

The Simple Firmware Interface

A. Leonard Brown

Intel Open Source Technology Center

len.brown@intel.com

Abstract

The Simple Firmware Interface (SFI) was developed as a lightweight method for platform firmware to communicate with the Operating System.

Intel's upcoming "Moorestown" hand-held platform will be deployed using SFI.

Here we summarize the motivation for SFI, summarize the contents of the SFI specification, and detail choices made in the Linux kernel implementation.

1 Introduction

The SFI project home page is <http://simplefirmware.org>.

This paper starts by briefly summarizing the site's content, including the content of the SFI specification. Then we describe the implementation of SFI on Linux.

For more details, readers are encouraged to look over the specification, to read and participate on sfi-devel@simplefirmware.org, and to review and suggest enhancements to the source code.

2 Motivation

Intel's upcoming Moorestown hand-held platform is the reason that SFI exists. However, SFI is intended to be both general and open, such that it could be re-used for other platforms.

While Moorestown contains an Intel® Atom™ processor and PCI Express®, it does not contain the legacy elements of a system that make it PC compatible, or ACPI¹ compatible.

¹Advanced Configuration & Power Interface, <http://www.acpi.info>

Moorestown cannot run in ACPI mode because its chipset does not include the required ACPI hardware, and it cannot run in legacy mode because the PC-compatible elements of the system simply do not exist.

3 SFI vs. ACPI

System platforms are either "SFI-platforms" supporting SFI firmware tables, or "ACPI-platforms" supporting ACPI tables.

An Operating System (OS) kernel that supports SFI is an "SFI-OS." An OS that supports ACPI is an "ACPI-OS."

An SFI-platform requires an SFI-OS to boot and run optimally. An ACPI-platform requires an ACPI-OS to boot and run optimally.²

A single OS binary can boot and run optimally on both SFI-platforms and ACPI-platforms. It simply includes the capabilities of the SFI-OS and ACPI-OS, making an "ACPI-SFI-OS."

It is conceivable to build an ACPI-SFI-platform, and such a lab prototype is useful for testing. However, it makes little sense to ship such a system as a product. Were an ACPI-SFI-OS to boot on an ACPI-SFI-platform, the SFI-platform support would simply be ignored in favor of the ACPI-platform.

That said, SFI-platforms can provide access to selected ACPI-defined and ACPI-reserved tables. However, extending SFI with ACPI tables does not make the platform into an ACPI-platform.

²ACPI platforms can often also boot in legacy PC mode, but no known SFI platforms are able to boot in legacy PC mode.

4 SFI and UEFI

SFI is agnostic as to whether a platform supports UEFI³ or not.

However, for platforms that choose not to implement UEFI, SFI does define a static “MMAP” table that returns the information defined by UEFI’s GetMemoryMap() API.

5 SFI Tables

SFI tables are simply a data structure in memory populated by system firmware for the benefit of the OS.

5.1 SFI Table Header

All SFI tables share a common table header format shown in Figure 1. The format is a proper sub-set of

Signature (4)
Length (4)
Revision (1)
Checksum (1)
OEMID (6)
OEM Table ID (8)
Table Payload
...

Figure 1: SFI Common Table Format

ACPI’s static table format⁴ and the semantics and use of the fields in SFI is exactly the same as in ACPI.

However, even though they share a similar format, SFI table signatures are entirely independent of ACPI table signatures. Were a future version of the specifications to define a table signature used by the other, they would refer to two entirely different tables, unless explicitly defined to refer to the same table.

Today SFI’s “XSDT” explicitly refers to the exact same XSDT as defined by ACPI. Indeed, the XSDT is the mechanism used by SFI to prevent name-space collisions between SFI and ACPI.

³UEFI, Unified Extensible Firmware Interface, <http://www.uefi.org>

⁴SFI deleted the OEM Revision, Creator ID, and Creator Revision because they had no apparent function.

5.2 SFI System Table (SYST)

The payload of the SFI System Table (SYST) is an array of pointers to other tables.

While the SYST must reside within a fixed memory region, using an array of pointers allows system firmware the flexibility to locate the actual tables and any convenient address.

It is not uncommon, however, for all of the tables shown in Figure 2 to reside on the same physical page of memory.

5.3 SFI CPUS Table

The optional CPUS table is an array of 32-bit Local APIC IDs, enumerating all the logical processors in the system.

5.4 SFI MMAP Table

The optional MMAP table describes the RAM present in the system. It contains memory descriptors as defined in UEFI’s GetMemoryMap() API.

5.5 SFI (IO) APIC Table

The optional APIC table is an array of physical addresses of the IO-APICs in the system.

5.6 SFI FREQ Table

The optional FREQ table describes the available processor frequencies in the system, in addition to the transition latency and the actual control word used for native hardware performance-state control.

The entries in the FREQ table apply to all processors in the system. The table applies to every logical processor in the system. If there are topology dependencies between processors, the OS must discover those via native hardware methods.

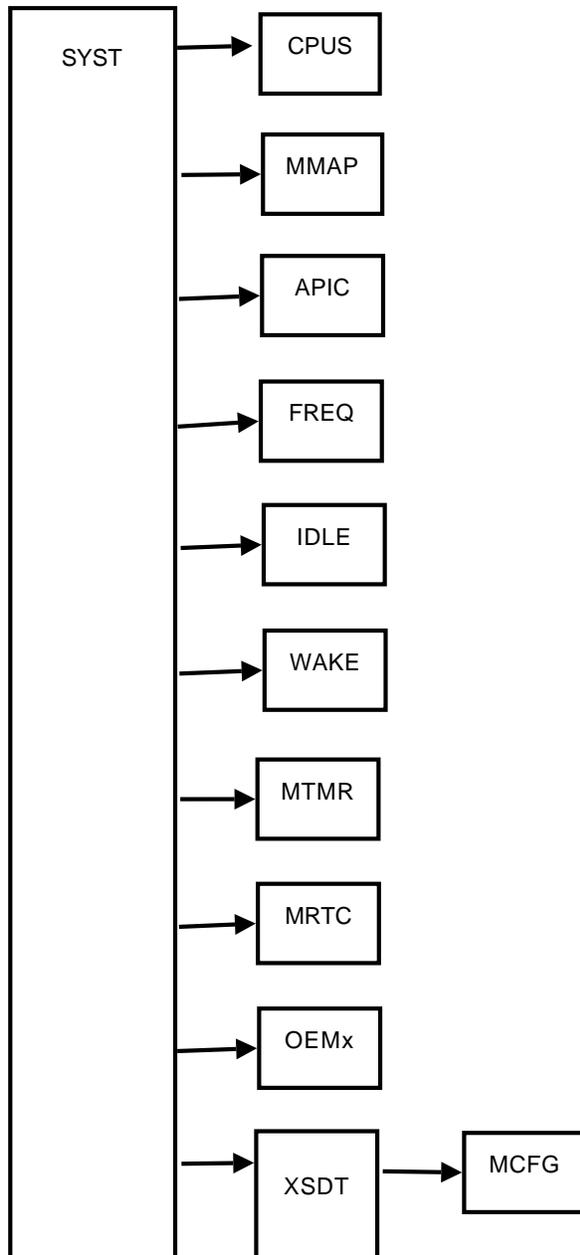


Figure 2: SFI 0.6 table structure

5.7 SFI IDLE Table

The optional Idle Table describes the power saving CPU idle states (e.g., ACPI C-states) available to the OS. These are accessed via the native hardware MWAIT instruction. The IDLE table also enumerates the worst-case exit-latency for each state.

The table applies to every logical processor in the system. If there are topology dependencies between processors, the OS must discover those via native hardware

methods.

5.8 SFI WAKE Table

The optional WAKE vector table contains the 64-bit physical address of the location where the OS writes its resume vector.

5.9 SFI MTMR Table

The optional MTMR table describes the location, frequency, and IRQ of the platform timers present in the Moorestown chip set.

5.10 SFI MRTC Table

The optional MRTC table describes the location and IRQ of the real time clock present in the Moorestown chip set.

5.11 SFI OEMx Table

The optional OEMx table allows OEMs to define vendor-specific SFI tables while avoiding name-space collisions with other platform vendors. The OS and drivers search for tables not only on their base signature, but also using the “6-byte” OEM-id and 8-byte “OEM table id.”

OEMx is intended to mean OEM1, OEM2, OEM3, etc. But the reality is that if a unique OEM-id and OEM-table-id are used in a table search, any arbitrary table signature would work. However, to avoid confusion in the table signature name-space, it is highly encouraged that the OEMx signature be used for vendor specific tables.

5.12 SFI XSDT Table

The optional SFI XSDT is a standard ACPI XSDT. A standard ACPI XSDT can appear in the SYST as a valid SFI table because the SFI table header is a proper subset of the ACPI table header. (SFI simply views the extra ACPI header fields as part of the table body.)

The purpose of the XSDT is to allow SFI to be extended by access to tables and table signatures defined and reserved by the ACPI specification in their standard format. It is not meant to imply that the same system should support both SFI and ACPI at the same time.

5.13 ACPI Tables, and the PCI MCFG

The PCI Memory Configuration Table (MCFG) is defined by the PCI Firmware Specification. It is shown in Figure 2 as an example of a standard ACPI table accessed via SFI.

6 Linux SFI Implementation

The SFI tables can be classified based on when in the boot process they are accessed.

6.1 Early Boot time

First the SYST is located in a reserved region of physical memory. The SYST must be properly aligned and must not cross a 4 KB boundary, which also puts an upper bound on its length.

Linux has several methods to discover the machine's physical memory map, including BIOS e820, UEFI, or boot parameters. If none of those are available, SFI SYST can point to an MMAP table, which must be located and parsed before the MMU is enabled.

6.2 Early OS Initialization

Parts of the kernel will parse SFI tables during the period after the MMU is enabled, but before the OS can set up permanent virtual mappings with `ioremap()`. During this period, the tables are temporarily mapped via `early_ioremap()` for the duration of the parsing routine.

`sfi_init()` is responsible for sanity checking all the SFI tables. It also prints out the table headers to the console.

Linux takes several steps to harden itself against firmware bugs. For a given table signature and version number, it will compare the table length to that listed in the specification before calculating the check-sum.

If any SFI tables fail to check-sum properly, SFI is disabled (and the system will likely not boot).

Linux parses the CPUS and (IO) APIC tables during this period, to enable the processors and interrupts.

6.3 Late OS Initialization

SFI tables can be parsed after the system is up and running and `__init` memory has been freed. Indeed, the main table parsing entry point is exported by the SFI core code such that drivers can parse SFI tables at any time.

6.4 Implementation Choices

In the original prototype, we copied the table headers into a static array in kernel `.data` to make scanning for table signatures fast and compact. However, at Andi Kleen's suggestion, the SFI core no longer copies any tables. Instead they are all parsed in place. The reason is that in the common case, the tables all reside on the same page of memory, so scanning the headers in-place requires no MMU operations and is thus the same speed as doing compares in a data structure optimized for that purpose. Also, most of the tables are scanned at boot and initialization time and never accessed again, so there seems little justification to keep a copy of all the headers around in kernel memory for the up-time of the system.

Of course the driver supplied parsing routine is still free to do whatever it wants with the table, including copying its data into local data structures.

Earlier we mentioned that Linux will sanity check each table signature, version, length, and compute a checksum. However, old versions of Linux must be able to handle tables that are defined by new versions of the specification. Obviously, it can not look up a future table's signature and version number to check its length. So for unknown tables, Linux uses an arbitrary 1 MB length limit before it check-sums a table.

6.5 Source Code

The core SFI patch is about 1,000 lines of code.

This includes the basic SFI table parsers. Drivers that consume SFI tables will provide their own table-specific parsers.

The source code is targeted to go upstream in Linux-2.6.32.

7 Conclusion

The Simple Firmware Interface is indeed simple.

The Linux Kernel patches to implement SFI have been public since late-June. They are currently running on Moorestown hardware, and are expected to be upstream in Linux-2.6.32.

To get involved, please go to the SFI home page, <http://simplefirmware.org>. Review the latest specification, join the mailing list, review and comment on the source code.

8 Acknowledgements

The author thanks Jacob Pan for prototyping the initial Linux SFI support, Feng Tang, for writing most of the final code—and testing it on pre-production hardware—Ingo Molnar, Andi Kleen, and everybody on the lists for their thoughtful code review.

Proceedings of the Linux Symposium

July 13th–17th, 2009
Montreal, Quebec
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium,*
Thin Lines Mountaineering

Programme Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium,*
Thin Lines Mountaineering

James Bottomley, *Novell*

Bdale Garbee, *HP*

Dave Jones, *Red Hat*

Dirk Hohndel, *Intel*

Gerrit Huizenga, *IBM*

Alasdair Kergon, *Red Hat*

Matthew Wilson, *rPath*

Proceedings Committee

Robyn Bergeron

Chris Dukes, *workfrog.com*

Jonas Fonseca

John 'Warthog9' Hawley

With thanks to

John W. Lockhart, *Red Hat*

Authors retain copyright to all submitted papers, but have granted unlimited redistribution rights to all as a condition of submission.