BITS: BIOS Implementation Test Suite

Josh Triplett and Burt Triplett

September 7, 2011

Overview

- BIOS and “the platform”
- Why do we want to test it?
- History of BITS
- Tour of existing functionality
- Fun with scripting in a ring 0, pre-OS environment

BIOS

- Minimal support needed to boot an OS

BIOS

- Minimal support needed to boot an OS
- Platform configuration
- Interface to platform functionality

The “Platform”

- CPUs
- Chipset
- Memory

The “Platform”

- CPUs
- Chipset
- Memory
- Minimal video
- Minimal input

The “Platform”

- CPUs
- Chipset
- Memory
- Minimal video
- Minimal input
- Non-standard stuff: lights, buttons, bells, whistles

Platform configuration

- Highly configurable hardware
- Powers on in a minimal safe configuration

Platform configuration

- Highly configurable hardware
- Powers on in a minimal safe configuration
- Programming CPU and chipset registers
- Tuning for optimal configuration

Platform configuration

- Highly configurable hardware
- Powers on in a minimal safe configuration
- Programming CPU and chipset registers
- Tuning for optimal configuration
- Enabling technologies that require additional configuration

Interface to platform functionality

- 16-bit interrupts

Interface to platform functionality

- 16-bit interrupts
- ACPI: Advanced Configuration and Platform Interface

Interface to platform functionality

- 16-bit interrupts
- ACPI: Advanced Configuration and Platform Interface
- Data structures describing standard components

Interface to platform functionality

- 16-bit interrupts
- ACPI: Advanced Configuration and Platform Interface
- Some data structures describing standard components
- Mostly, bytecode methods to interpret and execute

BIOS has gotten pretty complicated

- Thousands of pages of specifications and recommendations
- Various hardware, standard or system-specific
- A few decades of compatibility requirements
- A tiny, bare-metal programming environment
- No huge community of developers looking at it

What can go wrong?

- Crashes (ASPM)

What can go wrong?

- Crashes (ASPM)
- Broken CPU features (VT, NX, AES)

What can go wrong?

- Crashes (ASPM)
- Broken CPU features (VT, NX, AES)
- Sub-optimal power management (configuration, ACPI)

What can go wrong?

- Crashes (ASPM)
- Broken CPU features (VT, NX, AES)
- Sub-optimal power management (configuration, ACPI)
- Delays and latency (SMI)

What can go wrong?

- Crashes (ASPM)
- Broken CPU features (VT, NX, AES)
- Sub-optimal power management (configuration, ACPI)
- Delays and latency (SMI)
- General-purpose misbehavior (USB, performance counters)

Why might you want BITS?

- You develop a BIOS, and you want a better test criteria than "Windows boots, ship it"
- You hack OS or application code that relies on platform technologies
- You do bug triage, and want a bug reporter to check if the problem lies in their BIOS
- You want to play with hardware in a low-level way, but in a comfortable environment

“That’s why we wrote BITS”

- That’s the problem we wanted to solve
- That’s what BITS evolved into

“That’s why we wrote BITS”

- That’s the problem we wanted to solve
- That’s what BITS evolved into
- That’s not where we started

Testing reference code

- Initializes CPU power management registers
- Writes ACPI tables
- Supports frequency scaling, idling, and throttling

Testing reference code

- Initializes CPU power management registers
- Writes ACPI tables
- Supports frequency scaling, idling, and throttling
- Designed to run in a BIOS
- SMP, takes over CPUs
- How do you test it, without a custom BIOS?

Testing reference code

- Initializes CPU power management registers
- Writes ACPI tables
- Supports frequency scaling, idling, and throttling
- Designed to run in a BIOS
- SMP, takes over CPUs
- How do you test it, without a custom BIOS?
- DOS test harness
- 32-bit DOS extender
- Load and run the reference code

Booting an OS afterwards

- Rewrite the ACPI tables correctly
- See how the OS reacts
- Measure power consumption

Booting an OS afterwards

- Rewrite the ACPI tables correctly
- See how the OS reacts
- Measure power consumption
- BIOS interrupt 19H: load and boot an OS
- Read the MBR and jump to it
- Ends up back in the bootloader

Booting an OS afterwards

- Rewrite the ACPI tables correctly
- See how the OS reacts
- Measure power consumption
- BIOS interrupt 19H: load and boot an OS
- Read the MBR and jump to it
- Ends up back in the bootloader
- No OS ever does this
- Guess how consistently it works?

GNU GRUB2

- 32-bit flat address space
- C, malloc, printf
- File input
- Command line, argument parsing
- Menu system

GNU GRUB2

- 32-bit flat address space
- C, malloc, printf
- File input
- Command line, argument parsing
- Menu system
- DOS-like single thread of control
- No OS to disturb

GNU GRUB2

- 32-bit flat address space
- C, malloc, printf
- File input
- Command line, argument parsing
- Menu system
- DOS-like single thread of control
- No OS to disturb
- . . . and it's a bootloader

BITS on GRUB2

- Run power-management reference code
- Boot an OS

BITS on GRUB2

- Run power-management reference code
- Boot an OS
- Added SMP support to GRUB (`smp_call_function`)
- Implemented various new GRUB commands in C

BITS on GRUB2

- Run power-management reference code
- Boot an OS
- Added SMP support to GRUB (`smp_call_function`)
- Implemented various new GRUB commands in C
- Nice exploratory environment via menus and command line

BITS on GRUB2

- Run power-management reference code
- Boot an OS
- Added SMP support to GRUB (`smp_call_function`)
- Implemented various new GRUB commands in C
- Nice exploratory environment via menus and command line
- Beginnings of a test suite

Early testsuite functionality

```
menuentry "Power management test suite ..." {
...
test_msr_consistency "Max non-turbo ratio" \
    0xCE --mask=0xff00
test_pci "Bus master disable" \
    0 31 0 0xA9 --bytes=1 --shift=2 --mask=1 1
test_msr "C1 Auto Demotion Enable" \
    0xe2 --shift=26 --mask=1 1
...
test_summary
```

Expressiveness

- Based on GRUB2's scripting language, "bashish"
- No real calculation besides `--shift` and `--mask`
- Shell-like conditionals: `if [ $x -lt $y -a ... ]; then`
- Shell-like quoting rules (magic characters, but no magic)
- All non-trivial functionality required C
- Configuration files just glued commands together into menus

C expression parsing

- Evaluate command-line arguments as a C expression
- Store results of other commands in environment

```
cpuid32 --cpu=0 --env --quiet 1
c signature = eax "&" ~ 0xf
if c signature == 0x106a0 ; then
    set cpufamily=nhm
    ...
```

- 64-bit integers only, but that's 90% of what we needed

What about ACPI?

- We could test platform configuration, but what about platform interfaces?

What about ACPI?

- We could test platform configuration, but what about platform interfaces?
- Do tables contain the right data in the right structure?
- Do methods do the right thing, and return the right results?

What about ACPI?

- We could test platform configuration, but what about platform interfaces?
- Do tables contain the right data in the right structure?
- Do methods do the right thing, and return the right results?
- Hand-parsing ACPI is a bad idea

ACPICA

- Portable implementation of ACPI
- Already used by Linux and other OSes
- Bytecode parser, interpreter
- C API
- OS-specific interface layer

ACPICA

- Portable implementation of ACPI
- Already used by Linux and other OSes
- Bytecode parser, interpreter
- C API
- OS-specific interface layer
- Ported to GRUB2 in April

ACPI testing

- Find and parse tables
- Execute methods
- Display and check results

Scripting, again

- Lack of decent scripting becoming a serious problem
- ACPI test functions written entirely in C
- No sensible way to drive from shell scripting
- Doesn't allow exploration from the command line

Scripting, again

- Lack of decent scripting becoming a serious problem
- ACPI test functions written entirely in C
- No sensible way to drive from shell scripting
- Doesn't allow exploration from the command line
- Nobody other than us would ever write tests

Python

- Ported CPython 2.7 to GRUB in May
- Wrote a C/POSIX compatibility layer
- Floating-point support via “fdlibm”
- Ported much of the Python standard library
- Added “bits” and “acpi” modules

Scripting problems: solved!

- Lists, dictionaries, tuples, strings, bignums, floats
- Sorting, searching, comparisons, math
- Printing and formatting

Scripting problems: solved!

- Lists, dictionaries, tuples, strings, bignums, floats
- Sorting, searching, comparisons, math
- Printing and formatting
- Low-level functions in C, logic in Python

Scripting problems: solved!

- Lists, dictionaries, tuples, strings, bignums, floats
- Sorting, searching, comparisons, math
- Printing and formatting
- Low-level functions in C, logic in Python
- CPU and chipset registers
- PCI
- ACPI method evaluation and value decoding

Python scripting sample

Run an ACPI method on every CPU; collect the unique values and corresponding CPUs:

```
for cpupath in cpupaths:
    value = acpi.evaluate(cpupath + "." + method)
    uniques.setdefault(value, []).append(cpupath)
...
```

Implementing GRUB commands in Python

- We still need the GRUB command-line tools
- Useful for exploration and compatibility
- We don't want to write them in C

Implementing GRUB commands in Python

- We still need the GRUB command-line tools
- Useful for exploration and compatibility
- We don't want to write them in C
- Added GRUB commands implemented via Python callbacks
- Deleted a pile of C code

Logging test results

- GRUB has no file write support
- Testsuites print results on the screen
- A record would be nice

Logging test results

- GRUB has no file write support
- Testsuites print results on the screen
- A record would be nice
- Much like a kernel panic: got a camera?
- Serial port, remote KVM. . .

Hammer, meet nail

- We can write data to memory
- GRUB can reserve memory so the OS doesn't overwrite it
- Write special-case code, and read data from `/dev/mem`?

Hammer, meet nail

- We can write data to memory
- GRUB can reserve memory so the OS doesn't overwrite it
- Write special-case code, and read data from `/dev/mem`?
- Linux knows how to read ACPI tables
- GRUB knows how to write them

Hammer, meet nail

- We can write data to memory
- GRUB can reserve memory so the OS doesn't overwrite it
- Write special-case code, and read data from `/dev/mem`?
- Linux knows how to read ACPI tables
- GRUB knows how to write them
- GRUB only provides a command-line `acpi` command

Hammer, meet nail

- We can write data to memory
- GRUB can reserve memory so the OS doesn't overwrite it
- Write special-case code, and read data from `/dev/mem`?
- Linux knows how to read ACPI tables
- GRUB knows how to write them
- GRUB only provides a command-line `acpi` command
- `acpi` reads the ACPI table from a file

FUSE for Python and GRUB

- GRUB has devices like (hd0,0)
- We added a (python) device in August
- Reading (python)/foo invokes a Python callback

FUSE for Python and GRUB

- GRUB has devices like (hd0,0)
- We added a (python) device in August
- Reading (python)/foo invokes a Python callback
- Copy Python output to an internal log
- `grub> acpi (python)/acpilog`

FUSE for Python and GRUB

- GRUB has devices like (hd0,0)
- We added a (python) device in August
- Reading (python)/foo invokes a Python callback
- Copy Python output to an internal log

-
  ```
  grub> acpi (python)/acpilog
  ```

-
  ```
  linux# dd if=/sys/firmware/acpi/tables/BITS
            bs=1 skip=36 of=bits.log
  ```

Fun with writable files

- `configfile (python)/dynamic-menu.cfg`

Fun with writable files

- `configfile (python)/dynamic-menu.cfg`
- `initrd (python)/initramfs.cpio`

Current status of BITS

- Framework for testing, configuration, and exploration
- ACPI method evaluation
- Python scripting in a ring 0, pre-OS environment
- Test suites in areas of our expertise
  - Power management configuration
  - P-state ratios
  - C-state residency
  - CPU configuration registers
  - SMI frequency/latency and real-time response
- Converting tests and commands to Python

Current status of BITS

- Framework for testing, configuration, and exploration
- ACPI method evaluation
- Python scripting in a ring 0, pre-OS environment
- Test suites in areas of our expertise
  - Power management configuration
  - P-state ratios
  - C-state residency
  - CPU configuration registers
  - SMI frequency/latency and real-time response
- Converting tests and commands to Python
- Used by BIOS developers before shipping boards
- BIOS problems actually get fixed!

BITS needs you!

- We have a long list of requested tests

BITS needs you!

- We have a long list of requested tests
- We have two developers

BITS needs you!

- We have a long list of requested tests
- We have two developers
- What platform functionality do you care about?
- What bugs have you observed?
- What do you want to make sure new BIOSes get right?

BITS needs you!

- We have a long list of requested tests
- We have two developers
- What platform functionality do you care about?
- What bugs have you observed?
- What do you want to make sure new BIOSes get right?
- We can help!

BITS needs you!

- We have a long list of requested tests
- We have two developers
- What platform functionality do you care about?
- What bugs have you observed?
- What do you want to make sure new BIOSes get right?
- We can help!
- Come play with low-level functionality in high-level Python

For more information

- http://biosbits.org
- Download an .iso and play
- Download the code and hack
- Drop us an email

Questions?