IBM *Power Personal Systems* Architecture

PowerPC Reference Platform NVRAM Specification

Document Number: PPS-AR-FW0002

January 22, 1996

Version 0.3
Next Revision Planned: As required

Document Owner:

Raymond J. Pedersen
Dept. WK3
Zip 4357
(512)838-6536
rj@vnet.ibm.com

The responsibility for using the latest version of this document lies with the user of the document. To verify that you have the latest version, contact the document owner.

NVRAM

Document Number: PPS-AR-FW0002

Document Change List

Version 0.3: (January 22, 1996)

1. Several clarifying wording changes made.
2. “PowerPC Examples” corrected.
3. nvram.h (appendix) updated.

1.0 NVRAM

1.1 Introduction

This document describes the NVRAM layout for PowerPC Reference Platform systems. Its purpose is to establish the nomenclature to be used in the global structures and enumerate current usages of these areas. The document should be viewed as the current and extendible definition of NVRAM for Version 1.1 of the PowerPC Reference Platform Specification. The structure is provided as **`nvram.h`** in the appendix of this document. Feedback on this document in the form of corrections or suggestions for improvement should be sent via the Internet to the document owner named on the cover sheet.

There are a number of TBDs in the document. These are place holders for certain products that are still in development and will be filled in, changed or removed as these designs get finalized.

NVRAM consists of a **header** and three major areas:

1. **Global Environment Area (GEArea)** - This area is used to store information that is common to all elements of the system, including firmware and the operating systems.
2. **Operating System Area (OSArea)** - This area is private to the operating system and will not be discussed in any detail in this document. Note that the contents of this area cannot really be guaranteed across boots of the system since it is unpredictable whether the same OS will be booted. NVRAM_MAP.HEADER.LastOS is provided to indicate whether the contents of the OSArea can be considered to be valid.
3. **Configuration Area (ConfigArea)** - This area is used to store configuration data that is required to be maintained across boots of the system.

1.2 Nomenclature

The nomenclature used in this document is based on the P1275 Open Firmware Specification with reasonable assumptions made where P1275 was lacking definition at the time this needed to be resolved. There is no guarantee that this will match the final Open Firmware specifications.

1.2.1 Open Firmware Device Naming Conventions

A device is uniquely identified by its location within the system device tree. This is specified by writing

down the path from the root node of the tree out to the device as follows:

```
/node-name0/node-name1/ ... /node-nameN
```

A node-name is specified by

```
driver-name@unit-address:device-arguments
```

For purposes of this document:

- `driver-name` is an ASCII string giving the name of the node. Allowable names will be listed in this document.
- `unit-address` is the text representation of the physical address of the node with the address space of its parent. The exact form of this is device dependent.
- `device-arguments` is a field of printable characters of arbitrary length giving additional device information. These are also device dependent.

The leading "/" represents the root node which is not explicitly named. The root node is generally the processor.

1.2.2 PowerPC Device Naming

Conventions that will be used throughout the following discussion:

- All strings are lower case ASCII with the exception of the "PNP" in Plug and Play device ids (example: PNP0700).
- All numbers (hexadecimal or otherwise) are lower case ASCII representations of those numbers with leading zeros removed.

A PowerPC device tree is comprised of a set of nodes that may represent either a bus or a device**.** A bus is most completely described by its generating logic, usually referred to as a *bus bridge.* In this document, *devices* are further described as *device controllers* and actual end *devices*.

1.2.2.1 Bus Bridges

Bus bridges take on a form relative to their parent node. The current allowable forms are:

- **pci**@*aaaaaaaa* designates a PCI bus bridge based at root node address *aaaaaaaa*; *aaaaaaaa* is the hexidecimal base address decoded by the bridge controller with leading zeros removed.
- **pci***vvvv,dddd*@dd,f is used to designate a PCI device that is a bus bridge. See "Device Controllers" on page 2. for details.

1.2.2.2 Device Controllers

Device controllers take on a nomenclature that is peculiar to the bus on which it resides.

1.2.2.2.1 PCI Device Controllers
PCI device controllers take the form **pci***vvvv,dddd*@*dd,f*:`device-arguments,` where

- *vvvv* is the PCI configuration space *vendor identifier* for the device.
- *dddd* is the PCI configuration space *device identifier* for the device.
- *dd* is the PCI configuration space *device number* for the device.
- *f* is the PCI configuration space *function number* for the device.
- `device-arguments` are device dependent and are listed in Table 2 on page 3.

1.2.2.2.2 ISA Device Controllers
ISA device controllers take the form *xxxxxxx*@*gggg*:`device-arguments`, where

- *xxxxxxx* is the uncompressed Plug and Play device id.
- *gggg* is the controller's base I/O or memory address.
- `device-arguments` are device dependent and are listed in Table 2 on page 3.

1.2.2.3 Devices

Devices are generally represented by *ssssssss*@`unit-address:device-arguments`, where

- *ssssssss* is a string representing the type of device. The valid choices are **harddisk**, **cdrom**, **floppy**, and **tape**.
- `unit-address` is device dependent and are listed in Table 1 on page 3.
- `device-arguments` are device dependent and are listed in Table 2 on page 3.

Where there is an identity relationship between a controller and its device, the device may not need to be specified directly.

Table 1. unit-address assignments for devices

Device	Type	unit-address
(any)	SCSI	*gg,hh* is a hex number for the SCSI id, SCSI logical unit number.
(any)	IDE	*h* is 0 for the master IDE device, 1 for the slave IDE device.
floppy		a is the number of the floppy device (0-3)
tape		

1.2.2.4 device-arguments

The following `device-arguments` have been defined:

Table 2. device-arguments assignments for devices

Device	Type	device-arguments
controller	SCSI	*ii* is the hex number for the SCSI id of the SCSI controller.
network	Token Ring	*mm* is a hex number for the token ring speed. *mm* = 0x04 means 4 Mbps *mm* = 0x10 means 16 Mbps
network	Ethernet	*mm,nn* *mm* = 0x00 means TPI (10BASE-T Compatible Squelch Level) *mm* = 0x01 means Thin Ethernet (10BASE2) *mm* = 0x02 means Thick Ethernet (10BASE5) (AUI Port) *mm* = 0x03 means TPI (Reduced Squelch Level) *nn* = 0x01 means that the device conforms to 802.3, 0x00 means not conforming.

1.2.3 PowerPC Examples (IBM6070)

1. A PCI device on a root PCI bus:

   - /pci@80000000/pci1000,1@10,0 is the scsi controller (*vvvv* = 1000 = NCR).
   - /pci@80000000/pci1000,1@10,0:0 specifies the initiiator scsi id to be 0.
   - /pci@80000000/pci1000,1@10,0/harddisk@6,0 is the scsi harddisk with id=6.

2. An ISA device on a root PCI bus:

   - /pci@80000000/pci1014,a@b,0/PNP0700@3f0/floppy@0 is the floppy drive (1014 = IBM).

3. A PCI network device:

   - /pci@80000000/pci1022,2000@c,0:0,0 is the ethernet controller. Note that there is no need to specify a separate device.

1.3 Global Environment Area

The Global Environment Area (GEArea) provides an important interface between the firmware and the operating system. It is used for storage of system parameters that must be preserved between boots of the system.

1.3.1 Boot Device

The boot device is set up by the firmware each time the system is booted. This device specification must

not be modified by the operating system. It is designated as:

- **fw-boot-device=**<*device*>
  where <*device*> is the full ASCII string representation of the boot device.

1.3.2 Boot Device List

The boot device list contains up to 4 devices, separated by semicolons, that represents the current boot device sequence; *device1* will be attempted first, *device2* second, and so on. The device list can be read and altered by firmware, firmware utilities, and operating systems. It is designated as:

- **fw-boot-path=**<*device1*>**;**<*device2*>**;**<*device3*>**;**<*device4*>
  where <*deviceN*> is the full ASCII string representation of boot device N

Implementation Note: PowerPC legacy firmware will not use the `device-arguments` in the **fw-boot-path** strings.

1.3.3 SCSI Controller Identifier

A SCSI bus has a default initiator assigned, normally a controller on the bus with id=7. It is necessary for the operating system to be able to modify a controller's id to support a multi-initiator configuration. This string is created by an operating system and may not be altered by the firmware, except though a user-invoked firmware utility designed to deal with this configuration situation. It is designated as:

- **fw-scsicfg=**<*scsidevicecontroller1*>**;**<*scsidevicecontroller2*>**;** ... <*scsidevicecontrollerN*>
  where <*scsidevicecontrollerN*> is the full ASCII string representation of scsi controller N with `device-argument` *ii* specified.

nvram.h Appendix A

```
/* Structure map for NVRAM on PowerPC Reference Platform */

/* All fields are either character/byte strings which are valid either
endian or they are big-endian numbers.  The RESTART structure is in
whatever endian stored it and it is marked as to which it is.  It is
expected that little endian OSs will map the whole NVRAM into a form
suitable for it by flipping the bytes as necessary for the numbers.

There are a number of Date and Time fields which are in RTC format,
big-endian. These are stored in UT (GMT).

For enum's: if given in hex then they are bit significant, i.e. only
one bit is on for each enum.
*/

#ifndef _NVRAM_
#define _NVRAM_

#define NVSIZE 4096       /* size of NVRAM */
#define OSAREASIZE 512    /* size of OSArea space */
#define CONFSIZE 1024     /* guess at size of Configuration space */

/* The SECURITY fields are maintained by the firmware which should
be the only code that writes the fields. Each operating system may
supply a program which displays their values for the user. These fields
really belong in EEPROM so they are not lost on battery problems
and are secure. */

typedef struct _SECURITY {
  unsigned long BootErrCnt;         /* Count of boot password errors */
  unsigned long ConfigErrCnt;       /* Count of config password errors */
  unsigned long BootErrorDT[2];     /* Date&Time from RTC of last error in pw */
  unsigned long ConfigErrorDT[2];   /* Date&Time from RTC of last error in pw */
  unsigned long BootCorrectDT[2];   /* Date&Time from RTC of last correct pw */
  unsigned long ConfigCorrectDT[2]; /* Date&Time from RTC of last correct pw */
  unsigned long BootSetDT[2];       /* Date&Time from RTC of last set of pw */
  unsigned long ConfigSetDT[2];     /* Date&Time from RTC of last set of pw */
  unsigned char Serial[16];         /* Box serial number */
  } SECURITY;

/* There are 2 slots for recording errors before information is lost.
If all the slots are full and another error happens then the most
recent one should be overwritten to preserve the oldest ones which may
have triggered the flurry.  The overrun bit should be set on this one.
The detector of the error should find a free slot and fill in the
log information and set the Pending status.  If it then succeeds in
logging it, it should set the logged status.  If boot sees any bits on
in H type log entries it should ignore the FastBoot request and do a
```

```
more thorough POST.  In particular it should attempt to run
diagnostics on any failing devices.  It should then set the
DiagnosedOK or DiagnosedFail as appropriate.  When OS's start up they
should look at the log entries and try to record them in their
permanent logs.  If they succeed they should clear the status.  */

typedef enum _ERROR_STATUS {
  Clear = 0,                          /* empty entry - already logged */
  Pending = 1,
  DiagnosedOK = 2,
  DiagnosedFail = 3,
  Overrun = 4,
  Logged = 5
  } ERROR_STATUS;

typedef enum _OS_ID {
  Unknown = 0,
  Firmware = 1,
  AIX = 2,
  NT = 3,
  WPOS2 = 4,
  WPX = 5,
  Taligent = 6,
  Solaris = 7,
  Netware = 8,
  USL = 9,
  Low_End_Client = 10,
  SCO = 11
  } OS_ID;

typedef struct _ERROR_LOG {
  unsigned char Status;    /* ERROR_STATUS */
  unsigned char Os;        /* OS_ID */
  unsigned char Type;      /* H for Hardware, S for Software */
  unsigned char Severity;  /* S - severe, E - error */
    /* The severity should be set S if the OS cannot proceed unless
    fixed.  It should be set to E otherwise.  If the boot process sees
    an S severity it should not attempt to boot the operating system
    that caused it but remain in firmware state.  If it runs the
    diagnostics and gets DiagnosedOK, it may proceed.  */

  unsigned long ErrDT[2];  /* Date&Time from RTC */
  unsigned char code[8];   /* detailed classification of error */
  union {
    unsigned char detail[20];
    } data;
  } ERROR_LOG;

typedef enum _BOOT_STATUS {
    BootStarted = 0x01,
    BootFinished = 0x02,
    RestartStarted = 0x04,
    RestartFinished = 0x08,
    PowerFailStarted = 0x10,
    PowerFailFinished = 0x20,
    ProcessorReady = 0x40,
    ProcessorRunning = 0x80,
    ProcessorStart = 0x0100
    } BOOT_STATUS;

typedef struct _RESTART_BLOCK {
    unsigned short Version;
    unsigned short Revision;
    unsigned long BootMasterId;
```

```
    unsigned long ProcessorId;
    volatile unsigned long BootStatus;
    unsigned long CheckSum; /* Checksum of RESTART_BLOCK */
    void * RestartAddress;
    void * SaveAreaAddr;
    unsigned long SaveAreaLength;
  } RESTART_BLOCK;

typedef enum _OSAREA_USAGE {
  Empty = 0,
  Used = 1
  } OSAREA_USAGE;

typedef enum _PM_MODE {
  Suspend = 0x80, /* part of state is in memory */
// d1501  DirtyBit = 0x01,   /* used to decide if push botton needs to be checked */  // f1013
  Hibernate = 0   /* Nothing in memory */
  } PMMode;

typedef struct _HEADER {
  unsigned short Size;    /* NVRAM size in K(1024) */
  unsigned char Version;  /* Structure map different */
  unsigned char Revision; /* Structure map the same -
                             may be new values in old fields
                             in other words old code still works */
  unsigned short Crc1;    /* check sum from beginning of nvram to OSArea */
  unsigned short Crc2;    /* check sum of config */
  unsigned char LastOS;   /* OS_ID */
  unsigned char Endian;   /* B if big endian, L if little endian */
  unsigned char OSAreaUsage; /* OSAREA_USAGE */
  unsigned char PMMode;   /* Shutdown mode */
  RESTART_BLOCK RestartBlock;
  SECURITY Security;
  ERROR_LOG ErrorLog[2];

/* Global Environment information */
  void * GEAddress;
  unsigned long GELength;
  /* Date&Time from RTC of last change to Global Environment */
  unsigned long GELastWriteDT[2];

/* Configuration information */
  void * ConfigAddress;
  unsigned long ConfigLength;
  /* Date&Time from RTC of last change to Configuration */
  unsigned long ConfigLastWriteDT[2];
  unsigned long ConfigCount; /* Count of entries in Configuration */

/* OS dependent temp area */
  void * OSAreaAddress;
  unsigned long OSAreaLength;
  /* Date&Time from RTC of last change to OSAreaArea */
  unsigned long OSAreaLastWriteDT[2];
  } HEADER;

/*
NVRAM has three major large areas - one for storing Global environment
strings, one for storing Plug and Play compressed configuration
packets for devices that cannot be figured out dynamically, and one
for transient OS dependent information.  The space is shared and
dynamically adjustable.  There are address-length pairs that describe
them.  This allows larger NVRAM parts to be used without bumping the
version number.  Functions which add information and do not have room
can attempt to readjust the space balance and move the data as
```

```
necessary.

The global environment variable should be settable from the firmware
and some OSs may require that some of them are set as part of the
install process.  OSs should make efforts to use the same variable
names for similar things to conserve space.  These variables should be
things that are either needed or used by the firmware or things that
are common to a machine no matter what OS the user is running.

See the Plug and Play Spec for a complete definition of the format of
the compressed configuration packets.  Microsoft is in the process of
extending it to cover PCI and PCMCIA type devices.  This information
should be settable by the user from the firmware state when devices are
installed or removed.  OSs may also set this when device drivers are
installed.  They should attempt to respect values already set for a
particular device.  The compressed packets are expanded into the
configuration structure passed in the residual data at boot time along
with other dynamically discovered devices or devices defined in the
FLASH that are part of the planar.

Both the environment variables and the configuration information are
compatible with and required by OpenBoot.

The OSArea area saves information that is only valid until the next boot
of the machine.  It is valid if and only if the LastOS matches the
current OS */

/* Here is the whole map of the NVRAM */
typedef struct _NVRAM_MAP {
  HEADER Header;
  unsigned char GEArea[NVSIZE-CONFSIZE-OSAREASIZE-sizeof(HEADER)];
  unsigned char OSArea[OSAREASIZE];
  unsigned char ConfigArea[CONFSIZE];
  } NVRAM_MAP;

#endif  /* ndef _NVRAM_ */
```