• Number of data bytes in this partition.
42h	CISTPL_GEOMETRY	Partition geometry — This tuple is for use by cards that have disk-like partitions. Provides instructions to the file management system that requires data be

PCMCIA System Architecture

		located based on cylinders, tracks, and sectors.
43h	CISTPL_BYTEORDER	Byte ordering for disk-like partitions — This tuple is intended for PC Cards that have a memory-like organization. That is, cards that can be read from and written to one byte at a time. This tuple specifies the order for multi-byte data, and the order in which bytes map into words (even two byte block) for 16-bit cards.
44h	CISTPL_DATE	Card Initialization date — This tuple indicates the date and time that the PC Card was last formatted.
45h	CISTPL_BATTERY	Battery replacement date — This tuple is intended for PC Cards having battery-backed storage. It indicates the date of the last battery replacement, and the date that replacement is likely to be required again.
47h	CISTPL_FORMAT_A	Data recording format for Attribute Memory — This tuple provides information about how the card is organized and accessed for use as a virtual disk drive. This tuple includes information defining: <ul style="list-style-type: none"> • Whether access is memory-like (byte accessible) or disk-like (accessed in blocks of address space). • The error correction method employed and length. • Byte address of first data byte in this partition • Number of data bytes in this partition.

Appendix E: Metaformat Layers 2, 3, and 4

The following table describes the tuple defined for metaformat Layer 3: Data Organization Tuples.

Code	Name	Description
46h	CISTPL_ORG	<p>Partition organization — This tuple contains information about the organization of a partition within a PC Card. The tuple describes whether:</p> <ul style="list-style-type: none"> • the partition contains a file system and specifies type and version. • the partition contains applications-specific information and specifies name and version. • the partition contains executable code images and specifies name and version of the organization scheme. • the partition uses a vendor-specific organization.

The table below describes the tuple defined for metaformat Layer 4: System-Specific Standard Tuples.

Code	Name	Description
90h	CISTPL_SPCL	<p>Special Purpose — These tuple has meaning for DOS file systems and are used to define an interchange format for cards formatted with the DOS-FAT-based file system.</p> <p>Also provides a standard for executing code directly from a PC Card, called execute-in-place or XIP.</p>
80h - 8Fh	Vendor unique tuples	

Appendix F: References

Additional references on PCMCIA:

Dipert, Brian and Levy, Markus *Designing with Flash Memory*, Annabooks, 1993.

Mori, Michael T. And Welder, W. Dean, *The PCMCIA Developer's Guide*, Second Edition, Sycard Technology, 1994-95.

PCMCIA/JEIDA, *PC Card Standard*, Volumes 1-12, February 1995
2635 North First Street, Suite 209, San Jose, CA 95131, USA, phone: (408)
433-2273.

Glossary

Access speed. The time required for a given memory or I/O device to accept or supply data when it is selected by a given bus master.

Address offset. The value (typically measured in bytes) that specifies an address location relative to a given base (or, start) address.

Address translation. The process of converting one form of address to another. Typically performed by a bridge device when passing a transactions between buses that implement different addressing protocols.

Address window speed. The cycle time associated with accessing a memory or I/O device. PCMCIA specifies the same cycle time for all devices that are mapped within the same host bus adapter address window.

Advanced client services. A category of services within card services that perform advanced functions not typically used by standard client drivers (or, enablers).

AIMS interface. Auto-Indexing Mass Storage (or, AIMS) is an extension to the PCMCIA specification that provides a simple PC Card interface used typically for storing large images.

AT. An acronym for advanced technology used by IBM when naming their 80286-based PCs (i.e. IBM PC-AT). Note that AT and ISA (Industry Standard Architecture) are commonly used terms to designate compatibility with the IBM PC-AT.

ATA. An acronym for AT attachment. An ANSI standard that defines a disk drive interface between an AT compatible bus and IDE (integrated drive electronics) disk drives.

ATA PC Card. A PC Card that conforms to the ATA standard (in most respects), providing a standard programming interface which is supported by virtually all of today's PCs.

ATA flash card. An PC Card that employs flash memory and uses an ATA interface to emulate a disk drive.

Attribute memory. PC Card address space used to store configuration information. The card's CIS and configuration registers are mapped into attribute memory address space. Only even address locations within attribute memory contain valid information.

PCMCIA System Architecture

Battery warning. Memory cards used to emulate disk drives must provide a battery to retain information, if they use volatile memory. These cards have two signals pins (BVD1 and BVD2 – battery voltage detect 1 and 2) that can be used to report a low battery warning or a dead battery indication to the system.

Bulk memory services. A category of card services used by memory client drivers to gain access and manage block memory transfers.

BVD1 and BVD2. Memory cards used to emulate disk drives must provide a battery to retain information, if they use volatile memory. These cards have two signals pins (BVD1 and BVD2 – battery voltage detect 1 and 2) that can be used to report a low battery warning or a dead battery indication to the system.

Call-back. The process used by card services to notify clients of events that they should respond to. The call-back calls a routine within the client driver used to handle the event notification.

Card detection. The process of recognizing that a PC Card has been inserted into a socket and notifying enabling software responsible for configuring the card.

Card information structure. Also called the CIS, this data structure is incorporated into PC Cards to characterize the function(s) contained within the card. The CIS consists of individual elements called tuples, each of which describes a given characteristic of the card. The CIS is typically mapped into attribute memory address space, but is sometimes implemented in common memory address space.

Card services. A collection of software functions based on the client server model that permits unified control to all PC Card sockets and related hardware. PC Card client drivers register with card services to obtain access to PC Cards and sockets.

Resource Management Services. A category of card services that used by a client driver to acquire system resources needed for configuring their card. When a client driver requests resource for their card, card services checks to verify that the requested resource is not being used by some other device within the system. Card services performs look-ups within its resource management table that contains the resources that are available for allocation to PC Cards.

Card Types. PCMCIA defines three card types (types 1, 2, and 3). These cards have the same electrical interface and differ only in height.

CD1# and CD2#. The Card Detect (CD) pins signal that a card is fully inserted into a PC Card socket. When the HBA detects these pins asserted it notifies card services via an interrupt.

CE1# and CE2#. The card enable (CE) pins specify that a PC Card is being accessed and whether one or two bytes are being requested.

CIS. See Card information structure.

Client driver. A client driver, also called an enabler, is responsible for configuring and enabling a PC Card when it is first inserted into a card socket. The client registers with and makes calls to card services provides the services necessary to fulfill the clients requests.

Client utility services. A category of card services used by client drivers to request that card services perform complex tasks that would otherwise require many low-level requests be made by the client driver.

Common memory. Address space within a PC Card used as the working memory, where files are typically stored.

Configuration process. The process involving reading and interpreting a PC Cards CIS entries to determine the type of function implemented by the PC Card and it configuration requirements.

Configuration registers. Configuration registers within a PC Card provide the ability to program it for a given configuration and obtain status information about the card.

DACK. DMA acknowledge (or, DACK) is a PC Card input for PC Card I/O functions that support DMA transfers. This signal is returned to the I/O card from the host system's DMA controller in response to a DMA request by the card. DACK signals the beginning of the DMA transfer between memory and the I/O function.

Dedicated enabler. A PC Card enabler that is designed to configure and enable a specific PC Card. Dedicated enablers are usually supplied by the PC Card manufacturer. Also called Device-specific enabler.

Device-specific enabler. See Dedicated enabler

Digital Audio Waveform. The audio information output over a PC Card's SPEAKER# (SPKR#) pin to the host system's, used to drive the host system's speaker.

DMA. Direct memory access (or, DMA) in the PC environment is the a transfer between the host system's main memory and an I/O device. The transfer is controlled by the host-resident DMA controller.

DMA Acknowledge. DMA acknowledge (or, DACK) is a PC Card input for PC Card I/O functions that support DMA transfers. This signal is returned to the I/O card from

PCMCIA System Architecture

the host system's DMA controller in response to a DMA request by the card. DACK signals the beginning of the DMA transfer between memory and the I/O function.

DMA Bus Cycle. A bus cycle performed by the host system DMA controller

DMA clock. The clock used to run the Host DMA controller. 4MHz in a PC Compatible system.

DMA Request. DMA request (or, DREQ#) is a PC Card signal used by I/O cards that support DMA. The I/O card asserts DREQ# to notify the host DMA controller that it is ready to transfer data between itself and memory.

DREQ#. See DMA Request.

Dual-voltage cards. PC Cards that can operation at either 5vdc or 3.3vdc.

EMS. Expanded Memory Specification (or, EMS) defines a memory management procedure that allows additional memory to be added to a DOS-based PC that typically supports only 1MB of usable address space. EMS was defined by Lotus, Intel, and Microsoft and is also referred to as the (LIM specification). Only applications written to support EMS can access the additional system memory.

Enabler. The software responsible for detecting, configuring, and enabling a PC Card. Enablers are also called client drivers, because they interface to the PC Card environment via card services. Compare Point Enabler.

Event call-back. The mechanism used by card services to notify its clients (PC Card enablers) of specific events that have occurred at the PC Card and socket. The enablers are responsible for processing the events.

Event notification. Another term for a card services call-back. See Event call-back.

Event wakeup. An event that is external to the PC that stimulates a PC Card to perform some type of action (e.g. a remote call to a modem), when the system is in a power conservation state. The external event is used to "wake" the system so that it can respond.

ExCA. Exchangeable Card Architecture (or, ExCA) is a specification defined by Intel to promote interoperability of PC Cards between x86-based PCs. Note that ExCA has been renamed QuickSwap.

Execute-in-place. Execute-in-place (or, XIP) refers to the ability of a PC memory card that emulates a disk drive to execute code directly from memory, rather than

having to copy and execute the file from host memory. Only applications written to support XIP can execute directly from PC Card memory.

EXIP. A type of XIP, called Extended XIP, that requires a 386 or later x86 compatible processor. PC Card memory in this case is mapped into host system address space above 1MB.

Expanded Memory Specification. Expanded Memory Specification (or, EMS) defines a memory management procedure that allows additional memory to be added to a DOS-based PC that typically supports only 1MB of usable address space. EMS was defined by Lotus, Intel, and Microsoft and is also referred to as the (LIM specification). Only applications written to support EMS can access the additional system memory.

FFS. See Flash file system.

Flash file system. A file system designed to manage access to PC Card flash memory that emulates a disk drive. A specific file system is required for flash memory to support the special write characteristics of flash memory. Also called FFS.

Flash translation layer. A form of flash file system that interfaces to the DOS file system, rather than implementing a specific installable file system that replaces DOS when accessing virtual flash drives. Also called FTL.

FTL. See Flash translation layer.

Generic enabler. A PC Card enabler designed to recognize and configure a wide variety of card types. Compare Dedicated enabler.

HBA. Host Bus Adapter (or, HBA) is the hardware interface between the host expansion bus and PC Card sockets. The HBA bridges, or translates transactions between the PC Card sockets and the expansion bus.

HLDA. Hold Acknowledge (or, HLDA) is an output from an x86 processor and an input to the DMA controller. HLDA is asserted by the processor when it detects its HOLD signal has been asserted by the DMA controller in response to DREQ# being asserted by the PC Card I/O device.

HOLD. Hold request (or, HOLD) is an input to an x86 processor that directs it to relinquish control of the system bus. This signal is asserted by the host DMA controller to request use of the system buses so that it can perform a DMA transfer. See also HLDA.

PCMCIA System Architecture

Host bus adapter. Host Bus Adapter (or, HBA) is the hardware interface between the host expansion bus and PC Card sockets. The HBA bridges, or translates transactions between the PC Card sockets and the expansion bus.

Hot insertion. The term used to describe the ability of PC Cards to be inserted and removed from the system after it has already been powered on.

I/O address window. A range of I/O address location programmed into the HBA that corresponds to the address locations used by an I/O card to access its internal registers. The HBA recognizes when software attempts an access to a location within the specified range and forwards the transaction to the target I/O card and socket.

I/O Read Command. A command that is asserted by the host system to indicate that an I/O read operation is being performed. A card socket signal (IORD#) is asserted when an I/O read targets a PC Card register.

I/O size is 16-bits. A socket interface signal (IOIS16#) output from an I/O card, telling the HBA the size of the register being accessed (either 8- or 16-bits).

I/O Status Change. A socket interface signal (STSCHG#) asserted by an I/O card to notify the HBA and the enabler that a status change has occurred within the PC Card

I/O Write Command. A command that is asserted by the host system to indicate that an I/O write operation is being performed. A PC Card signal (IOWR#) is asserted by the HBA when it recognizes that an I/O write operation is targeting a PC Card register.

IDE. Integrated drive electronics (or, IDE) is a type of hard drive that incorporates most of the hard drive controller logic within the drive itself. The interface between an ISA compatible bus and the IDE drive is called the ATA interface.

INPACK#. See Input port acknowledge.

Input port acknowledge. Input port acknowledge (or, INPACK#) is an output signal from an I/O card during an I/O read that accesses a PC Card register. This signal notifies the HBA that the access belongs to the PC Card.

Interrupt request. A request to the system that indicates that the I/O card needs servicing. The interrupt request ultimately calls the PC Card's interrupt service routine (or, ISR). The interrupt request is signaled via the card's IREQ# pin.

IOIS16#. I/O is 16 bits (or, IOIS16#) is a socket interface signal output from an I/O card, telling the HBA the size of the register being accessed (either 8- or 16-bits).

IOR#. The I/O read command (or, IORD#) is a signal asserted by the host system to indicate that an I/O read operation is being performed from the I/O card.

IOWR#. A command that is asserted by the host system to indicate that an I/O write operation is being performed. The PC Card signal (IOWR#) is asserted by the HBA when it recognizes that an I/O write operation is targeting a PC Card register.

IPL. Initial Program Load refers to the process of loading the operating system during the power-up sequence. Sometimes also referred to as the boot process.

IREQ# Also called IREQ#, this PC Card

JEDEC. Acronym for Joint Electronics Device Engineering Council.

JEIDA. Acronym for Japanese Electronics Industry Development Association.

Level mode interrupts. A method of signaling interrupts to the host system. A PC Card using level mode interrupts causes an interrupt to be registered, or triggered, by asserting the IREQ# pin and keeping it low until the interrupt is cleared by the ISR.

LIM 4.0. See Expanded Memory Specification.

Low voltage socket. A PC Card socket that can apply either 5vdc or 3.3vdc as the initial Vcc power to the socket. The HBA that supports a low voltage socket monitors the voltage sense pins (VS1# and VS2#) to detect the initial voltage required by the PC Card.

LXIP. A form of XIP referred to as Expanded XIP that employs an expanded memory approach to map the PC Card memory.

Management interrupts. Interrupts generated by the HBA to notify card services that a status change has occurred within the PC Card environment.

Memory address windows. A range of memory address location programmed into the HBA that corresponds to the address locations used by a memory card to access its internal memory array. The HBA recognizes when software attempts an access to a location within the specified range and forwards the transaction to the memory card and socket.

Memory enabler. PC Card software designed to recognize, configure, and enable memory cards. Also called memory client driver.

PCMCIA System Architecture

Memory-only interface. The PC Card socket interface that supports only memory cards. The socket is always configured as a memory-only socket when a PC Card is first inserted. The socket can be changed to a memory or I/O interface if the enabler detects that the PC Card contains an I/O function.

Memory or I/O interface. The PC Card socket interface that supports both memory and I/O functions. The socket is changed from a memory-only to a memory-only socket when the HBA and PC Card are configured.

Memory technology driver. A client driver employed to handle low-level access to flash memory that requires special programming algorithms. Card services calls the memory technology driver when a flash memory enabler calls the bulk memory services. Also referred to as an MTD.

Metaformat. A formatting standard defined by the PC Card standard that describes low-level formatting information.

MTD. See Memory technology driver.

OE#. Output enabler (or, OE#) is a socket interface signal that indicated that a memory read command is being performed from PC Card memory.

Offset. The value (typically measured in bytes) that specifies an address location relative to a given base (or, start) address.

PCMCIA. Acronym for Personal Computer Memory Card International Association.

Point enablers. A PC Card enabler that recognizes and configures a PC Card by accessing the HBA and PC Card directly without the support of card and socket services.

Power Management. A hardware and software solution employed to conserve power.

Pulse mode interrupts. A form of interrupt triggering used to support interrupt sharing in the ISA environment. The interrupt is triggered on the trailing edge of the PC Card's negative pulse.

READY. An output pin from a memory card that indicates that it is ready for the next transaction. If deasserted, indicates that the memory card is busy performing a command and is not ready to receive the next transaction.

REG#. The register (REG#) pin is a socket interface pin asserted by the HBA to indicate an attribute memory access (when OE# or WE# is asserted) or an I/O access (when IORD# or IOWR# is asserted).

Registration. The process used by enablers to obtain the services of card services.

Reset. A socket interface pin used to reset the PC Card.

Resource allocation. The process employed by a PC Card enabler and card services to determine if system resources are available to be assigned to the PC Card.

Socket interface. The electrical and mechanical interface for PC Cards.

Socket services. PCMCIA specific software that provides low-level routines needed to access a given implementation of HBA. Socket services consists of a collection of functions that are typically called by card services to access HBA registers.

SPKR#. The speaker signal defined by the memory or I/O socket interface. Used to carry digital audio information from the PC Card to the host speaker.

Status change events. PC card and socket events that reflect some change in the status of the PC Card. When a status change event occurs, the HBA generates an interrupt to signal card services of the event.

STSCHG#. Status change is an output signal from an I/O card to signal that a status change has occurred.

SXIP. A type of XIP, called simple XIP, that maps PC Card memory into an address range no larger than 64KB in size.

TC. An output signal from the DMA controller indicating that the transfer is complete (i.e. the DMA controller has reached the Terminal Count). This signal is an input to PC Cards that support DMA transfer.

Tuples. The name given to the elements within the CIS that describe characteristics of the PC Card.

Type I card. A PC Card with a maximum thickness of 3.3mm.

Type II card. A PC Card with a maximum thickness of 3.3mm.

Type III card. A PC Card with a maximum thickness of 3.3mm.

Virtual disk. A memory card that is used to emulate a hard drive.

Vpp1 and Vpp2. Programming voltage pins defined for the socket interface.

PCMCIA System Architecture

VS1# and VS2#. Voltage sense pins defined for the low-voltage socket interface. These pins determine the initial Vcc level to apply to the socket.

WAIT#. A socket interface pin used by a PC Card to force wait states into a transaction.

WP. See Write-protect.

Write-protect. Write-protect (or, WP) is a output pin from a PC memory card indicating whether the user has chosen to write-protect memory (i.e. files).

XIP. See Execute-in-place.

XIP-expanded memory. A form of XIP referred to as Expanded XIP that employs an expanded memory approach to map the PC Card memory.

XIP-extended. A type of XIP, called Extended XIP, that requires a 386 or later x86 compatible processor. PC Card memory in this case is mapped into host system address space above 1MB.

XIP-Simple. A type of XIP, called simple XIP, that maps PC Card memory into an address range no larger than 64KB in size.

Index

—8—

8237 DMA controller, 90

—A—

Access speed, 22
Access timing, 134
AccessConfigRegisters, 280
AcknowledgeInterrupt, 292
Address offset, 127
Address translation, 121
Address window speed, 132
AddSocketServices, 291
AdjustResourceInfo, 291
Advanced client services functions defined,
264
AIMS interface, 107
Artificial card insertion events, 279
ATA
Sample CIS, 205
Support for two drives, 106
ATA CIS example, 202
ATA configuration options, 106, 201
ATA contiguous I/O address mapping, 105
ATA flash card designs, 103
ATA Function Extensions, 202
ATA interface, 101
ATA memory-mapped, 106
ATA PC Cards, 103
ATA primary address, 105
ATA registers, 346
ATA resource requirements, 105, 201
ATA secondary address, 105
ATA support for two drives, 201
ATA vs ATA PC Card, 105
Attribute memory, 48, 65

Attribute memory address, 57
Attribute memory read timing, 66
Attribute memory transfer speed, 65
Attribute memory write timing, 67

—B—

Battery location, 36
Battery warning, 63
Booting from ATA cards, 313
Booting from memory cards, 312
Booting from PC Cards, 294, 311
Bulk Memory functions, 289
Bulk memory services, 289
Bulk memory services functions defined,
264
BVD1 and BVD2, 60, 63

—C—

Call-back, 292, 293
Call-backs, 278
Card detection, 117
Card information structure, 24
Card insertion call-back, 296, 300, 303,
304
Card keying, 40
Card lock, 141
Card lock mechanisms, 141
Card power, 50
Card services, 22, 28, 264
Advanced Client functions, 290, 291
Card Services Specification, 18
Card services, power management, 267
Card servicesResource Management func-
tions:, 285, 286
Card Types, 35
CardInsertion call-back, 307

PCMCIA System Architecture

- CD1# and CD2#, 60, 117
- CE1# and CE2#, 55, 71, 134
- CheckEraseQueue, 289
- CIS, 24, 29, 145, 147
 - Access timing, 158
- Configuration Table, 195
- FAX/modem example, 191
- Flash example, 183
- SRAM example, 177
- Client driver, 24
- Client driver call-backs, 278
- Client services functions defined, 264
- Client services functions, 275
- Client utility functions, 280
- Client Utility functions defined, 264
- CloseMemory, 289
- CL-PD6722, 335
 - Address windows, 340
 - ATA registers, 346
 - ATA socket interface, 346
 - DMA support, 348
 - Features, 335
 - I/O window registers, 342
 - Interface Status register, 345
 - Interrupt and General Control register, 345
 - Interrupt steering, 345
 - Management interrupt configuration register, 344
 - Memory window registers, 340
 - Power control, 336
 - Status Change register, 344
 - Status change reporting, 344
 - Timing register set, 340
 - Vcc control, 337
 - Vpp1 control, 338
- Common memory, 48, 65
 - 16-bit address mode, 68
 - Common memory cycle time, 69
 - Common memory, 8-bit address mode, 70
- Configuration process, 229, 296
- Configuration registers, 163, 196, 214
- I/O base registers, 217

- I/O limit register, 218
- Configuration table entry, 288
- Configuraton registers
 - Configuration Option Register, 164, 196
 - Configuration Status Register, 166, 197, 216
 - Pin Replacement Register, 169, 197
 - Socket and Copy Register, 170
- CopyMemory, 289
- Cycle time, common memory, 69

—D—

- DACK/REG#, 93
- DACK2#, 89
- Data Transfers, 59
- Dedicated enablers, 232
- DeregisterClient, 280
- DeregisterEraseQueue, 289
- Device size, 135
- Digital Audio Waveform**, 75
- Dimensions of PC Cards, 36
- Direct mapping, 121
- DMA Acknowledge, 89
- DMA Bus Cycle, 94
- DMA channels, 90
- DMA clock, 95
- DMA Compressed Timing, 96
- DMA Extended Write option, 97
- DMA read/write definition, 87
- DMA request, 89
- DMA start memory address, 87
- DMA Transfer Count, 87
- DMA transfer count exhausted, 90
- DMA, EOP (End-of-Process), 90
- DMA, TC (Terminal Count reached), 90
- DMAC states, 95
- DREQ#, 92
- DREQ2, 89
- Dual-voltage cards, 40

—E—

EDC support, 141
Electrical Specification, 18
EMS, 316
Enabler, 22, 28
EOP (End-of-Process), 90
Event call-back, 291
Event notification, 291
ExCA, 321
ExCA, Card insertion/removal requirements, 326
ExCA, Card services requirements, 329
ExCA, Event wakeup, 330
ExCA, HBA requirements, 322
ExCA, I/O address windows, 323
ExCA, Interrupt requirements, 324
ExCA, Memory address windows, 323
ExCA, PC Card Interrupt requirements, 324
ExCA, Socket services requirements, 327
ExCA, Status Change Interrupt requirements, 324
ExCA, System Power requirements, 326
Execute-in-place, 315
EXIP, 316, 318
Expanded Memory Specification, 316

—F—

FFS, 303
First level interrupt handler, 224
Flash file system, 298, 303
Flash memory client drivers, 298
Flash memory enablers, 298
flash translation layer, 303
FLIH, 222, 224
FTL, 303

—G—

Generic enabler, 24
Generic enablers, 233

GetCardServicesInfo, 276
GetClientInfo, 291
GetConfigurationInfo, 280, 282, 283
GetFirstClient, 291
GetFirstPartition, 281, 284
GetFirstRegion, 281, 284
GetFirstTuple, 280, 283
GetFirstWindow, 286
GetMemPage, 286
GetNextClient, 291
GetNextPartition, 281, 284
GetNextRegion, 281, 284
GetNextTuple, 280, 283
GetNextWindow, 286
GetTupleData, 280, 283

—H—

HBA, 22, 26, 113, 115
 Functions, 115
HBA, EDC, 141
HBA, maximum number supported, 116
HBA, maximum sockets/HBA, 117
HBA, Power management, 141
HLDA, 89
HOLD, 89
Host bus adapter, 22. see HBA
Hot insertion, 22

—I—

I/O address window overlapping, 130
I/O address windows, 130
I/O address windows, direct mapped, 130
I/O addressing, 8-bit mode, 80
I/O data transfers, 79
I/O Read Command, 75
I/O size is 16-bits, 75
I/O Status Change, 75
I/O status change events, 78
I/O transfer timing, 81
I/O transfers
 16-bit access to 8-bit register, 82

PCMCIA System Architecture

I/O transfers with 16-bit registers, 82
I/O Write Command, 75
IDE Drive, 104
Initial program load, 312
Initialization, 265
Initialization byte, 313
INPACK#, 75, 78, 130
Input Port Acknowledge, 75
Interrupt Request, 75
Interrupts, 135
IOIS16#, 75, 77, 78, 80, 83
IORD#, 75
IORD# and IOWR#, 77
IOWR#, 75
IPL, 312
IPL from ATA cards, 204
IPL from PC Cards, 294, 311
IREQ#, 75, 78

—J—

JEIDA, 14
JEIDA battery status, 63
JEIDA Extensions, 19

—L—

Level interrupts, 137
Level mode interrupts, 136
LIM 4.0, 318
Low voltage socket, 41, 51
Low-voltage socket, 53
LXIP, 316, 318

—M—

Management interrupts, 139
MapLogSocket, 290
MapLogWindow, 290
MapMemPage, 286
MapPhySocket, 290
MapPhyWindow, 290
Media Storage Formats Specification, 19

Memory Address mapping, 121
Memory address windows, 126
 Overlapping, 127
Memory client drivers, 297
Memory data transfers, 65
Memory enablers, 297
Memory interface, 48
Memory or I/O interface, 74, 133
Memory read timing, Attribute memory,
 66
Memory technology driver, 305
Memory technology drivers, 297
Memory transfer speed, 65
Memory transfer using WAIT#, 71
Memory-only interface, 133
Metaformat, 29
 Layers, 31
Metaformat Specification, 18
ModifyConfiguration, 286
ModifyWindow, 286
MTD, 290, 297, 303, 305
Multiple function card, 209, 210
Multiple function interrupts, 219, 221

—O—

OE#, 59
OpenMemory, 289, 300, 304
Overlapping I/O Windows, 130
Overlapping memory windows, 127

—P—

PC Card Address, 55
PC Card ATA specification, 19
PC Card Data, 59
PC Card detection, 117
PC Card device size, 135
PC Card dimensions, 36
PC Card Interrupts, 135
PC Card socket, 22
PC Cards, 14
PCMCIA, 14

PCMCIA Evolution, 17
PCMCIA Extensions, 19
PCMCIA.SYS, 317
Physical Specification, 18
Pin definition, memory interface, 48
Pin length, 42
Pin replacement register, 79
Point enablers, 233, 309
POST, 313
Power Control register, 337
Power Management, 141, 267
Power switching, 119
Pulse interrupts, 138
Pulse mode interrupts, 136

—R—

Read transfers, common memory, 68
ReadMemory, 289, 305
READY, 60, 62
READY Status, 62
REG#, 59, 77
RegisterClient, 276, 300, 303, 304
RegisterEraseQueue, 289
RegisterMTD, 290
RegisterTimer, 290
Registration, 275
RegistrationComplete, 277, 279
ReleaseConfiguration, 286
ReleaseDMA, 286
ReleaseExclusive, 291
ReleaseIO, 285
ReleaseIRQ, 285
ReleaseSocketMask, 286
ReleaseWindow, 285
ReplaceSocketServices, 291
RequestConfiguration, 286, 288
RequestDMA, 285
RequestExclusive, 291
Requesting resources, 287
RequestIO, 285
RequestIRQ, 285
RequestSocketMask, 286

RequestWindow, 285
Reset, 64
Resource allocation, 284
Resource management functions defined, 264
ReturnSSEntry, 290
Ring Indicate, 330

—S—

SetRegion, 290
Socket
 2.x compliant socket, 40
 Access timing, 134
 Keying, 2.x socket, 40
 Keying, low voltage socket, 40
 Low voltage, 52
Socket Address, 16-bit mode, 56
Socket Address, 8-bit mode, 57
Socket and Copy Register, 106, 202
Socket functions, 249
Socket interface, 116
 Address lines, 55
 Data lines, 59
 Vpp1 and Vpp2, 54
Socket interface control, 133
Socket interface selection, 133
Socket keying, 40
Socket power, 50, 119
Socket service
 Installation, 237
Socket services, 22, 27, 235
 AcknowledgeInterrupt, 248
 Adapter functions, 243
 Adapter functions defined, 238
 EDC functions defined, 239
 Function summary, 237
 functions, 239
 GetAccessOffsets, 248
 GetAdapter, 245
 GetAdapterCount, 243
 GetSetPriorHandle, 244
 GetSetSSAddr, 248

PCMCIA System Architecture

- GetSocket, 249
- GetSSInfo, 243
- GetStatus, 252
- GetVendorInfo, 247
- GetWindow, 253
- InquireAdapter, 245
- InquireSocket, 249
- InquireWindow, 253
- ResetSocket, 252
- SetAdapter, 245
- SetSocket, 249
- SetWindow, 253
- Socket functions defined, 238
- VendorSpecific, 247
- Window functions, 253
- Window functions defined, 238
- x86 function codes, 241
- Socket Services Specification, 18
- Socket Status Change, 119
- Socket transfer timing control, 133
- Socket, low voltage, 41
- Sockets, 22
- SPKR#, 75, 79
- SRAM client drivers, 298, 299, 301
- SRAM enablers, 298
- Standard socket, 52
- Status change events, 60, 277
- Status signals, 60
- STSCHG#, 75, 79
- SXIP, 316, 318

—T—

- TC (DMA Terminal Count), 90
- TC, to PC Card, 93
- Tuples, 148, 330
 - Definition, 148
 - Summary listing, 158
 - Tuple format, 148
- Type 1 card, 36
- Type I card, Extended, 39
- Type II card, 37

- Type II card, Extended, 39
- Type III card, 38
- Types of PC Card, 36

—V—

- ValidateCIS, 290
- Vcc, 119
- Vcc, 2.x socket, 50
- Vcc, low voltage socket, 51
- VendorSpecific, 291
- Virtual disk, 22
- Voltage Sense, 51, 120
- Voltage sense pins, 50
- Vpp1 and Vpp2, 54, 120
- VS1# and VS2#, 50, 51, 120

—W—

- WAIT#, 64, 71, 79, 134
- Wait# timing, 71
- Word vs byte access, 134
- WP, 60, 62
- write protect switch location, 36
- Write timing, attribute memory, 67
- WriteMemory, 289, 305
- Write-Protect, 62

—X—

- x86 function codes, 271
- XIP, 315
- XIP application, 318
- XIP device driver, 317
- XIP file management, 316
- XIP loader, 317
- XIP manager, 318
- XIP Specification, 19
- XIP.SYS, 317
- XIP-expanded memory, 318
- XIP-extended, 318
- XIP-Simple, 318



informIT


www.informit.com

YOUR GUIDE TO IT REFERENCE



Articles

Keep your edge with thousands of free articles, in-depth features, interviews, and IT reference recommendations – all written by experts you know and trust.



Online Books

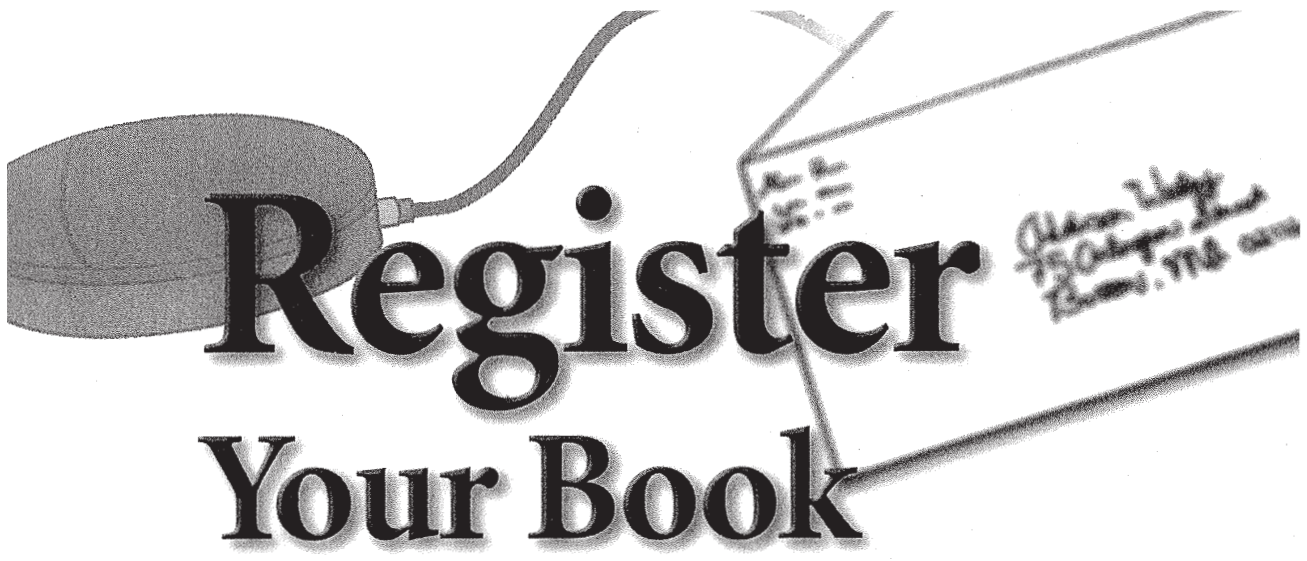
Answers in an instant from **InformIT Online Book's** 600+ fully searchable on line books. For a limited time, you can get your first 14 days **free**.

Safari
TECH BOOKS ONLINE



Catalog

Review online sample chapters, author biographies and customer rankings and choose exactly the right book from a selection of over 5,000 titles:



Register Your Book

at www.awprofessional.com/register

You may be eligible to receive:

- Advance notice of forthcoming editions of the book
- Related book recommendations
- Chapter excerpts and supplements of forthcoming titles
- Information about special contests and promotions throughout the year
- Notices and reminders about author appearances, tradeshow, and online chats with special guests



Contact us

If you are interested in writing a book or reviewing manuscripts prior to publication, please write to us at:

Editorial Department
Addison-Wesley Professional
75 Arlington Street, Suite 300
Boston, MA 02116 USA
Email: AWPro@aw.com



Visit us on the Web: <http://www.awprofessional.com>

“The realization of full compatibility and interoperability with PC card technology is built on a thorough understanding of PC card architecture. *PCMCIA System Architecture* explains this architecture with a systematic approach that is clear and concise.”

—Michael J. Homic
Engineering Manager, Peripheral Systems Development
Dell Computer Corporation

PCMCIA System Architecture: 16-Bit PC Cards, Second Edition describes PC card hardware and software interfaces and their relationships to overall system design. Developed by the Personal Computer Memory Card International Association (PCMCIA) and the Japan Electronics Industry Development Association (JEIDA), the PC Card Standard defines a standard hardware and software interface for small removable 16-bit cards. The PC Card Standard also defines a new 32-bit PC Card called CardBus. For more information on this standard see *CardBus System Architecture* (Addison-Wesley, 1995). PCMCIA expert Don Anderson provides a comprehensive treatment of the interface including:

- socket interface
- card information structure
- host adapter requirements
- card electrical interface
- bus cycle control
- address translation
- low voltage support
- translation of card interrupts
- card event notification
- DMA socket interface
- enabling and configuring PC cards
- card and socket services

This book also examines an adapter implementation using Cirrus Logic CL-PD6722.

If you design or test hardware or software that involves 16-bit PC cards, *PCMCIA System Architecture* is an essential, time-saving tool.

The *PC System Architecture Series* is a crisply written and comprehensive set of guides to the most important PC hardware standards. Each title explains from a programmer's perspective the architecture, features, and operations of systems built using one particular type of chip or hardware specification.

MindShare, Inc., is one of the leading technical training companies in the hardware industry, providing innovative courses for dozens of companies, including Intel, IBM, and Compaq.

Don Anderson passes on his wealth of experience in digital electronics and computer design by training engineers, programmers, and technicians for MindShare.

Cover design by Barbara T. Atkinson
Cover photograph by Tatsuhiko Shimada/Photonica



ISBN 0-201-40991-7

\$39.99 US
\$62.99 CANADA

ADDISON-WESLEY
Pearson Education