Fastboot Techniques for x86 Architectures

Marcus Bortel
Field Application Engineer
QNX Software Systems
Introduction

BIOS and BIOS boot time

Fastboot versus BIOS?

Fastboot time

Customizing the boot process

Fastbooting with a BIOS

Demo: Advantech ??? fastboot

Conclusion
Introduction

- **x86 architecture:** typically associated with industrial PCs, desktops, servers, and other PC configurations

- **Traditional barriers to using x86 in deeply embedded applications:**
  - Power consumption – recent developments have mitigated this
  - Long boot times due to reliance on BIOS

- **x86 systems rely on boot firmware called the Basic Input Output System (BIOS):**
  - Provides a wide range of device compatibility
  - Detects the system configuration at boot time
  - Not necessary for embedded systems since hardware platform is pre-defined

- **Fastboot replaces BIOS boot for very fast board startup**
BIOS Boot Time

- The BIOS: a mini OS responsible for interfacing with hardware
- BIOS boot code has the greatest effect on boot time
  - Executes on power up; detects and initializes attached devices
  - Devices include keyboard, mouse, interrupt controllers, video card, CPU, cache, RAM, PCI devices, USB devices, sensors, disk drives
  - Besides initialization, the BIOS must also handle
    - Variations in quantity (e.g. RAM) or features (e.g. CPU support for 64-bit addressing)
    - Execute boot code extension for each device with a BIOS-expansion ROM
  - Completion time is device dependent; varies with configuration
- The Good: booting from the BIOS allows a system design to support a wide range of hardware configurations and devices
- The Bad: Flexibility comes at the cost of slower boot time
Fastboot Time

- Fastboot is fast mostly because of the things that it doesn’t do:
  - Eliminates the need to support a variable hardware configuration
  - Eliminates need to scan devices and do on-the-fly interrupt assignments
  - Doesn’t need to consider processor features or variable RAM

- Developer can make fastboot even faster:
  - Boot from a linearly addressed storage device (EEPROM or Flash)
  - Defer resource allocation

- Fastboot can support “mini drivers”
  - Mini-divers allow communication with devices before system is fully running
  - Example: supports the need to signal a device on a shared bus within a small time limit
### BIOS Boot Time versus Fastboot Time

**Clock settle time**

**Power On Self Test**

- **Graphics HW / Startup Msg**

- **Device Inventory / PNP**

<table>
<thead>
<tr>
<th>Initial Power On Test</th>
<th>Startup Message</th>
<th>Device discovery, initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ms</td>
<td>&gt; 100 ms</td>
<td>200 – 500 ms</td>
</tr>
</tbody>
</table>

**Fast Boot**

- **Boot loader initialized**

- **Mini driver**

- **Boot loader running**

- **Kernel initialized**

- **Full driver**

- **Message buffer**

- **IO Requests**

<table>
<thead>
<tr>
<th>Initial Power On Code</th>
<th>Copy OS Image, Initialize OS, Handle IO Requests</th>
<th>OS Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ms</td>
<td>≤ 50 ms</td>
<td>200 – 500 ms</td>
</tr>
</tbody>
</table>

**Other Drivers and Applications**

**Message buffer**

**IO Requests**

**Boot loader initialized**

**Mini driver**

**Boot loader running**

**Kernel initialized**

**Full driver**

**Message buffer**

**IO Requests**

**Boot loader initialized**

**Mini driver**

**Boot loader running**

**Kernel initialized**

**Full driver**

**Message buffer**

**IO Requests**

**Other Drivers and Applications**
Customizing the Boot Process

- In a fastboot configuration, the developer is responsible for routing interrupts
  - Hard-code interrupt assignments so they are known at compile time
  - Assign interrupts to improve performance
    - Dedicate channels to high-traffic interrupts and share interrupt lines among low-traffic interrupts
    - Sharing IRQs incurs a run-time cost

- Deferring PCI resource allocation
  - Both memory and I/O resources can be allocated after boot; the resource requirements are obtained by probing the PCI device.
  - PCI interrupts, like ISA interrupts, are configured by the developer so that the interrupt assignments are known at compile time

- Booting from flash or EEPROM helps to improve boot times
  - Reduces the complexity of the boot code
  - Avoids slower BIOS calls required to read an image from disk
The Fastboot Process

- **Initial Program Loader (IPL) phase**
  - Handles basic initialization, locates OS image, copies Startup portion of the OS image, and jumps to it

- **Startup phase**
  - Finishes hardware initialization, copies rest of the OS image to RAM, and populates OS data structures

- Developer must now perform steps that the BIOS previously handled
  - Advantage: Developer can choose what needs to be done
  - Disadvantage: Requires developer time and effort
Fastbooting with a BIOS

- Fully customized boot loader means development time and money
  - Initialize the memory controller, CPU, and cache; enable access to high memory; and enter protected mode.
  - For this initialization fastboot loader doesn’t perform these steps any faster than a BIOS

- Researchers at Intel have investigated some ways to reduce boot times of BIOS-based booting:
  - Paper entitled “Fastboot BIOS” lists several potential optimizations\(^1\)
  - Recommendations include bypassing device inventory and hard-coding device initialization, interrupts, etc.
  - Similar to Fastboot process and may be sufficient
  - Strikes a balance between reduced boot times and developer effort

A fastboot-based system will boot significantly faster than a BIOS-based system

BIOS isn’t needed in embedded applications based on platforms that use Intel Atom

- Discarding the BIOS requires developer effort
- For a visual demonstration see YouTube video:
  QNX fast boot on Intel Atom / Advantech SOM-6760
  http://www.youtube.com/results?search_query=qnx+advantech&aq=f
Q&A
Marcus Bortel
MBortel@qnx.com