DRAFT WORKING NOTE (PSARC Version)
Interim Bootstrapping and Configuration Strategy for Solaris on PowerPC

Steven Z., SunSoft

August 19, 1994

1. INTRODUCTION

The System Binary Interface for Solaris on PowerPC describes the means by which Solaris on PowerPC expects to accommodate variations in hardware on PReP-compliant platforms using a combination of services provided by Open Firmware (IEEE P1275, mandated for all PReP-compliant systems shipped after June 1, 1995), the SunSoft Kernel Binary Interface (KBI), and the SunSoft Device Driver Interface (DDI). This internal working note describes the means by which SunSoft will support early systems that do not have Open Firmware.

Various components of Solaris on PowerPC, most notably the bootstrap[1], the Solaris kernel, and the Solaris kernel debugger depend on services that must be obtained either from the firmware or from a ‘‘firmware surrogate’’. In Solaris on PowerPC, these services are the client services defined in the IEEE P1275 standard with the bindings specified in the PowerPC ISA binding to Open Firmware. This note describes how those services will be accessed and the means by which Solaris on PowerPC can support systems in the absence of Open Firmware without changes to the components of Solaris that depend on firmware services.

This proposal calls for emulation of a portion of the Open Firmware interface, and requires that drivers be provided for all devices needed during bootstrapping (console/keyboard or asynchronous terminal, and disk(s) and/or network). Although this is a proposal for an interim solution to problems which Open Firmware will solve for PReP-compliant platforms, it also provides a means by which SunSoft can support systems without Open Firmware, albeit with less generality.

NOTE TO PSARC: The deliverable for this project consists of components enabling a partial emulation of Open Firmware sufficient for booting and configuration. It is not part of Solaris, *per se*, but part of an OEM platform adaptation kit that is expected to be licensed in source form. It will consist of a sample implementation, broken into a collection of common functions that most implementations will probably want to use unchanged, and code for a specific platform, the pre-Open Firmware PReP reference

1. The *bootstrap* referred to here is the program loaded by the firmware of the machine, which in turn loads standalone programs, such as the Solaris kernel itself. This program is often called the ‘‘secondary bootstrap’’ or ‘‘secondary booter’’, the primary bootstrap being the firmware that loads the secondary bootstrap.

implementation. An OEM is free to use the interfaces to the common functions as provided, and will probably find it convenient to do so, but may change them as desired. Thus, the interfaces provided are not official interfaces with any level of commitment by SunSoft.

2. OVERVIEW

2.1 Requirements

For bootstrapping, the essential problem is that we need some means of accessing the ‘‘boot devices’’. More specifically, we need:

- Drivers for those devices, and
- An architected interface between the drivers and both the kernel and the secondary bootstrap (as well as the kernel debugger).

For configuration, Solaris on PowerPC (and on other architectures) requires that the devices available on the system, as well as their properties such as physical addresses, be available to the kernel through the bootstrap. Again, this requires:

- That the bootstrap know the configuration, and
- An architected interface for obtaining the configuration information.

2.2 Open Firmware as the Archtectural Solution

Open Firmware provides both the bootstrapping and configuration facilities and defines the interfaces for accessing them.

IEEE P1275 defines a processor independent programming interface with a single entry point. This entry point is passed to client programs when Open Firmware loads (boots) them. To invoke a client service, client programs call into that entry point passing it an array of ‘‘cells’’ containing:

1. a pointer to a string specifying the service desired,
2. the number of input arguments and return values, and
3. the arguments and result pointers.

Solaris on PowerPC will use this interface, with the register conventions established in the PowerPC supplement to IEEE P1275, as the one and only interface between clients and firmware services *on all PowerPC platforms, whether or not Open Firmware is available*.[2] Thus, the bootstrap, the kernel, the kernel debugger, and any other client programs will be independent of how those services are implemented and are

---

2. Thanks to David Kahn for suggesting and tenaciously presenting the arguments for this approach, the wisdom of which seems obvious in retrospect.

independent of, and need not be linked with, the code providing the services.

Systems without Open Firmware will not be required to provide the entire P1275 client interface, or any part that requires a FORTH (FCode) interpreter (such as the ‘‘interpret’’ or ‘‘call-method’’ client interfaces). The required portion of the client interface is specified further below.

2.3 In the Absence of Open Firmware . . .

If Open Firmware is not available on a platform, the platform manufacturer must provide a client interface module (CIF module) containing a client interface handler (CIF handler) that implements the required parts of the client interface. This may be accomplished in either of two ways:

1. A software implementation of Open Firmware adapted to the platform, complete with an FCode interpreter, or
2. A more restricted implementation that provides the essential configuration and device driver services.

For the latter case, SunSoft will provide a framework consisting of a library of platform-independent functions (in source code form) and sample source code templates for constructing the required platform-dependent functions. In either case, the CIF module includes:

- Driver code for all devices involved in bootstrapping. The driver code is normally in FCode proms on systems with Open Firmware, but in the absence of Open Firmware, may either be built into the bootstrap directly or may use call-backs into the system firmware, if provided on the platform.
- Configuration information, at least for all the devices involved in bootstrapping, and properties required in the PowerPC and relevant bus bindings specified in supplements to the Open Firmware specification. This information may be built into the bootstrap itself (least flexibility), may be obtained from NVRAM in a format determined by the platform vendor, or may be determined from reading configuration information from memory on busses that provide it (e.g., PCI).

In either case, the bootstrapping and configuration mechanism works as follows:

- Firmware (the ROM-based primary bootstrap) provided on the platform loads the CIF module into memory. How it locates the CIF module is platform dependent.
- The CIF module constructs the device information tree, then loads the secondary bootstrap, an ELF (Extensible Linking Format) file whose name (by default) is **BOOT.ELF**, from a FAT (MS-DOS) file system.

Regardless of how the drivers are provided or how the configuration information is obtained, the bootstrap and the kernel accesses them through the P1275 client interface.

3. CLIENT INTERFACE CALLS

The client interface is defined in chapter 6 of IEEE Standard P1275. This section specifies the services that must be available. The inputs and outputs of these are described in the P1275 document. The descriptions and comments that follow are slanted towards implementations based on the Solaris template source code and libraries rather than a full Open Firmware implementation; the requirements of the latter should be obvious.

3.1 Client interface

test
: Test for the existence of a service.

3.2 Device tree services

The CIF module must construct a device tree containing, as a minimum, all the properties required by the P1275 specification and the bindings for the PowerPC and any busses on the system. These services allow the caller to traverse the device tree and obtain the system configuration (devices, processors, memory, etc.).

peer
: Tree traversal

child
: Tree traversal

parent
: Tree traversal

instance-to-package
: Needed to convert the ‘‘ihandles’’ of the cpu and mmu nodes, of the standard input and output (the keyboard/display), or other ‘‘open’’ed devices to ‘‘phandles’’ (package handles) that can be passed to the property services to obtain the properties of those devices.

getproplen
: Reading properties

getprop
: Reading properties

nextprop
: Reading properties

canon
: Converts possibly ambiguous device specifiers to a fully qualified pathname.

finddevice
: Converts a device specifier to a phandle.

instance-to-path
: Converts an instance (an identifier such as one returned from the ‘‘open’’ service) to a fully qualified pathname.

package-to-path
: Converts a phandle to a path.

The implementation of all these functions, assuming a standard representation of the device tree itself, is platform independent. SunSoft will provide an implementation, including a function or functions for the bootstrap to use to construct and add nodes to the device tree. SunSoft will also provide sample code for constructing a device tree including all required properties, such as those required by P1275 and the PowerPC Supplement to 1275.

3.3 Device I/O services

open

close

read

write

seek

These services dispatch to the drivers for devices. When adding a device node to the tree, the caller provides a pointer to a structure that contains the open, close, read, and seek functions for the device, which must be provided by the implementation *for all (and only) devices involved in bootstrapping* including the console (keyboard/display), the CD-ROM drive containing the Solaris CD, and the device containing the ‘‘abstraction software’’. For systems supporting network booting, the implementation must provide a network driver.

3.4 Memory services

These services handle virtual and physical memory management.

claim
: This service allocates physical memory and maps it to a specified (or Open Firmware selected) virtual address.

release
: Unmaps virtual memory and frees the underlying physical memory.

The implementation of these functions, assuming a standard representation of the internal memory lists and location of the page table, is platform independent. SunSoft will provide an implementation.

3.5 Control transfer services

The required services are:

boot
: Resets and re-boots the system.

exit
: Ends execution of the client program.

The implementation of these functions is platform dependent. If they cannot be implemented because the hardware/firmware does not support them, the implementation should disable all but one processor (on a multiprocessor system), place the system in big-endian mode, then cause a reset with the Interrupt Prefix (MSR[IP]) set to 1. This should have the effect of rebooting the system, but without passing arguments to the bootstrap.

3.6 Summary

Although Solaris uses Open Firmware client services for bootstrapping and configuration, the OEM is required only to construct the device tree, to provide drivers for the boot devices, and to provide the boot and exit functions. This requires no FORTH or FCode knowledge or experience whatsoever, or even knowledge of the client interface.

4. DISTRIBUTION FORMAT

Installation on a platform not supported by the Solaris on PowerPC CD-ROM requires access to software on both the CD and some other medium containing the OEM-provided ‘‘abstraction software’’ for the platform. There are two requirements that affect the format of the distribution medium that contains the Solaris abstraction software:

- The firmware must know how to locate and load the CIF module on the distribution medium, and
- Solaris must be able to access the PSM and kernel driver and bus nexus modules on the distribution medium.

The Open Firmware supplement for PowerPC requires an implementation to recognize and boot from floppy disks in MS-DOS FAT format, from FAT format partitions on hard drives, or from ISO-9660 format CD-ROM drives[3] and to recognize and load bootstraps that are in ELF or PE format.

Historically, firmware has simply loaded a ‘‘boot block’’ or a number of blocks from an area located in, or specified in, some fixed portion of a disk. For example, firmware on the current PReP reference implementation (described in the PReP

---

3. Implementations are expected to recognize the formats from their contents rather than the type of disk, so that one could partition a floppy as well as a hard drive, or put an ISO-9660 image on a hard drive.

specification) finds the bootstrap image in a special 0x41 FDISK partition, where the partition table is in the first sector of the disk. We believe that, except for Open Firmware conforming to the PowerPC supplement, firmware will not contain file system (e.g., FAT) or file format (ELF or PE) code for locating and loading a bootstrap. While we do not feel that we can specify where and how the boot firmware locates the code to boot on the distribution medium, we must have some file system structure on the medium so that Solaris can locate the PSM and other modules on the distribution disk.

To reconcile the two requirements, Solaris will expect the distribution medium to contain in its active partition (possibly all of a floppy disk) a FAT file system, with the Solaris files in directories parallel to those from which modules are obtained on the Solaris CD (e.g., **usr****kernel****drv**, etc.), with the PSM in **usr****kernel****platforms**\<platform-name>. The only requirement is that the medium contain a FAT file system either occupying the whole disk or in the active partition of a partitioned disk (recognized by 0x55 in byte 510 and 0xAA in byte 511 of the disk). The firmware can use a separate partition for the CIF module, can load it from reserved space not allocated in the partition table, or can recognize a file in any format (possibly just a memory image) with a given name. The function performed by the CIF module are specified, but how the bootstrap and CIF module get loaded into memory is entirely up to the firmware.

5. BOOTSTRAPPING THE INSTALLED SYSTEM

Once the system has been booted from the distribution medium (e.g., floppy disk) and has installed itself, perhaps on a hard disk, there must be some way of bootstrapping the installed system. We would prefer to boot directly from the hard disk and not require loading the CIF module from the distribution medium. For the initial installation, the firmware was assumed to know where to find the bootstrap and how to load it from the distribution medium, and the restrictions on the format of the bootstrap and on how the firmware would find it were minimal. However, we need a means for installing the CIF module on the hard disk in the *firmware-dependent* place where it must reside. This requires us to architect a format that can accommodate simple firmware that does not understand file system formats while not unnecessarily restricting Solaris's structuring of the disk. Looking ahead to the availability of Open Firmware, Solaris requires that hard disks contain an MS-DOS style partition table, a FAT partition, and a Solaris partition. We do not require that the CIF module be a file in the FAT file system or the Solaris partition, which would require the firmware to understand those file system formats, but the firmware must use information in the partition table to locate the CIF module.

The Solaris installation procedure provides a means by which the platform provider can install the bootstrap in the place where the firmware expects to find it. At the end of the installation procedure, Solaris copies files from the FAT file system on the distribution medium to the corresponding places in the Solaris partition on the hard disk. It then executes a program (whose name is TBD, and which may be a shell command file) from the distribution medium, which performs the platform-specific

actions needed to install the CIF module on the hard drive. Its arguments are the file names of the disk containing the installation medium and of the disk on which the root directory of the system has been installed. The program can use the partition table to determine where to place the CIF module, and can modify the partition table as needed (e.g., change a partition type to 0x41, for the current PReP reference implementation).

6. SUMMARY

With the interim mechanism described in this note, kernel loading and startup are much more restricted than with Open Firmware. Instead of a single, generic Solaris bootstrap that uses Open Firmware (FCode) drivers on peripheral cards or in ROM on the system motherboard, we require a CIF module that incorporates drivers for all the devices involved in the bootstrapping process. However, once the firmware has loaded the bootstrap, the loading of the base kernel, and in turn, its loading of the PSM and other Solaris modules, proceeds in the same way as if Open Firmware were available because the essential Open Firmware client services are provided by code in the bootstrap.

This mechanism is transparent to the kernel, and even to the bootstrap. When Open Firmware is available, no changes to any part of the system are required.