Plug and Play ISA Specification

Version 1.0a
May 5, 1994

Revision History

Issue Date	Comments
March 1, 1993	Preliminary draft
March 27, 1993	Revised based on review comments
April 29, 1993	Revised based on review comments
May 28, 1993	Release 1.0
March 24, 1994	Release 1.0a. Incorporated eratta and clarifications. OBSOLETED
May 5, 1994	Release 1.0a. Final Changes: - Incorporated clarifications from 'eratta.doc' - Additional clarification on order of resource descriptors - Additional clarification on use of dependent functions - Added support for 32-bit memory - Clarified structure of Vendor ID and use of Logical Device ID - Removed redundancies and contradictions with BIOS specification

Microsoft and Intel do not make any representation or warranty regarding this specification or any product or item developed based on this specification. Microsoft and Intel disclaim all express and implied warranties, including but not limited to the implied warranties of merchantability, fitness for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Microsoft and Intel do not make any warranty of any kind that any item developed based on this specification, or any portion of it, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is your responsibility to seek licenses for such intellectual property rights where appropriate. Microsoft and Intel shall not be liable for any damages arising out of or in connection with the use of this specification, including liability for lost profit, business interruption, or any other damages whatsoever. Some states do not allow the exclusion or limitation of liability for consequential or incidental damages; the above limitation may not apply to you.

Plug and Play ISA Specification
(c) Copyright 1993, 1994 by Intel Corporation and Microsoft Corporation.

Table of Contents

Revision History ..... **ii**
1. Introduction **2**
1.1. Goals 3
1.2. Implementation Considerations 3
2. Auto-configuration Sequence **4**
2.1. Plug and Play System Configuration Sequence 4
2.2. Plug and Play Card Configuration Sequence 6
3. Isolation and Identification **7**
3.1. Auto-configuration Ports 7
3.1.1. ADDRESS Port 8
3.1.2. WRITE_DATA Port 8
3.1.3. READ_DATA Port 8
3.2. Initiation Key 8
3.3. Isolation Protocol 9
3.3.1. Hardware Protocol 11
3.3.2. Software Protocol 12
4. Programming Plug and Play Devices **13**
4.1. State Summary 13
4.2. Plug and Play Register Summary 14
4.3. Control Register Summary 15
4.4. Plug and Play Isolation Sequence 19
4.5. Reading Resource Data 20
4.6. Configuring Card Resource Usage 20
4.6.1. Order of Configuration Reads 21
4.6.2. Resource Programming 22
4.7. Run Time Access to Plug and Play registers 23
5. Plug and Play Functionality **24**
5.1. Minimum Functionality for a Configurable Logical Device 24
5.2. Minimum Level of Functionality for a Non-configurable Logical Device 24
6. Plug and Play Resources **25**
6.1. Serial Identifier 25
6.1.1. Vendor ID 26
6.1.2. Serial/Unique Number 26
6.1.3. Checksum 26
6.2. Plug and Play Resource Data Types 27
6.2.1. Resource Data Requirements 27
6.2.2. Small Resource Data Type 28
6.2.2.1. Plug and Play Version Number 29
6.2.2.2. Logical Device ID 30
6.2.2.3. Compatible Device ID 31
6.2.2.4. IRQ Format 32
6.2.2.5. DMA Format 33
6.2.2.6. Start Dependent Functions 34
6.2.2.7. End Dependent Functions 34
6.2.2.8. I/O Port Descriptor 35
6.2.2.9. Fixed Location I/O Port Descriptor 36
6.2.2.10. Vendor Defined 36
6.2.2.11. End Tag 36

6.2.3. Large Resource Data Type 37
6.2.3.1. Memory Range Descriptor 38
6.2.3.2. ANSI Identifier String 40
6.2.3.3. Unicode Identifier String 40
6.2.3.4. Vendor Defined 41
6.2.3.5. 32-bit Memory Range Descriptor 41
6.2.3.6. 32-bit Fixed Location Memory Range Descriptor 44
7. Resource Data and Dependent Functions 46
7.1. Example One 46
7.2. Example Two 47
7.3. Example Three 47
7.4. Example Four 47
Appendix A. Plug and Play Standard Registers 49
A.1. Plug and Play Card Control Registers 50
A.2 Plug and Play Logical Device Control Registers 52
A.3. Plug and Play Logical Device Configuration Registers 53
A.3.1. Memory Configuration Registers 53
A.3.2. I/O Configuration Registers 57
A.3.3. Interrupt Configuration Registers 58
A.3.4. DMA Configuration Registers 59
A.3.5. Reserved and Vendor Defined Configuration Registers 59
A.4. Reserved Register 59
Appendix B. LFSR Definition 60
B.1. Initiation LFSR Function 60
B.2. LFSR Checksum Functions 61
Appendix C. Possible Enhancements 62
C.1. Plug and Play Boot Devices 62
C.2. BIOS Support for Plug and Play Devices 63
C.3. Plug and Play Devices and Non-Plug and Play Operating Systems 63
Appendix D. Sample Configuration Record for ABC Ethernet Card 64

List of Tables

Table 1. Auto-configuration Ports 7
Table 2. Plug and Play Header 25
Table 3. Small Resource Data Type Tag Bit Definitions 28
Table 4. Small Resource Items 28
Table 5. Large Resource Data Type Tag Bit Definitions 37
Table 6. Large Resource Items 37
Table A-1. Plug and Play Standard Registers 50
Table A-2. Plug and Play Logical Device Control Registers 52
Table A-3. Memory Space Configuration 54
Table A-4. 32-bit Memory Space Configuration 55
Table A-5. I/O Space Configuration 57
Table A-6. Interrupt Configuration 58
Table A-7. DMA Channel Configuration 59
Table A-8. Logical Device Configuration 59
Table A-9. Logical Device Reserved 59

List of Figures

Figure 1. ISA Configuration Flow for Existing Non-Plug and Play BIOS 4
Figure 2. Plug and Play ISA Configuration Flow for Plug and Play BIOS 5
Figure 3. Logic Flow for Auto-configuration 7
Figure 4. Plug and Play ISA Card Isolation Algorithm 9
Figure 5. Shifting of Serial Identifier 10
Figure 6. Plug and Play High-Level Register Map 14
Figure 7. Serial Identifier and Resource Data 16
Figure 8. Plug and Play ISA Card State Transitions 18
Figure A-1. Plug and Play Register Map 49
Figure B-1. Initiation Key LFSR 60
Figure B-2. Checksum LFSR 61

Abstract

This specification presents a mechanism to provide automatic configuration capability to *Industry Standard Architecture* (ISA) cards thus enabling full Plug and Play in the PC. The essential elements of Plug and Play ISA are:

- Isolate the ISA card
- Read the card's resource data
- Identify the card and configure its resources
- Locate a driver for the card

The mechanism has the flexibility to allow cards to come up either inactive or active. Cards required to participate in the boot process typically come up active. Information that identifies the card and describes the system resources which are requested by the card, such as memory and I/O space, DMA channel, and interrupt level supported is maintained in a standard read-only format. This allows the Plug and Play software to identify system resource usage, arbitrate conflicts, and, optionally, re-configure a card. This process is done automatically at every hard reset of the system. Plug and Play ISA cards will inter-operate with standard ISA cards in a fully compatible manner.

Possible BIOS extensions required to support Plug and Play ISA cards are also discussed in this specification. However, user interface issues for installation of device drivers are not addressed.

1. Introduction

The ISA bus is the most popular expansion standard in the PC industry. The bus architecture requires the allocation of memory and I/O address spaces, DMA channels and interrupt levels among multiple ISA cards. However, the ISA interface has no defined hardware or software mechanism for allocating these resources. As a result, configuration of ISA cards is typically done with "jumpers" that change the decode maps for memory and I/O space and steer the DMA and interrupt signals to different pins on the bus. Further, system configuration files may need to be updated to reflect these changes. Users typically resolve sharing conflicts by referring to documentation provided by each card manufacturer. For the average user, this configuration process can be unreliable and frustrating.

Alternative bus standards (for example, Micro Channel and EISA) have hardware and software mechanisms to identify the resources requested by a card and resolve conflicts. These mechanisms are not compatible with the installed base of PCs with ISA card slots.

This specification proposes a hardware and software mechanism for incorporation in the next generation of ISA cards, referred to as Plug and Play ISA cards, that enables resolution of conflicts between Plug and Play ISA cards. In other words, the Plug and Play software optimally allocates system resources between the Plug and Play ISA cards and other devices in the system without user intervention.

In a system that uses only Plug and Play ISA cards, it will be possible to achieve full auto- configuration. However, it is recognized that the current generation or standard ISA cards will co-exist with Plug and Play ISA cards in the same system. In such systems, the configuration solution needs to be augmented in the BIOS and/or operating system to manage and arbitrate ISA bus resources. User interaction may still be necessary in some cases.

This specification defines mechanisms that each Plug and Play ISA card must implement to support identification, resource usage determination, conflict detection, and conflict resolution. This specification also presents a process for Plug and Play software to automatically configure the new cards without user intervention.

1.1. Goals

The following are the architectural goals to support full Plug-and-Play of ISA systems:

1. Focus on ease-of-use for the end-user
2. Provide the ability to uniquely address individual cards even when two or more cards are configured to use the same system resources.
3. Define a protocol that provides the ability to read a configuration template on each card which identifies current resource usage and options for each system resource requested on each card
4. Define a mechanism to set or modify the current configuration of each card
5. Provide a range of cost-benefit compliance points
6. Maintain backward and forward compatibility as follows:
    - Plug and Play ISA cards will electrically and functionally inter-operate with standard ISA cards in any existing ISA based PC system; however, the system may not be fully auto-configurable.
    - With the addition of Plug and Play software (i.e. utilities, BIOS enhancements or operating system enhancements, and user interrogation), the above system can be made increasingly auto-configurable.
    - A system with only Plug and Play ISA cards and appropriate software will be fully auto-configurable.

1.2. Implementation Considerations

The solution of the ISA configuration problem addresses major concerns of end-users, system integrators, ISA card manufacturers, and operating system vendors. It also presents an excellent business opportunity for providing measurable value and differentiation in the short term.

In defining this mechanism, the cost of incremental hardware has been taken into consideration. The specification allows an implementor to make tradeoffs in the area of technology and level of integration depending on the functionality and market for each card.

2. Auto-configuration Sequence

This section presents an overview of the auto-configuration flow. Hardware and software mechanisms will be described in more detail in later sections.

2.1. Plug and Play System Configuration Sequence

The next two figures show a possible flow of the configuration process for Plug and Play ISA cards. The first figure represents configuration on a system with a standard BIOS, the second figure is configuration in a system with a Plug and Play BIOS. These figures are meant as an example of how this process will work, the exact implementation details may be different.

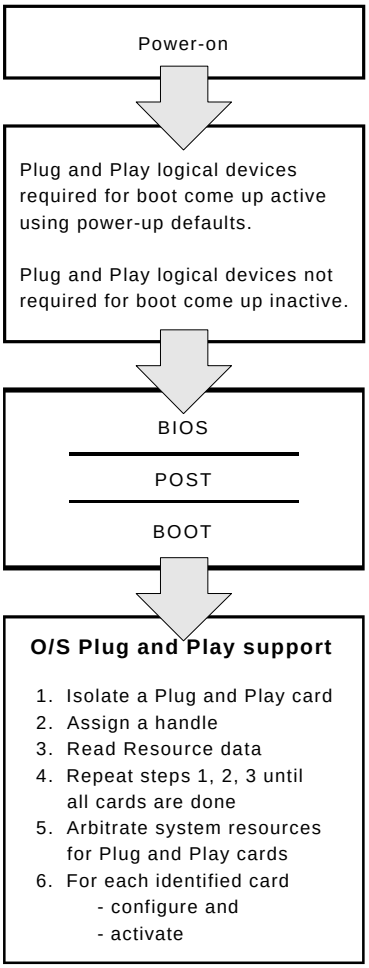

Figure 1. ISA Configuration Flow for Existing Non-Plug and Play BIOS

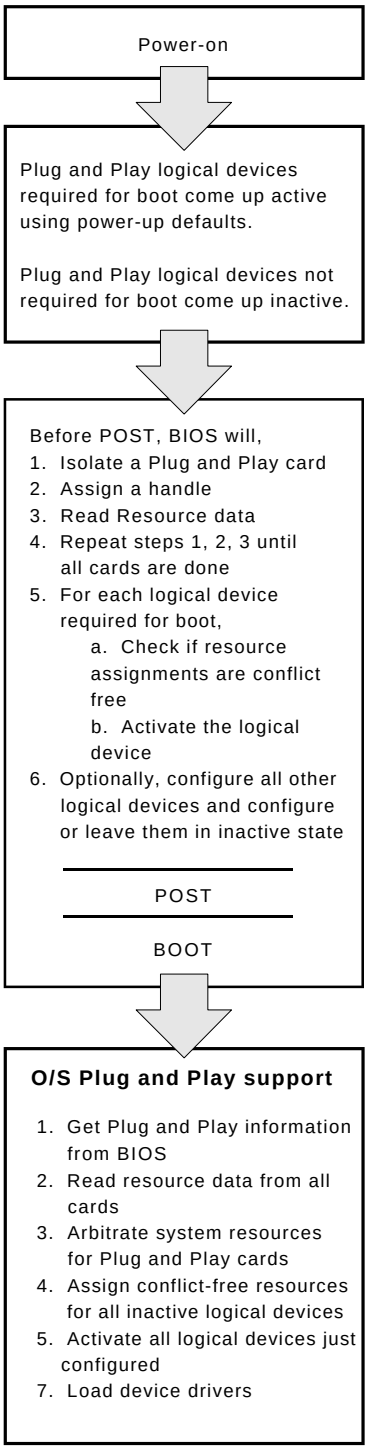

Figure 2. Plug and Play ISA Configuration Flow for Plug and Play BIOS

2.2. Plug and Play Card Configuration Sequence

The major steps of the auto-configuration process are as follows:

- Put all Plug and Play ISA cards in configuration mode
- Isolate one Plug and Play ISA card at a time
- Assign a handle and read the card's resource data structure
- After the resource requirements and capabilities are determined for all cards, use the handle to assign conflict free resources to each card
- Activate all Plug and Play ISA cards and remove them from configuration mode

The Plug and Play software identifies and configures devices using a set of commands defined in this specification. The commands are executed using three, 8-bit I/O ports. 16-bit accesses (assertion of IOCS16#) to the configuration ports are not supported. A sequence of data writes to one of the ports is used as the key to enable the Plug and Play logic on all cards in the system. This sequence, referred to as the initiation key, is described in the next section.

All Plug and Play cards respond to the same I/O port addresses so the Plug and Play software needs an isolation mechanism to address one particular card at a time. The isolation protocol uses a unique number on each card to isolate one Plug and Play card at a time. After isolation, the Plug and Play software assigns each card a handle, which is used to select that unique Plug and Play card. The use of the handle eliminates the need to use the more elaborate and time consuming isolation protocol to select a unique card.

Each card supports a readable resource data structure that describes the resources supported and those requested by the functions on that card. The structure supports the concept of multiple functions per ISA card. Each function is defined as a logical device. Plug and Play resource information is provided for each logical device and each logical device is independently configured through the Plug and Play standard registers.

Following isolation, the Plug and Play software reads the resource data structure on each card. When all resource capabilities and demands are known, a process of resource arbitration is invoked to determine resource allocation to each ISA card.

The configuration of ISA cards is done using the command registers specified for each resource type. It should be noted that some ISA functions may not be re-configurable. In these cases, the resources requested will be equivalent to the resources supported. However, the resource data structure informs the arbiter that it cannot assign these resources to other Plug and Play cards in the system.

After the assignment of resources, an I/O conflict detection mechanism may be invoked. This mechanism provides a means to insure that I/O resources assigned are not in conflict with standard ISA cards.

The command set also supports the ability to activate or deactivate the function(s) on the card.

After configuration is complete, Plug and Play cards are removed from configuration mode. To re-enable configuration mode, the initiation key needs to be re-issued. This prevents accidental erasure of the configuration information.

3. Isolation and Identification

This section specifies the assignment of auto-configuration ports and the isolation protocol.

3.1. Auto-configuration Ports

Three 8-bit ports are used by the software to access the configuration space on each Plug and Play ISA card. The ports are listed in table 1. The configuration space is implemented as a set of 8-bit registers. These registers are used by the Plug and Play software to issue commands, check status, access the resource data information, and configure the Plug and Play hardware.

The ports have been chosen so as to avoid conflicts in the installed base of ISA functions, while at the same time minimizing the number of ports needed in the ISA I/O space.

Table 1. Auto-configuration Ports

Port Name	Location	Type
ADDRESS	0x0279 (Printer status port)	Write-only
WRITE_DATA	0x0A79 (Printer status port + 0x0800)	Write-only
READ_DATA	Relocatable in range 0x0203 to 0x03FF	Read-only

The ADDRESS and WRITE_DATA ports are located at fixed addresses. The WRITE_DATA port is located at an address alias of the ADDRESS port. All three auto-configuration ports use a 12-bit ISA address decode.

The READ_DATA port is relocatable within the I/O range from 0x0203h to 0x03FFh. This is the only readable auto-configuration port.

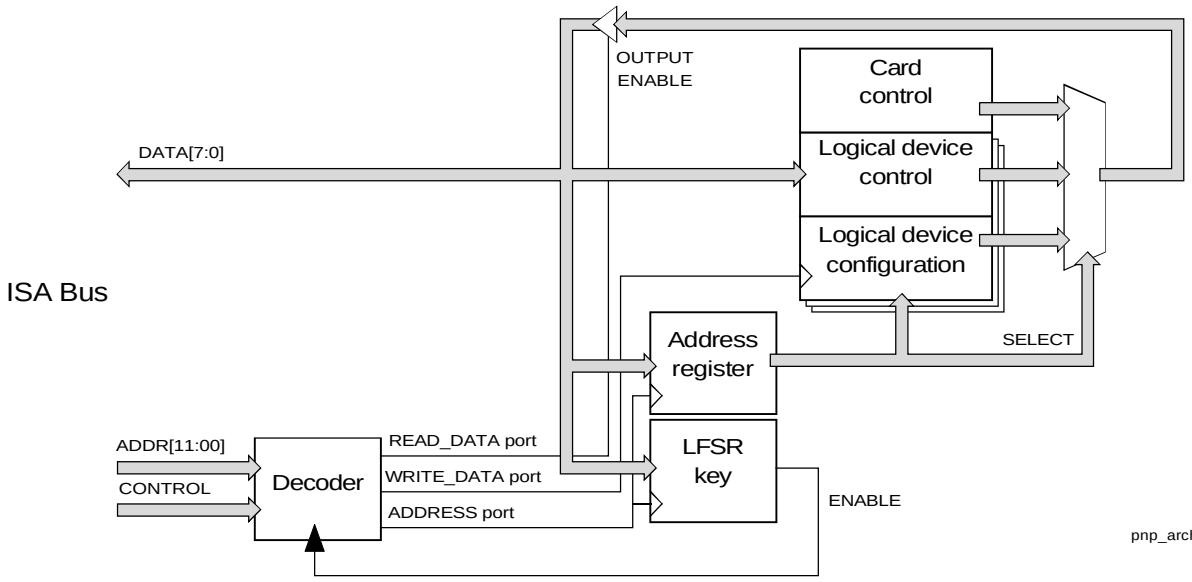

Figure 3. Logic Flow for Auto-configuration

3.1.1. ADDRESS Port

The Plug and Play registers are accessed by first writing the address of the desired register to the ADDRESS port, followed by a read of data from the READ_DATA port or a write of data to the WRITE_DATA port. A write to the ADDRESS port may be followed by any number of WRITE_DATA or READ_DATA accesses to the same register location without the need to write to the ADDRESS port before each access.

The ADDRESS port is also the write destination of the initiation key, which is described later.

3.1.2. WRITE_DATA Port

The WRITE_DATA port is used to write information to the Plug and Play registers. The destination of the data is determined by the last setting of the ADDRESS port.

3.1.3. READ_DATA Port

The READ_DATA port is used to read information from the Plug and Play registers. The source of the data is determined by the last setting of the ADDRESS port.

The address of the READ_DATA port is set by writing the proper value to a Plug and Play control register. The isolation protocol verifies that the location selected for the READ_DATA port is free of conflict.

3.2. Initiation Key

The Plug and Play logic is quiescent on power up and must be enabled by software.

The initiation key places the Plug and Play logic into configuration mode. This is done by a predefined series of writes to the ADDRESS port. The write sequence is decoded by on-card logic. If the proper series of I/O writes is detected, then the Plug and Play auto-configuration ports are enabled.

The hardware check of the initiation key is implemented as a *linear feedback shift register* (LFSR). A hardware diagram of the LFSR is shown in the Appendix B.

Software generates the LFSR sequence and writes it to the ADDRESS port as a sequence of 8-bit write cycles. The hardware compares the byte of write data with the value in the shift register at each write. Any time the data does not match, the hardware will reset to the initial value of the LFSR. Software should reset the LFSR to its initial value by a sequence of two write cycles of 0x00 to the ADDRESS port before the initiation key is sent.

The initial value of the LFSR and the exact sequence are documented in Appendix B.

3.3. Isolation Protocol

A simple algorithm is used to isolate each Plug and Play card. This algorithm uses the signals on the ISA bus and requires lock-step operation between the Plug and Play hardware and the isolation software.

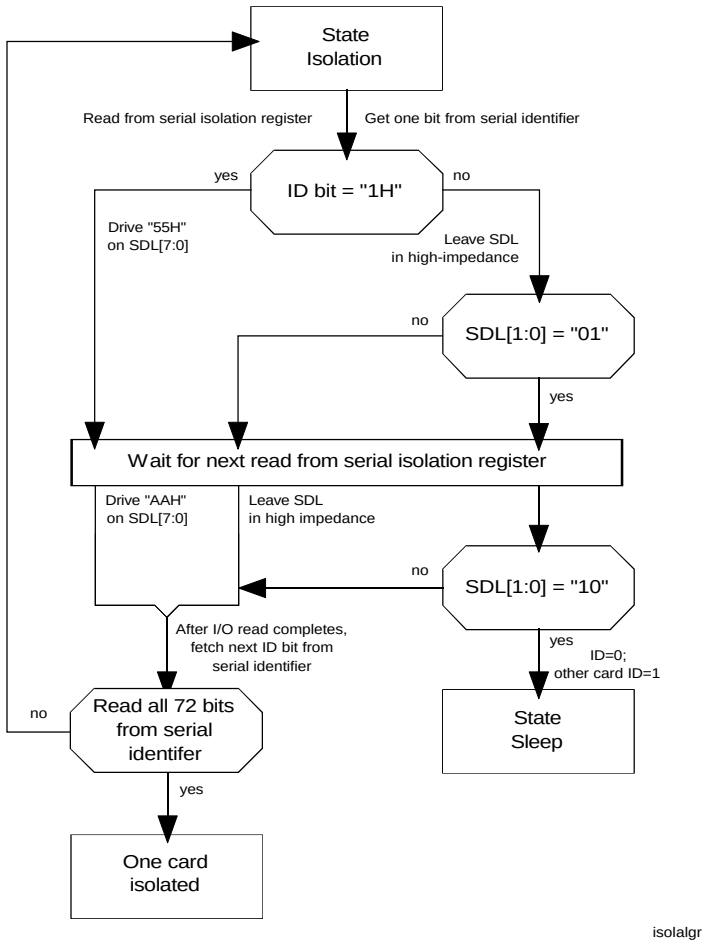

Figure 4. Plug and Play ISA Card Isolation Algorithm

3.3. Isolation Protocol (cont.)

The key element of this mechanism is that each card contains a unique number, referred to as the *serial identifier* for the rest of the discussion. The serial identifier is a 72-bit unique, non-zero, number composed of two, 32-bit fields and an 8-bit checksum. The first 32-bit field is a vendor identifier. The other 32 bits can be any value, for example, a serial number, part of a LAN address, or a static number, as long as there will never be two cards in a single system with the same 64 bit number. The serial identifier is accessed bit-serially by the isolation logic and is used to differentiate the cards. Refer to section 6.1. *Serial Identifier*, for further information.

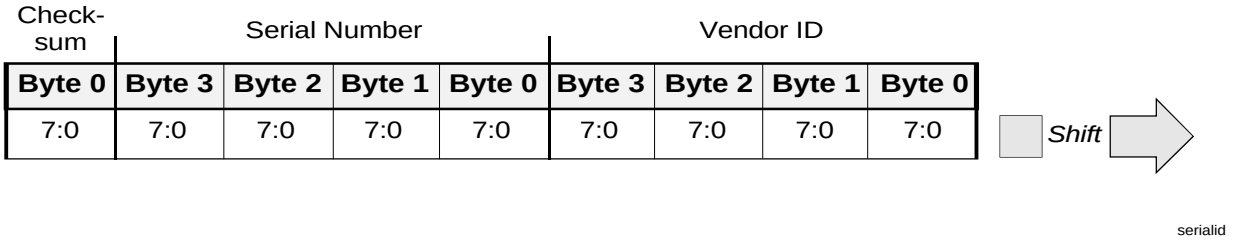

Figure 5. Shifting of Serial Identifier

The shift order for all Plug and Play serial isolation and resource data is defined as bit[0], bit[1], and so on through bit[7].

3.3.1. Hardware Protocol

The isolation protocol can be invoked by the Plug and Play software at any time. The initiation key, described earlier, puts all cards into configuration mode. The hardware on each card expects 72 pairs of I/O read accesses to the READ_DATA port. The card's response to these reads depends on the value of each bit of the serial identifier which is being examined one bit at a time, in the sequence shown in figure 5.

If the current bit of the serial identifier is a "1", then the card will drive the data bus to 0x55 to complete the first I/O read cycle. If the bit is "0", then the card puts its data bus driver into high impedance. All cards in high impedance will check the data bus during the I/O read cycle to sense if another card is driving D[1:0] to "01." During the second I/O read, the card(s) that drove the 0x55, will now drive a 0xAA. All high impedance cards will check the data bus to sense if another card is driving D[1:0] to "10."

If a high impedance card sensed another card driving the data bus with the appropriate data during both cycles, then that card ceases to participate in the current iteration of card isolation. Such cards, which lose out, will participate in future iterations of the isolation protocol.

NOTE: *During each read cycle, the Plug and Play hardware drives the entire 8-bit data bus, but only checks the lower 2 bits .*

If a card was driving the bus or if the card was in high impedance and did not sense another card driving the bus, then it should prepare for the next pair of I/O reads. The card shifts the serial identifier by one bit and uses the shifted bit to decide its response.

The above sequence is repeated for the entire 72-bit serial identifier.

At the end of this process, one card remains. This card is assigned a handle referred to as the *Card Select Number* (CSN) that will be used later to select the card. Cards which have been assigned a CSN will not participate in subsequent iterations of the isolation protocol. Cards must be assigned a CSN before they will respond to the other commands defined in the specification.

It should be noted that the protocol permits the 8-bit checksum to be stored in non-volatile memory on the card or generated by the on-card logic in real-time. The same LFSR algorithm described in the initiation key section is used in the checksum generation. The exact description of the checksum algorithm and an example are shown in Appendix B.

Plug and Play cards must not drive the IOCHRDY signal during serial isolation. However, cards may drive IOCHRDY at any other time.

3.3.2. Software Protocol

The Plug and Play software sends the initiation key to all Plug and Play cards to place them into configuration mode. The software is then ready to perform the isolation protocol.

The Plug and Play software generates 72 pairs of I/O read cycles from the READ_DATA port. The software checks the data returned from each pair of I/O reads for the 0x55 and 0xAA driven by the hardware. If both 0x55 and 0xAA are read back, then the software assumes that the hardware had a "1" bit in that position. All other results are assumed to be a "0."

During the first 64 bits, software generates a checksum using the received data. The checksum is compared with the checksum read back in the last 8 bits of the sequence.

There are two other special considerations for the software protocol. During an iteration, it is possible that the 0x55 and 0xAA combination is never detected. It is also possible that the checksum does not match. If either of these cases occur on the first iteration, it must be assumed that the READ_DATA port is in conflict. If a conflict is detected, then the READ_DATA port is relocated. The above process is repeated until a non-conflicting location for the READ_DATA port is found. The entire range between 0x200 and 0x3FF is available, however in practice it is expected that only a few locations will be tried before software determines that no Plug and Play cards are present.

During subsequent iterations, the occurrence of either of these two special cases should be interpreted as the absence of any further Plug and Play cards (i.e. the last card was found in the previous iteration). This terminates the isolation protocol.

NOTE: *The software must delay 1 msec prior to starting the first pair of isolation reads, and must wait 250 µsec between each subsequent pair of isolation reads. This delay gives the ISA card time to access information from possibly very slow storage devices.*

4. Programming Plug and Play Devices

This section describes how configuration resource data is read from Plug and Play ISA cards as well as how resource selections are programmed. The Plug and Play state machine and Plug and Play commands are introduced. The Plug and Play state machine is shown in figure **Error! Bookmark not defined.**. Addresses for Plug and Play registers are summarized in Appendix A.

4.1. State Summary

Before explaining the Plug and Play state transitions it is necessary to introduce a register in each ISA card called the *Card Select Number* (CSN). The CSN is an 8-bit register used to select one or more ISA cards when those cards are in certain states. The CSN is defined as an 8-bit register to allow a wide variety of devices to manage their configuration and control using this mechanism. The CSN is defined such that all cards power-up with this register is set to 0x0. Once a card has been isolated, the CSN on that card is assigned a unique value. This value enables the Plug and Play software to select this card at later points in the configuration process, without going through the isolation protocol again.

The Plug and Play states are summarized as follows:

- *Wait for Key* - All cards enter this state after power-up reset or in response to the Reset and Wait for Key commands. No commands are active in this state until the initiation key is detected on the ISA bus. The *Wait for Key* state is the default state for Plug and Play cards during normal system operation. After configuration and activation, software should return all cards to this state.

- *Sleep* - In this state, Plug and Play cards wait for a Wake[CSN] command. This command will selectively enable one or more cards to enter either the *Isolation* or *Config* states based on the write data and the value of the CSN on each card. Cards leave the *Sleep* state in response to a Wake[CSN] command when the value of write data bits[7:0] of the Wake[CSN] command matches the card's CSN. If the write data for the Wake[CSN] command is zero then all cards that have not been assigned a CSN will enter the *Isolation* state. If the write data for the Wake[CSN] command is not zero then the one card whose assigned CSN matches the parameter of the Wake[CSN] command will enter the *Config* state.

- *Isolation* - In this state, Plug and Play cards respond to reads of the Serial Isolation register as described in the previous chapter on isolation protocol. Once a card is isolated, a unique CSN is assigned. This number will later be used by the Wake[CSN] command to select the card. Once the CSN is written, the card transitions to the *Config* state.

- *Config* - A card in the *Config* state responds to all configuration commands including reading the card's resource configuration information and programming the card's resource selections. Only one card may be in this state at a time.

4.2. Plug and Play Register Summary

Plug and Play card standard register space is divided into three parts; card control, logical device control, and logical device configuration. There is exactly one of each card control register on each ISA card. Card control registers are used for global functions that control the entire card. Logical device control registers and logical device configuration registers are repeated for each logical device. Logical device control registers control device functions, such as enabling the device onto the ISA bus. Logical device configuration registers are used to program the device's ISA bus resource use. There are several vendor defined registers in all three register locations so vendors may configure non-standard ISA resources through the Plug and Play mechanism as well.

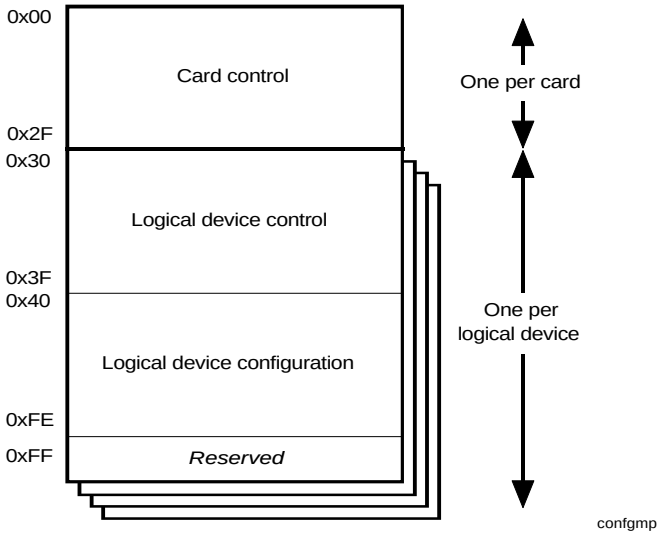

Figure 6. Plug and Play High-Level Register Map

4.3. Control Register Summary

Plug and Play cards respond to commands written to Plug and Play registers as well as certain ISA bus conditions. These commands are summarized below:

- RESET_DRV - This is the ISA bus reset signal. When a Plug and Play card detects this signal it enters the *Wait for Key* state. All CSNs are reset to 0x0. The configuration registers for all logical devices are loaded with their power-up values from non-volatile memory or jumpers. All non-boot logical devices become inactive. Boot devices become active using their power-up ISA resources.

  Note: The software must delay 1 msec after RESET_DRV before accessing the auto-configuration ports.

- Config control register - The Config Control register consists of three independent commands which are activated by writing a "1" to their corresponding register bits. These bits are automatically reset to "0" by the hardware after the commands execute.

  - Reset command - The Reset command is sent to the Plug and Play cards by writing a value of 0x01 to the Config Control register. All Plug and Play cards in any state, except **Wait for Key**, respond to this command. This command performs a reset function on all logical devices. This resets the contents of configuration registers to their default state. The configuration registers for all logical devices are loaded with their power-up values from non-volatile memory or jumpers. The READ_DATA port, CSN and Plug and Play state are preserved.

    Note: The software must delay 1 msec after issueing the reset command before accessing the auto-configuration ports.

  - Wait for Key command - The Wait for Key command is sent to the Plug and Play cards by writing a value of 0x02 to the Config Control register. All Plug and Play cards in any state will respond to this command. This command forces all Plug and Play cards to enter the *Wait for Key* state. The CSNs are preserved and no logical device status is changed.

  - Reset CSN command - The Reset CSN command is sent to the Plug and Play cards by writing a value of 0x04 to the Config Control register. All Plug and Play cards in any state, except **Wait for Key**, will reset their CSN to 0.

    Implementor's note: On a CTRL-ALT-DEL key sequence, the BIOS issues a reset of all logical devices, restores configuration registers to their default values, and returns all cards to the ***Wait for Key*** *state (i.e., write a value of 0x03 to the Config Control register). This retains the CSNs and READ_DATA port and will eliminate the need to go through the isolation sequence again. A write to this register with all three bits set is equivalent to a RESET_DRV event.*

- Set RD_DATA Port command - This command sets the address of the READ_DATA Port for all Plug and Play cards. Write data bits[7:0] is used as ISA I/O bus address bits[09:02]. The ISA bus address bits[1:0] is fixed at binary "11." The ISA bus address bits[15:10] is fixed at binary "000000." This command can only be used in the *Isolation* state. The exact method for setting the read data port is:

  ```
  Issue the Initiation Key
  Send command Wake[0]
  Send command Set RD_DATA Port
  ```

  Note: After a RESET_DRV or Reset CSN command, this register is considered uninitialized and must be reinitialized.

- Serial Isolation register - A read from the Serial Isolation register causes Plug and Play cards in the *Isolation* state to respond to the ISA bus read cycle as described in the Isolation Protocol section above. Cards that "lose" the isolation protocol will enter the *Sleep* state.
- Card Select Number - A Card Select Number is uniquely assigned to each Plug and Play card when the card has been isolated and is the only card in the *Isolation* state. A Card Select Number of zero represents an unidentified card. Valid Card Select Numbers for identified ISA cards range from 1 to 255 and must be assigned sequentially starting from 1. The Card Select Number is used to select a card via the Wake[CSN] command as described above. The Card Select Number on all ISA cards is set to zero on a RESET_DRV or Reset CSN command. The CSN is never set to zero using the CSN register.

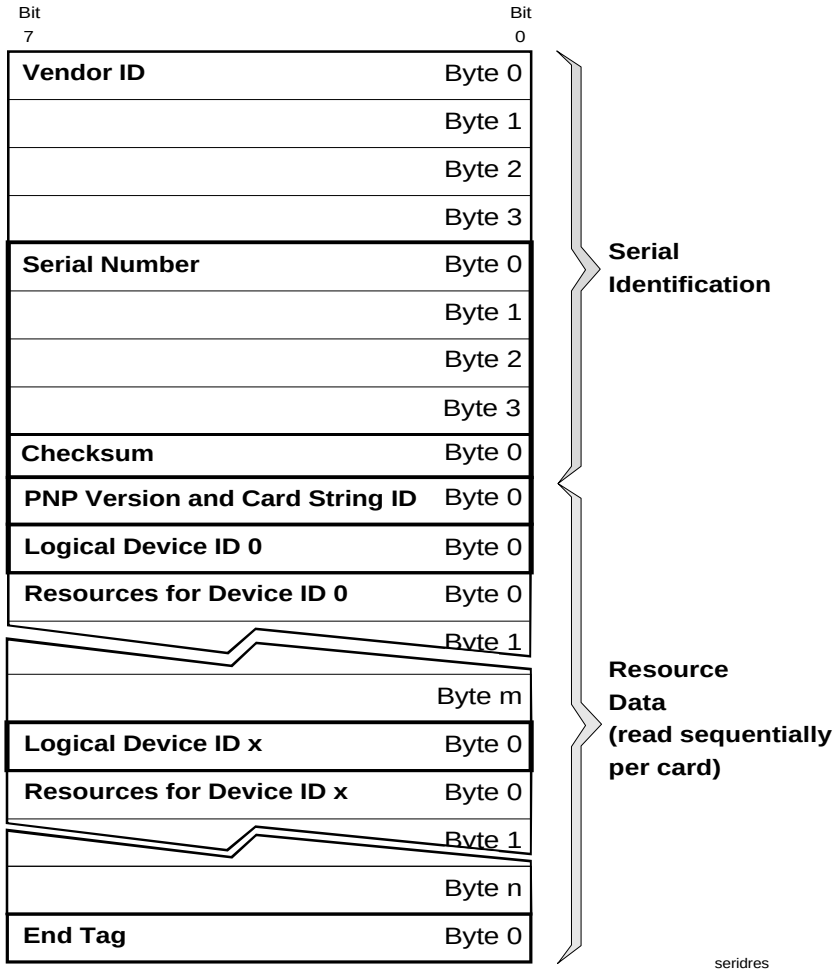

Figure 7. Serial Identifier and Resource Data

- Wake[CSN] command - This command is used to bring ISA cards in the *Sleep* state to either the *Isolation* state or the *Config* state. A Wake[CSN] command with a parameter of zero will force all cards without a CSN to enter the **Isolation** state. A Wake[CSN] command with a parameter other than zero will force a card with a matching CSN to enter the *Config* state. Any card in the *Isolation* or *Config* states that receives a Wake[CSN] command with a parameter that does not match its CSN will transition to the *Sleep* state.

  All Plug and Play cards function as if their 72-bit serial identifier and their resource data come from a single serial device. The pointer to this data is reset to the beginning whenever a card receives any Wake[CSN] command.

- Resource Data register - A read of the Resource Data register will return one byte of resource data from a Plug and Play card in the *Config* state. Resource data is always returned byte sequentially. The Status register must always be read to confirm that resource data is available before the Resource Data register is read.

- Status register - Bit[0] of the Status register indicates that the next byte of resource data is available to be read. If this bit is equal to one then data is available, otherwise resource data is not yet available. The Plug and Play software will poll this location until bit[0] is set, then the next data byte from the Resource Data register is read.

- Logical Device Number register - The logical device number register is used to select which logical device the following configuration commands will operate on. Cards may contain more than one logical device, in which case the logical device is selected by writing the 8-bit logical device number into this register. The logical device number is determined by the order in which the logical devices are read from the resource data. The first logical device number is 0, the second is 1, and so on.

- I/O Range Check register - I/O Range Check register allows the Plug and Play software to determine if another card conflicts with the I/O port range that has been assigned to a logical device. The I/O range check works by having all I/O ranges that would be used by a logical device return 0x55 then 0xAA on I/O read commands. The Plug and Play software performs reads to all the ports that would be used by the logical device and verifies that the correct data is returned. If a conflict is detected, then the Plug and Play software relocates the I/O range of the logical device to a new location. Setting bit[1] of this register enables the I/O check range check logic. Setting bit[0] forces the logical device to respond to I/O reads within its assigned I/O range with the value 0x55. If bit[0] is cleared, then the logical device responds to reads within its assigned I/O range with the value of 0xAA. This function operates only when bit[0] of the Activate register (described below) is not set. This command is optional and is not implemented on cards that do not have configurable I/O port ranges.

- Activate register - The Activate register is a read/write register that is used to activate a logical device. An active logical device responds to all ISA bus cycles as per its normal operation. An inactive logical device does not respond to nor drive any ISA bus signals. Bit[0] is the active bit, if it is set to "1" then the logical device is active, otherwise it is inactive.

4.3. Control Register Summary (cont.)

The state transitions for the Plug and Play ISA card are shown below.

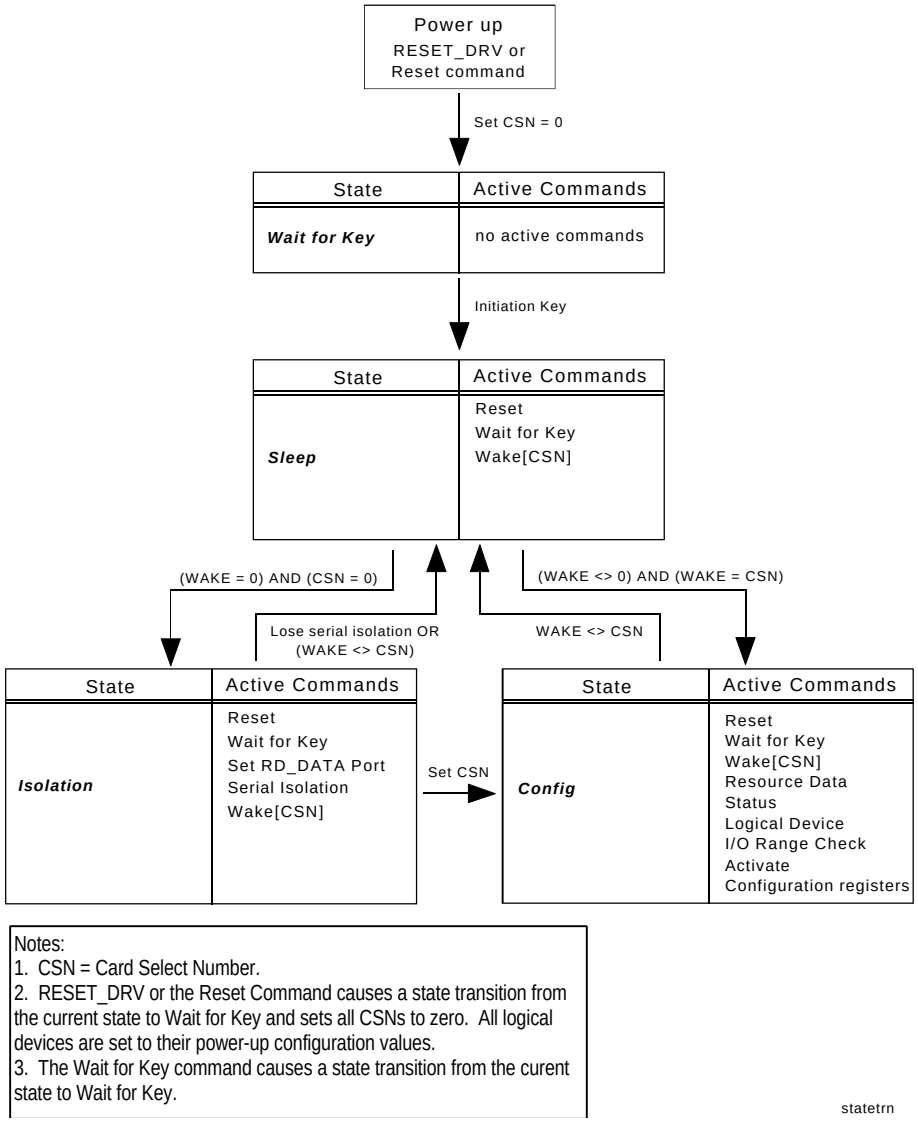

Figure 8. Plug and Play ISA Card State Transitions

4.4. Plug and Play Isolation Sequence

On power up, all Plug and Play cards detect RESET_DRV, set their CSN to 0, and enter the *Wait for Key* state. There is a required 2 msec delay from either a RESET_DRV or ResetCmd to any Plug and Play port access to allow a card to load initial configuration information from a non-volatile device.

Cards in the *Wait for Key* state do not respond to any access to their auto-configuration ports until the initiation key is detected. Cards ignore all ISA accesses to their Plug and Play interface.

When the cards have received the initiation key, they enter the *Sleep* state. In this state, the cards listen for a Wake[CSN] command with the write data set to 0x00. This Wake[CSN] command will send all cards to the *Isolation* state and reset the serial identifier/resource data pointer to the beginning.

The first time the cards enter the *Isolation* state it is necessary to set the READ_DATA port address using the Set RD_DATA port command.

Next, 72 pairs of reads are performed to the Serial Isolation register to isolate a card as described previously. If the checksum read from the card is valid, then this means one card has been isolated. The isolated card remains in the *Isolation* state while all other cards have failed the isolation protocol and have returned to the *Sleep* state. The CSN on this card is set to a unique number. Writing this value causes this card to transition to the *Config* state. Sending a Wake[0] command causes this card to transition back to *Sleep* state and all cards with a CSN value of zero to transition to the *Isolation* state. This entire process is repeated until no Plug and Play cards are detected.

4.5. Reading Resource Data

Card resource data may only be read from cards in the *Config* state. A card may get to the *Config* state by one of two different methods. A card enters the *Config* state in response to the card "winning" the serial isolation protocol and having a CSN assigned. The card also enters the *Config* state in response to receiving a Wake[CSN] command that matches the card's CSN.

As shown in figure 7, all Plug and Play cards function as if their 72-bit serial identifier and their resource data both come from a single byte-serial device. As stated earlier, the pointer to the byte-serial device is reset in response to any Wake[CSN] command. This implies that if a card enters the **Config** state directly from **Sleep** state in response to a Wake[CSN] command, the 9-byte serial identifier must be read first before the card resource data is accessed. The Vendor ID and Unique Serial Number are valid; however, the checksum byte, when read in this way, is not valid. For a card that enters the **Config** state from the **Isolation** state (i.e. after the isolation protocol has been run and all 72 bits of the serial identifier have been read), the first read of the Resource Data register will return resource data.

Card resource data is read by first polling the Status register and waiting for bit[0] to be set. When this bit is set it means that one byte of resource data is ready to be read from the Resource Data register. After the Resource Data register is read, the Status register must be polled before reading the next byte of resource data. This process is repeated until all resource data is read. The format of resource data is described in the following section.

The above operation implies that the hardware is responsible for accumulating 8 bits of data in the Resource Data register. When this operation is complete, the status bit[0] is set. When a read is performed on the Resource Data register, the status bit[0] is cleared, eight more bits are accumulated in the Resource Data register, then the status bit[0] is set again.

4.6. Configuring Card Resource Usage

Plug and Play cards support the following registers which are used for configuring the card's standard ISA resource usage per logical device.

- Memory Address Base registers (up to four non-contiguous ranges)
- I/O Address Base registers (up to eight non-contiguous ranges)
- Interrupt Level Select registers (up to two separate interrupt levels)
- DMA Channel Select registers (up to two DMA channels)

These registers are read/write and always reflect the current operation of all logical devices on the Plug and Play card. If a resource is not programmable, then the configuration register bits are read-only.

4.6.1. Order of Configuration Reads

Resource data is read from each Plug and Play card that describes how many logical devices are on the card as well as the resource requirements for each logical device. Any of the configuration registers that are programmable must be programmed into the logical device through the card's Plug and Play register interface.

Logical device's configuration registers are programmed by first writing the logical device number to the Logical Device Number register. The logical device number is determined by the order in which logical devices are read from the card's resource data. The first logical device read is number 0, the next is number 1, and so on. Logical devices may be programmed in any order.

After the logical device is selected, configuration registers are programmed by writing the proper values to the Plug and Play register interface. The Plug and Play register that must be written for each resource is determined by the order in which each resource is read from the Resource Data. For example, a logical device has the following Resource Data structure:

```
TAG Logical Device
    TAG Memory Descriptor       (becomes memory descriptor 0)
    TAG Memory Descriptor       (becomes memory descriptor 1)
    TAG I/O Descriptor          (becomes I/O descriptor 0)
    TAG Start DF                (dependent function)
        TAG I/O Descriptor      (becomes I/O descriptor 1)
        TAG Int Descriptor      (becomes interrupt descriptor 0)
     TAG Start DF               (next set of dependent resources)
        TAG I/O Descriptor      (I/O descriptor 1)
        TAG Int Descriptor      (Interrupt descriptor 0)
    TAG End DF                  (end of resource dependencies)
TAG END
```

This example shows a logical device that needs two independent memory ranges and one independent I/O range. They are defined as Memory Descriptor 0, Memory Descriptor 1 and I/O Descriptor 0 based on the order they were read from the Resource Data structure. Memory Descriptor 0 is programmed at Plug and Play registers 0x40 - 0x44, Memory Descriptor 1 is programmed at Plug and Play registers 0x48 - 0x4C and I/O Descriptor 0 is programmed at Plug and Play registers 0x60 - 0x61. These resource data formats are fully defined in a following section. It is recommended that all independent descriptors appear before any dependent functions. This may simplify the mapping of resource descriptors to configuration registers in some hardware implementations.

The Resource Data for this logical device next calls out a dependent function of one I/O port range and one interrupt. These become I/O descriptor 1 and interrupt descriptor 0 because they are the second I/O and first interrupt resources read for this logical device. The next dependent function entry still represents I/O descriptor 1 and interrupt descriptor 0.

These configuration registers may be programmed in any order, only the register bindings are defined based on the order they are read from the Resource Data structure.

4.6.2. Resource Programming

Plug and Play cards are programmed by sending the card a Wake[CSN] command with the write data set to the card's CSN. This will force the one card with the matching CSN into the *Config* state and force all other cards into the *Sleep* state. Next, the logical device to be programmed is selected by writing the logical device number to the Logical Device Number register. If the card has only one logical device, this step may be skipped.

Resource configuration for each logical device is programmed into the card using the registers for I/O, memory, IRQ, and DMA selection defined in Appendix A. Each and every resource requested by a logical device must be programmed, even if the resource is not assigned. Each resource type is described below.

- Memory Configuration - Memory space resource use is programmed by writing the memory base addressto the memory base address registers. Next, the memory control is written with the correct 8/16/32 bit memory operation value and the decode select option. If the memory decode option was set to range length, then the range length is written to the memory upper limit/range length registers. If the memory decode option was set to upper limit, then the upper limit memory address is written to the upper limit/range length register. If no memory resource is assigned, the memory base address registers must be set to zero and the upper limit/range length registers must be set to zero.

- I/O Space Configuration - I/O space resource use is programmed by writing the I/O base address[15:0] to the I/O port base address registers. If a logical device indicated it uses 10-bit I/O space decoding, then bits [15:10] of the I/O address are not implemented on the card. If no I/O resource is assigned, the I/O base address registers must be set to zero.

- Interrupt Request Level - The interrupt request level for a logical device is selected by writing the interrupt request level number to the Interrupt Level Select register. This select number represents the number of the interrupt on the ISA bus. The edge/level and high/low active state of the interrupt must be written to the Interrupt Request Type register. If no interrupt is assigned, the Interrupt Level Select register must be set to 0.

  The IRQ2 signal on the ISA bus is routed to IRQ 9 on the 8259 interrupt controller. To select IRQ 2 on the ISA bus, the Interrupt Level Select register must be set to 2, not 9.

- DMA Channel - The DMA channel for a logical device is selected by writing the DMA channel number to the DMA Channel Select register. The select number represents the number of the DRQ/DACK channel on the ISA bus. If no DMA channel is assigned, this register must be set to 4.

The last step in the programming sequence is to set the logical device's activate bit. This forces the logical device to become active on the ISA bus at its assigned resources. When finished programming configuration registers, all cards must be set to the *Wait for Key* state.

4.7. Run Time Access to Plug and Play registers

Read access to Plug and Play configuration is available at all times with no impact to the function(s) on the card. However, write accesses to Plug and Play registers must be done with the full knowledge of the device driver controlling the device and the operating system environment. Even though it is possible to re-assign the CSNs during run time, this is not necessary since CSNs for all Plug and Play cards are assigned during initialization. The only exception to this case is for docking stations, hot-insertion capability or if power management is supported. It is required that prior to changing the value of any registers, the logical device be de-activated, the resource register re-programmed, and the logical device is activated again. When finished accessing Plug and Play registers, all cards must be returned to the *Wait for Key* state.

5. Plug and Play Functionality

5.1. Minimum Functionality for a Configurable Logical Device

The minimum functionality configurable Plug and Play logical device identifies itself through the Plug and Play process. The device also provides complete resource data and must accept configuration commands from Plug and Play software. The configuration registers must be read/write. The logical devices must not use any resources except for those assigned. This means that a device that has previous default resource settings must either use the new resource settings upon being activated or become inactive until it is re-configured in its own proprietary way. Cards must report their actual active status when the Activate register is read.

This minimum function exists to support logical devices that are re-configurable using other, different, methods, such as EISA configuration. In order for this type of device to participate in the Plug and Play process, the configuration of the device must be readable using Plug and Play standard registers. This is done by both reading the device 's resource data structure and reading the device 's configuration registers.

5.2. Minimum Level of Functionality for a Non-configurable Logical Device

The minimum functionality non-configurable Plug and Play logical device identifies itself and provides resource information about the fixed resources it uses. The device should support the Activate command so that it can be removed from the bus if the Plug and Play software detects a non-resolvable conflict with a logical device on this card and another logical device in the system. Resource usage does not need to programmable since it may not be re-located.

This is the minimum functionality level only for a device with fixed compatible resource usage. This is not a desirable solution because there is no guarantee that any given configuration of devices with fixed resources will be conflict free.

6. Plug and Play Resources

Plug and Play cards return read-only configuration information in two formats. The serial identifier is returned bit-wise by the Plug and Play devices in response to reads from the Serial Isolation register. This information is returned in a serial format to facilitate the Plug and Play device selection algorithm described earlier. Plug and Play cards also provide resource data sequentially a byte at a time in response to reads from the Resource Data register. The resource configuration data completely describes all resource needs and options of the device.

6.1. Serial Identifier

The serial identifier of all Plug and Play ISA cards is defined below.

Table 2. Plug and Play Header

Field Name	Length	Definition
Vendor ID Byte 0	8 bits	Bit[7] 0 Bits[6:2] First character in compressed ASCII Bits[1:0] Second character in compressed ASCII bits[4:3]
Vendor ID Byte 1	8 bits	Bits[7:5] Second character in compressed ASCII bits[2:0] Bits[4:0] Third character in compressed ASCII
Vendor ID Byte 2	8 bits	(Vendor Assigned) Bits[7:4] First hexadecimal digit of product number (bit 7 is msb) Bits[3:0] Second hexadecimal digit of product number (bit 3 is msb)
Vendor ID Byte 3	8 bits	(Vendor Assigned) Bits[7:4] Third hexadecimal digit of product nmber (bit 7 is msb) Bits[3:0] Hexadecimal digit of revision level (bit 3 is msb)
Serial/Unique Number Byte 0	8 bits	Unique device number so the system can differentiate between multiple cards of the same type in one system. Bits[7:0]
Serial Number Byte 1	8 bits	Serial Number Bits[15:8]
Serial Number Byte 2	8 bits	Serial Number Bits[23:16]
Serial Number Byte 3	8 bit	Serial Number Bits [31:24]
Checksum	8 bits	Checksum of ID and serial number verifies that the information has been correctly read from a Plug and Play ISA card.

6.1.1. Vendor ID

The 32-bit Vendor ID is an EISA Product Identifier (ID). This ID consists of:

- bits[15:0] - three character compressed ASCII EISA ID.

  Compressed ASCII is defined as 5 bits per character, "00001" = "A" ... "11010" = "Z". This field is assigned to each manufacturer by the EISA administrative agent.

- bits[31:16] - manufacturer specific product number and revision. It is the responsibility of each vendor to select unique values for this field.

The purpose of this field is to serve as a unique board identifier that allows Plug and Play card selection through the isolation algorithm described earlier.

6.1.2. Serial/Unique Number

The 32-bit serial number is used only in the isolation process for selection of individual Plug and Play ISA cards. This number differentiates between multiple cards with the same Vendor ID when they are plugged into one system. This number must be unique in order to support multiple cards with the same Vendor ID in one system. If this feature is not supported then this field must be returned as "FFFFFFFF." Lack of a unique serial number implies that only one instance of a Vendor ID can be supported in a system.

6.1.3. Checksum

The checksum field is used to ensure that no conflicts have occurred while reading the device identifier information. The checksum generation is described in appendix B.

6.2. Plug and Play Resource Data Types

Plug and Play resource data fully describes all resource requirements of a Plug and Play ISA card as well as resource programmability and interdependencies. Plug and Play resource data is supplied as a series of "tagged" data structures. To minimize the amount of storage needed on Plug and Play ISA cards two different data types are supported. These are called small items and large items. The general format of the structure is shown below. The first byte defines the type and size for most information and is followed by one or more bytes of the actual information. Bit[7] of the first byte is used as the tag identifier to differentiate between a small item or a large item data type.

6.2.1. Resource Data Requirements

The minimum level of functionality of Plug and Play cards was defined earlier. This implies that the Plug and Play version number data type (small item format), the identifier string data type (large item format) and all resource types to identify the fixed resources will always be present. The identifier string is used by the operating system for device related configuration and error messages. Card vendors may include several identifiers if they would like to support multiple text formats (ANSI, Unicode) and international languages. The description of an identifier string is included in the section on large item format.

It is recommended that configurable boot devices include all configuration information on their cards to allow Plug and Play BIOS to manage resources during boot time. Refer to the section on boot devices for more information.

A Plug and Play logical device may use any number of resources and any combination of small item or large item data types. The general format is :

1. Plug and Play version number type
2. Identifier string resource type
3. Logical device ID resource type
    - Any compatible device ID resource type for this logical device
    - Resource data types to match what the function uses (IRQ, memory, I/O, DMA)-the order is not important.
    - Any dependent functions needed if the Plug and Play card is configurable. The order of the resource data establishes the binding to the configuration registers.

    (**Note**: Step 3 is repeated for each logical device present on the Plug and Play card.)
4. End tag resource type to indicate the end of resources for this Plug and Play card.

The order of resource descriptors is significant because configuration registers are programmed in the same order that descriptors are read (see section 4.6.1). This may be important in some hardware implementations. Further, in the case of Dependent Functions it may be necessary to include null descriptors ('filler') in order to maintain the desired descriptor-to-register mapping regardless of which Dependent Function is programmed by the software.

6.2.2. Small Resource Data Type

A small resource data type may be 2-8 bytes in size and adheres to the following format:

Table 3. Small Resource Data Type Tag Bit Definitions

Offset	Field		
Byte 0	Tag Bit[7]	Tag Bits[6:3]	Tag Bits [2:0]
	Type = 0	Small item name	Length = n bytes
Bytes 1 to n	Actual information		

The following small information items are currently defined for Plug and Play ISA devices:

Table 4. Small Resource Items

Small Item Name	Value
Plug and Play version number	0x1
Logical device ID	0x2
Compatible device ID	0x3
IRQ format	0x4
DMA format	0x5
Start dependent Function	0x6
End dependent Function	0x7
I/O port descriptor	0x8
Fixed location I/O port descriptor	0x9
Reserved	0xA - 0xD
Vendor defined	0xE
End tag	0xF

6.2.2.1. Plug and Play Version Number

The Plug and Play Version Number identifies the version of the Plug and Play specification with which this card is compatible. A vendor specific version number is included and may be used by a device driver to verify the version of the card.

Offset	Field Name
Byte 0	Value = 00001010B (Type = 0, small item name = 0x1, length = 2)
Byte 1	Plug and Play version number (in packed BCD format, major bits[7:4], minor bits[3:0]) Example: Version 1.0 = 0x10, Version 2.3 = 0x23, Version 2.9 = 0x29
Byte 2	Vendor specific version number

6.2.2.2. Logical Device ID

The Logical Device ID provides a mechanism for uniquely identifying multiple logical devices embedded in a single physical board. The format of the logical device ID is identical to the Vendor ID field (see Table 2, Sec. 6.1).

- bits[15:0] - three character compressed ASCII EISA ID.

  Compressed ASCII is defined as 5 bits per character, "00001" = "A" ... "11010" = "Z". This field must contain a valid EISA ID, although it is not required to have the same 3 letters as the Vendor ID.

- bits[31:16] - manufacturer-specific function number and revision. It is the manufacturer's responsibility to have unique bits[31:16] for different functions.

This identifier may be used to select a device driver for the device. Because of this, Logical Device IDs must be uniquely associated with a specific function. However, there is no need for the Logical Device ID itself to have a unique value, either on a card, or across cards. For instance, a card that implements two communications ports may use the exact same Logical Device ID for both. Similarly, two different products (different Vendor IDs) may both implement the same function, and therefore will use the same Logical Device ID for it. The Logical Device ID is required on all cards. On single-function cards, the Logical Device ID may be the same as the card's vendor ID

The Logical Device ID includes information about what optional commands are supported. Also, bit zero of the flags field is used to indicate that this device should be activated by the BIOS at boot time if this system includes a Plug and Play BIOS. Refer to the section on Plug and Play boot devices (Appendix C, Sec. C-2) for details.

Offset	Field Name
Byte 0	Value = 00010 1xxB (Type = 0, small item name = 0x2, length = (5 or 6))
Byte 1	Logical device ID bits[7:0]
Byte 2	Logical device ID bits[15:8]
Byte 3	Logical device ID bits[23:16]
Byte 4	Logical device ID bits[31:24]
Byte 5	Flags: Bits[7:1], if set, indicate commands supported per logical device for registers in the range of 0x31 to 0x37 respectively. Bit[0], if set, indicates this logical device is capable of participating in the boot process Note: Cards that power-up active MUST have this bit set. However, if this bit is set, the card may or may not power-up active.
Byte 6	Flags: Bit[7:0], if set, indicate commands supported per logical device for registers in the range of 0x38 to 0x3F respectively.

6.2.2.3. Compatible Device ID

The compatible device ID provides the IDs of other devices with which this device is compatible. The operating system uses this information to load compatible device drivers if necessary. There can be several compatible device identifiers for each logical device. The order of these device IDs may be used by the operating system as a criteria for determining which driver should be searched for and loaded first.

Offset	Field Name
Byte 0	Value = 00011100B (Type = 0, small item name = 0x3, length = 4)
Byte 1	Compatible device ID bits[7:0]
Byte 2	Compatible device ID bits[15:8]
Byte 3	Compatible device ID bits[23:16]
Byte 4	Compatible device ID bits[31:24]

As an example of the use of compatible IDs, consider a card vendor who ships a device with logical ID 0xABCD0000. At a later date, this vendor ships a new device with a logical ID 0xABCD0001. This new device is 100% compatible with the old device but also has added functionality. For this device, the vendor could include the Compatible device ID 0xABCD0000. In this case, the exact driver for 0xABCD0001 will be loaded if it can be located. If the driver for 0xABCD0001 can not be found, the driver for device 0xABCD0000 will be loaded for the device.

A list of standard compatible device drivers is available from the Plug and Play Association as file "devids.txt" on the Association's Compuserve forum, PLUGPLAY.

6.2.2.4. IRQ Format

The IRQ data structure indicates that the device uses an interrupt level and supplies a mask with bits set indicating the levels implemented in this device. For standard ISA implementation there are 16 possible interrupt levels so a two byte field is used. This structure is repeated for each separate interrupt level required.

Offset	Field Name
Byte 0	Value = 00100011B (Type = 0, small item name = 0x4, length = (2 or 3))
Byte 1	IRQ mask bits[7:0]. Bit[0] represents IRQ0, bit[1] is IRQ1, and so on.
Byte 2	IRQ mask bits[15:8]. Bit[0] represents IRQ8, bit[1] is IRQ9, and so on.
Byte 3	IRQ Information. Each bit, when set, indicates this device is capable of driving a certain type of interrupt. (Optional--if not included then assume ISA compatible edge sensitive, high true interrupts) Bit[7:4] *Reserved and must be 0* Bit[3] Low true level sensitive Bit[2] High true level sensitive Bit[1] Low true edge sensitive Bit[0] High true edge sensitive (Must be supported for ISA compatibility)

NOTE: *Low true, level sensitive interrupts may be electrically shared, the process of how this might work is beyond the scope of this specification. Only IRQs that exist on the ISA bus connectors are valid.*

6.2.2.5. DMA Format

The DMA data structure indicates that the device uses a DMA channel and supplies a mask with bits set indicating the channels actually implemented in this device. This structure is repeated for each separate channel required.

Offset	Field Name
Byte 0	Value = 00101010B (Type = 0, small item name = 0x5, length = 2)
Byte 1	DMA channel mask bits[7:0]. Bit[0] is channel 0.
Byte 2	Bit[7] *Reserved and must be 0* Bits[6:5] DMA channel speed supported Status 00 Indicates compatibility mode 01 Indicates Type A DMA as described in the EISA Specification 10 Indicates Type B DMA 11 Indicates Type F Bit[4] DMA word mode Status 0 DMA may not execute in count by word mode 1 DMA may execute in count by word mode Bit[3] DMA byte mode status Status 0 DMA may not execute in count by byte mode 1 DMA may execute in count by byte mode Bit[2] Logical device bus master status Status 0 Logical device is not a bus master 1 Logical device is a bus master Bits[1:0] DMA transfer type preference Status 00 8-bit only 01 8- and 16-bit 10 16-bit only 11 *Reserved*

6.2.2.6. Start Dependent Functions

Each logical device requires a set of resources. This set of resources may have interdependencies that need to be expressed to allow arbitration software to make resource allocation decisions about the logical device. Dependent functions are used to express these interdependencies. The data structure definitions for dependent functions are shown here. For a detailed description of the use of dependent functions refer to the next section.

Offset	Field Name
Byte 0	Value = 00110000xB (Type = 0, small item name = 0x6, length =(0 or 1))

Start Dependent Function fields may be of length 0 or 1 bytes. The extra byte is optionally used to denote priority for the resource group following the Start DF tag. If the extra byte is not included, this indicates the dependent function priority is 'acceptable'. If the Priority byte is included, the priorities are defined as:

Value	Definition
0	Good configuration - Highest Priority and preferred configuration
1	Acceptable configuration - Lower Priority but acceptable configuration
2	Sub-optimal configuration - Functional configuration but not optimal
3 - 255	Reserved

Note that if multiple Dependent Functions have the same priority, they are further prioritized by the order in which they appear in the resource data structure. The Dependent Function which appears earliest (nearest the beginning) in the structure has the highest priority, and so on.

6.2.2.7. End Dependent Functions

Offset	Field Name
Byte 0	Value = 00111000B (Type = 0, small item name = 0x7 length =0)

Note that only one End Dependent Function item is allowed per logical device. This enforces the fact that Dependent Functions are not nestable (see section 7.0).

6.2.2.8. I/O Port Descriptor

There are two types of descriptors for I/O ranges. The first descriptor is a full function descriptor for programmable ISA cards. The second descriptor is a minimal descriptor for old ISA cards with fixed I/O requirements that use a 10-bit ISA address decode. The first type descriptor can also be used to describe fixed I/O requirements for ISA cards that require a 16-bit address decode. This is accomplished by setting the range minimum base address and range maximum base address to the same fixed I/O value.

Offset	Field Name	Definition
Byte 0	I/O port descriptor	Value = 01000111B (Type = 0, Small item name = 0x8, Length = 7)
Byte 1	Information	Bits[7:1] are reserved and must be 0 Bit[0], if set, indicates the logical device decodes the full 16 bit ISA address. If bit[0] is not set, this indicates the logical device only decodes ISA address bits[9:0].
Byte 2	Range minimum base address bits[7:0]	Address bits[7:0] of the minimum base I/O address that the card may be configured for.
Byte 3	Range minimum base address bits[15:8]	Address bits[15:8] of the minimum base I/O address that the card may be configured for.
Byte 4	Range maximum base address bits[7:0]	Address bits[7:0] of the maximum base I/O address that the card may be configured for.
Byte 5	Range maximum base address bits[15:8]	Address bits[15:8] of the maximum base I/O address that the card may be configured for.
Byte 6	Base alignment	Alignment for minimum base address, increment in 1 byte blocks.
Byte 7	Range length	The number of contiguous I/O ports requested.

6.2.2.9. Fixed Location I/O Port Descriptor

Offset	Field Name	Definition
Byte 0	Fixed Location I/O port descriptor	Value = 01001011B (Type = 0, Small item name = 0x9, Length = 3)
Byte 1	Range base address bits[7:0]	Address bits[7:0] of the base I/O address that the card may be configured for. This descriptor assumes a 10 bit ISA address decode.
Byte 2	Range base address bits[9:8]	Address bits[9:8] of the base I/O address that the card may be configured for. This descriptor assumes a 10 bit ISA address decode.
Byte 3	Range length	The number of contiguous I/O ports requested.

6.2.2.10. Vendor Defined

The vendor defined resource data type is for vendor use.

Offset	Field Name
Byte 0	Value = 01110xxxB (Type = 0, small item name = 0xE, length = (1-7))
Byte 1 to 7	Vendor defined

6.2.2.11. End Tag

The End tag identifies an end of resource data. Note: If the checksum field is zero, the resource data is treated as if it checksummed properly. Configuration proceeds normally.

Offset	Field Name
Byte 0	Value = 01111001B (Type = 0, small item name = 0xF, length = 1)
Byte 1	Check sum covering all resource data after the serial identifier. This check sum is generated such that adding it to the sum of all the data bytes will produce a zero sum.

6.2.3. Large Resource Data Type

To allow for larger amounts of data to be included in the configuration data structure the large format is shown below. This includes a 16-bit length field allowing up to 64 Kbytes of data.

Table 5. Large Resource Data Type Tag Bit Definitions

Offset	Field Name
Byte 0	Value = 1xxxxxxxB (Type = 1, Large item name = xxxxxxx)
Byte 1	Length of data items bits[7:0]
Byte 2	Length of data items bits[15:8]
Bytes 3 to n	Actual data items

There following large information items are currently defined for Plug and Play ISA devices:

Table 6. Large Resource Items

Large Item Name	Value
Memory range descriptor	0x1
Identifier string (ANSI)	0x2
Identifier string (Unicode)	0x3
Vendor defined	0x4
32-bit memory range descriptor	0x5
32-bit fixed location memory range descriptor	0x6
Reserved	0x7 - 0x7F

6.2.3.1. Memory Range Descriptor

Size	Field Name	Definition
Byte	Memory range descriptor	Value = 10000001B (Type = 1, Large item name = 1)
Byte	Length, bits[7:0]	Value = 00001001B (9)
Byte	Length, bits[15:8]	Value = 00000000B
Byte	Information	This field provides extra information about this memory. Bit[7] *Reserved and must be 0* Bit[6] Memory is an expansion ROM Bit[5] Memory is shadowable. Bits[4:3] Memory control. Status 00 8-bit memory only 01 16-bit memory only 10 8- and 16-bit supported. 11 *Reserved* Bit[2] Support type Status 1 decode supports high address 0 decode supports range length. Bit[1] Cache support type Status 1 read cacheable, write-through 0 non-cacheable. Bit[0] Write status Status 1 writeable 0 non-writeable (ROM)

(continued on next page)

Size	Field Name	Definition
Byte	Range minimum base address bits[7:0]	Address bits[15:8] of the minimum base memory address for which the card may be configured.
Byte	Range minimum base address bits[15:8]	Address bits[23:16] of the minimum base memory address for which the card may be configured
Byte	Range maximum base address bits[7:0]	Address bits[15:8] of the maximum base memory address for which the card may be configured.
Byte	Range maximum base address bits[15:8]	Address bits[23:16] of the maximum base memory address for which the card may be configured
Byte	Base alignment bits[7:0]	This field contains the lower eight bits of the base alignment. The base alignment provides the increment for the minimum base address. (0x0000 = 64 KByte)
Byte	Base alignment bits[15:8]	This field contains the upper eight bits of the base alignment. The base alignment provides the increment for the minimum base address. (0x0000 = 64 KByte)
Byte	Range length bits[7:0]	This field contains the lower eight bits of the memory range length. The range length provides the length of the memory range in 256 byte blocks.
Byte	Range length bits[15:8]	This field contains the upper eight bits of the memory range length. The range length field provides the length of the memory range in 256 byte blocks.

NOTE: Address bits [7:0] of memory base addresses are assumed to be 0.

NOTE: A Memory range descriptor can be used to describe a fixed memory address by setting the range minimum base address and the range maximum base address to the same value.

NOTE: Mixing of 24-bit and 32-bit memory descriptors is not allowed (see section A.3.1).

6.2.3.2. ANSI Identifier String

Size	Field Name	Definition
Byte	Identifier string	Value = 10000010B (Type = 1, Large item name = 2)
Byte	Length, bits[7:0]	Lower eight bits of identifier string length
Byte	Length, bits[15:8]	Upper eight bits of identifier string length
N * bytes	Identifier string	Device description as an ANSI string

The identifier string is 8-bit ANSI. The length of the string is defined in the structure so the string does not need to be zero terminated. Display software will insure the proper termination gets added to the string so that the termination byte does not need to be stored in the card's non-volatile storage. Each card is required to have an identifier string, each logical device may optionally have an identifier string.

6.2.3.3. Unicode Identifier String

Size	Field Name	Definition
Byte	Identifier string	Value = 10000011B (Type = 1, Large item name = 3)
Byte	Length, bits[7:0]	Lower eight bits of length of string plus four
Byte	Length, bits[15:8]	Upper eight bits of length of string plus four
Byte	Country identifier, bits[7:0]	*To be determined*
Byte	Country identifier, bits[15:8]	*To be determined*
N * bytes	Identifier string	Device description characters

Currently, only ANSI identifier strings are defined. The definition for Unicode will be added at a later time.

6.2.3.4. Vendor Defined

The vendor defined resource data type is for vendor use.

Size	Field Name	Definition
Byte	Vendor defined	Value = 10000100B (Type = 1, Large item name = 4)
Byte	Length, bits[7:0]	Lower eight bits of vendor defined data
Byte	Length, bits[15:8]	Upper eight bits of vendor defined data
N * bytes	Vendor Defined	Vendor defined data bytes

6.2.3.5. 32-bit Memory Range Descriptor

Size	Field Name	Definition
Byte	Memory range descriptor	Value = 10000101B (Type = 1, Large item name = 5)
Byte	Length, bits[7:0]	Value = 00010001B (17)
Byte	Length, bits[15:8]	Value = 00000000B
Byte	Information	This field provides extra information about this memory. Bit[7] *Reserved and must be 0* Bit[6] Memory is an expansion ROM Bit[5] Memory is shadowable. Bits[4:3] Memory control. Status 00 8-bit memory only 01 16-bit memory only 10 8- and 16-bit supported. 11 32-bit memory only Bit[2] Support type Status 1 decode supports high address 0 decode supports range length Bit[1] Cache support type Status 1 read cacheable, write-through 0 non-cacheable. Bit[0] Write status Status 1 writeable 0 non-writeable (ROM)

(continued on next page)

Size	Field Name	Definition
Byte	Range minimum base address bits[7:0]	Address bits[7:0] of the minimum base memory address for which the card may be configured.
Byte	Range minimum base address bits[15:8]	Address bits[15:8] of the minimum base memory address for which the card may be configured
Byte	Range minimum base address bits[23:16]	Address bits[23:16] of the minimum base memory address for which the card may be configured.
Byte	Range minimum base address bits[31:24]	Address bits[31:24] of the minimum base memory address for which the card may be configured
Byte	Range maximum base address bits[7:0]	Address bits[7:0] of the maximum base memory address for which the card may be configured.
Byte	Range maximum base address bits[15:8]	Address bits[15:8] of the maximum base memory address for which the card may be configured
Byte	Range maximum base address bits[23:16]	Address bits[23:16] of the maximum base memory address for which the card may be configured.
Byte	Range maximum base address bits[31:24]	Address bits[31:24] of the maximum base memory address for which the card may be configured
Byte	Base alignment bits[7:0]	This field contains Bits[7:0] of the base alignment. The base alignment provides the increment for the minimum base address.
Byte	Base alignment bits[15:8]	This field contains Bits[15:8] of the base alignment. The base alignment provides the increment for the minimum base address.
Byte	Base alignment bits[23:16]	This field contains Bits[23:16] of the base alignment. The base alignment provides the increment for the minimum base address.

(continued on next page)

Size	Field Name	Definition
Byte	Base alignment bits[31:24]	This field contains Bits[31:24] of the base alignment. The base alignment provides the increment for the minimum base address.
Byte	Range length bits[7:0]	This field contains Bits[7:0] of the memory range length. The range length provides the length of the memory range in 1 byte blocks.
Byte	Range length bits[15:8]	This field contains Bits[15:8] of the memory range length. The range length provides the length of the memory range in 1 byte blocks.
Byte	Range length bits[23:16]	This field contains Bits[23:16] of the memory range length. The range length provides the length of the memory range in 1 byte blocks.
Byte	Range length bits[31:24]	This field contains Bits[31:24] of the memory range length. The range length provides the length of the memory range in 1 byte blocks.

NOTE: Mixing of 24-bit and 32-bit memory descriptors is not allowed (see section A.3.1).

6.2.3.6. 32-bit Fixed Location Memory Range Descriptor

Size	Field Name	Definition
Byte	Memory range descriptor	Value = 10000110B (Type = 1, Large item name = 6)
Byte	Length, bits[7:0]	Value = 00001001B (9)
Byte	Length, bits[15:8]	Value = 00000000B
Byte	Information	This field provides extra information about this memory. Bit[7] *Reserved and must be 0* Bit[6] Memory is an expansion ROM Bit[5] Memory is shadowable. Bits[4:3] Memory control. Status 00 8-bit memory only 01 16-bit memory only 10 8- and 16-bit supported. 11 32-bit memory only Bit[2] Support type Status 1 decode supports high address 0 decode supports range length Bit[1] Cache support type Status 1 read cacheable, write-through 0 non-cacheable. Bit[0] Write status Status 1 writeable 0 non-writeable (ROM)

(continued on next page)