Porting the Linux Kernel to X86 MID Platforms

April, 2010

Jacob Pan

Intel Open Source Technology Center

Embedded Linux Conference 2010

Agenda

Why porting is needed? Our goals
- X86 PC compatibility
- Moorestown as an embedded platform

The challenges
- Booting
- Device enumeration
- Interrupts
- Timers

The Changes and solutions
- Simple Firmware Interface (SFI) specification
- PCI shim
- IOAPIC emulation
- Abstractions, x86_init, legacy_pic, etc.

1978

2010

X86 PC compatibility

Hardware
- x86 core
- North/South bridges, legacy devices (still needed for booting)

Firmware
- BIOS

Peripheral bus
- PCI/PCI-E

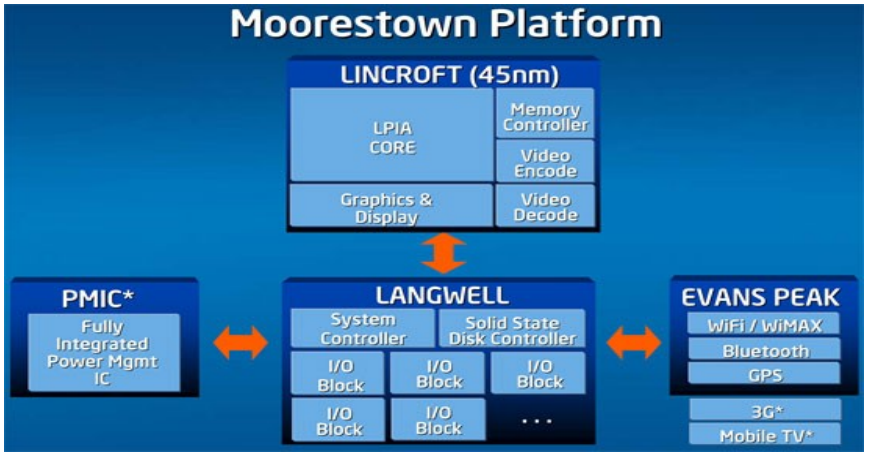
Abstraction interface standard
- ACPI

Introduction to Moorestown

Moorestown is Intel's low power IA based Mobile Internet Device (MID) platform

A two chip solution (maybe more):

- Lincroft: the CPU complex with integrated graphics and memory controller
- Langwell; I/O hub with peripherals such as SPI, I2C, SDIO/MMC, USB host, USB OTG
- Power Management IC (PMIC) for power delivery control, mixed signal, GPIO, audio



What remains compatible with a PC?

- Atom based X86 CPU core
- PCI (partially emulated)
- APIC (IOAPIC partially emulated)

What is new to Moorestown MID?

Moorestown deviates from x86 PC platform

- No real mode in CPU core
- No BIOS, i.e. int calls, SMI/SCI
- No ACPI
- No legacy devices, e.g. i8259 PIC, 8254 PIT
- No PCI on Langwell (I/O hub)
- No real IOAPIC
- No PC compatible platform timers, i.e. HPET, PM timers, PIT, real RTC
- No I/O ports

Kernel enabling challenges

The goal: maintain arch/x86 as a unified platform

- kernel$ tree -d arch/arm/ -L 1 | wc -l 86
- On x86, xen, lguest, and the rest.

Booting

- No existing boot loaders to use due to lack of BIOS and real-mode

Device enumeration

- On chip devices, PCI, statically I/O mapped
- Off chip, non-probable devices

Interrupt routing and delivery

- Utilizing local and IO APIC

Timers

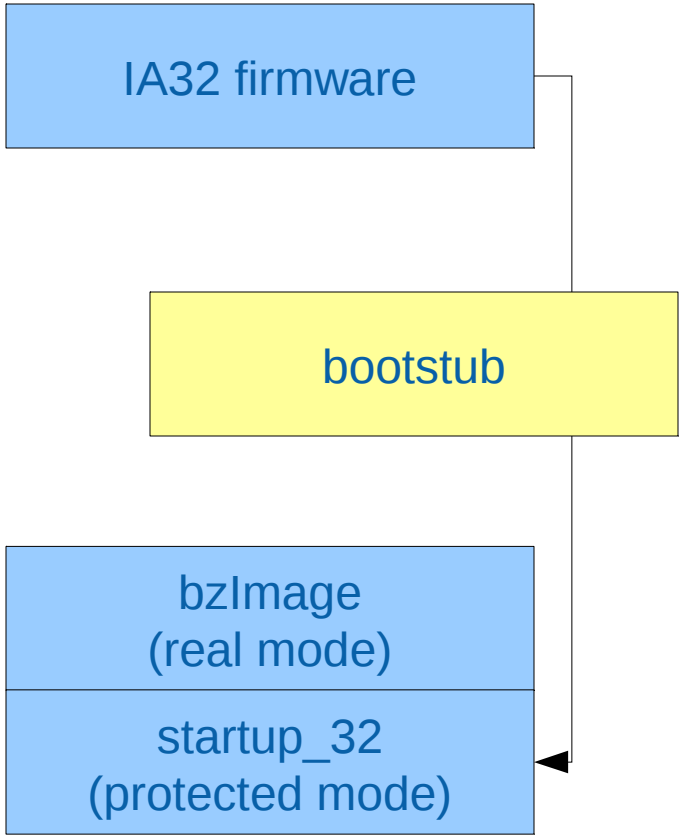
- Use legacy replacement timers

Make it to boot

Firmware hands off to Linux kernel in protected mode.

A small loader is added to

- Bridge between FW and bzImage p-mode entry condition
- Setup boot parameters required by the x86 boot protocol
  - E820 memory map
  - Hardware sub-architecture ID (newly added for Moorestown)



Passing boot information to the kernel

Existing methods on x86 PC include ACPI, BIOS tables (EBDA, MP table), PCI configuration space, DMI, etc.

Introduce Simple Firmware Interface (SFI): a method for platform firmware to export static tables to the operating system (OS).

Stored between 0x000E0000

and 0x000FFFFF

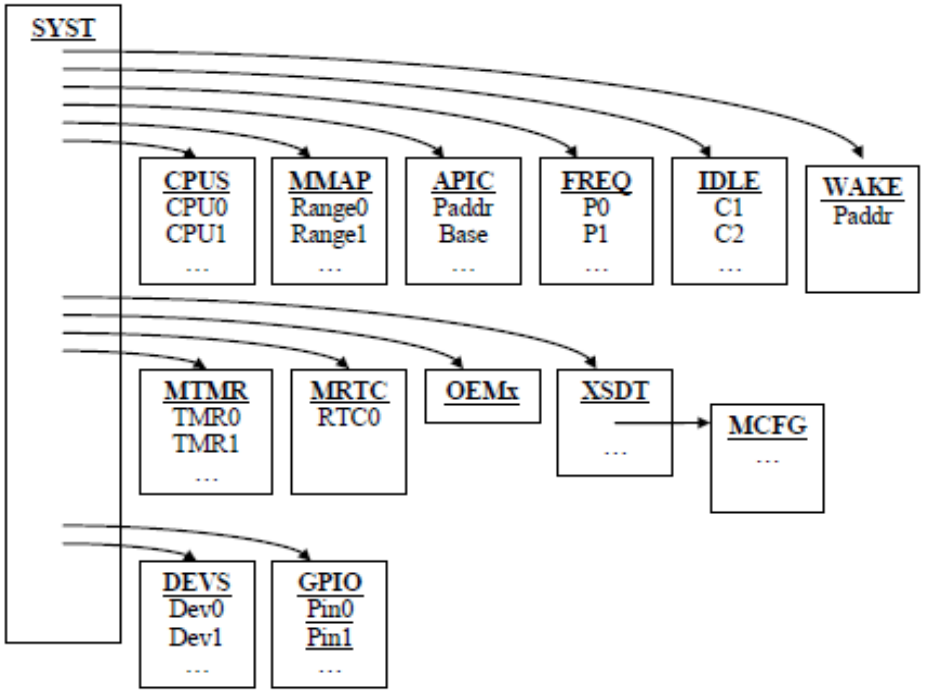


Figure 1 SFI Table Structure

Device Enumeration Methods

System devices (system timers)

- SFI tables

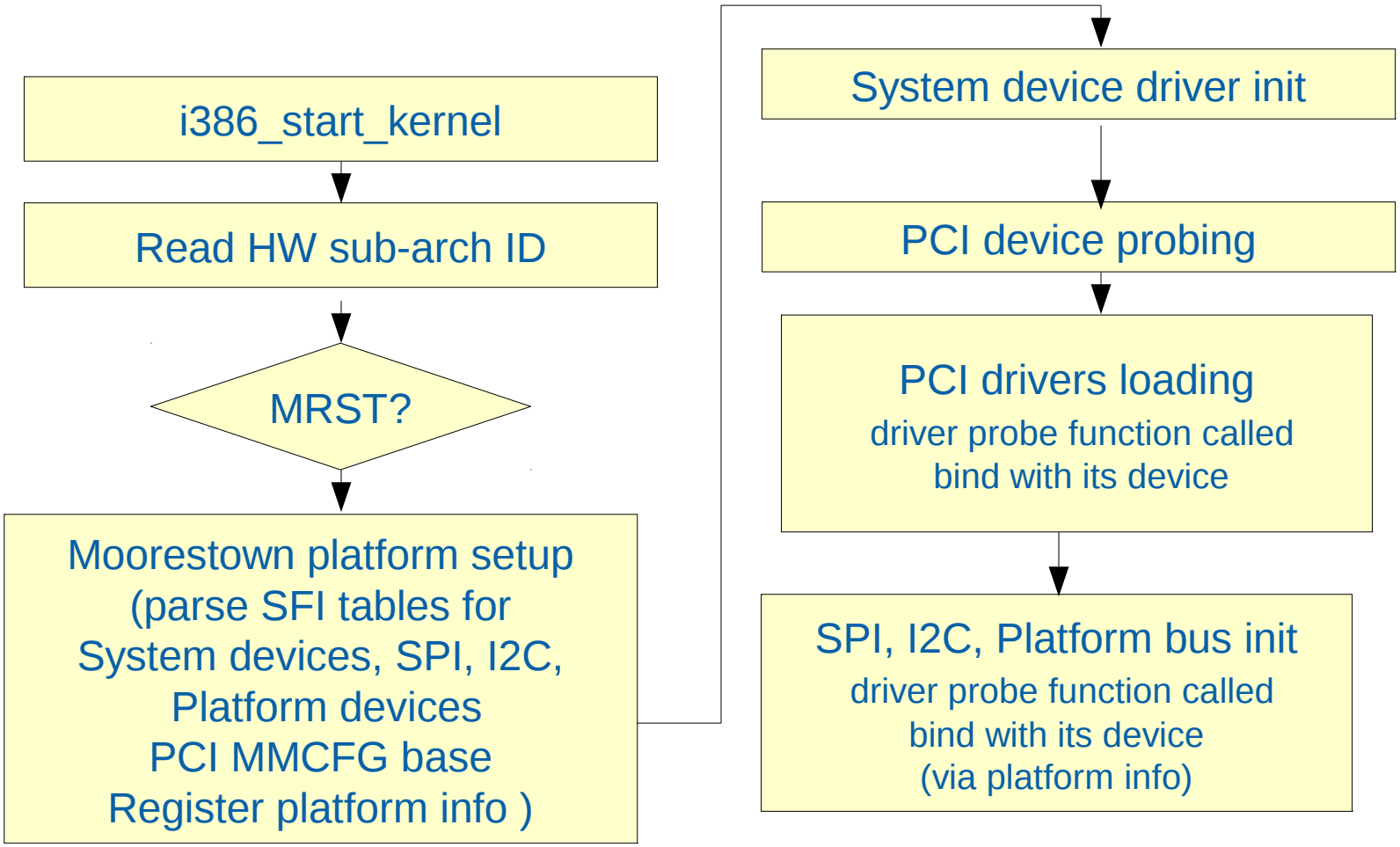
On-chip devices (graphics, USB, SD host controllers, etc):

- PCI based with mix of real and fake PCI devices
- Use PCI header to provide memory mapped IO space and IRQ routing

Non-probable platform devices (SPI, I2C, GPIO)

- SFI tables

Device enumeration and driver loading flow



PCI

True PCI in the north complex

- e.g. Graphics engine

Fake PCI covers all south complex devices

- e.g. USB, SD/MMC, NAND

PCI shim (Fake PCI MMCFG space written by the FW in main memory)

- Contains both true and fake PCI devices
- MMCFG location is stored in SFI table

PCI shim pros and cons

Pros:

- Leverage the existing device enumeration method
- Reuse generic PCI drivers

Cons:

- Can not detect device presence automatically, increased FW maintenance burden
- PCI shim is read-only therefore can not handle writes, e.g. BAR sizing
  - Added a new PCI-E extended capability called “Fixed BAR” so that the driver reads the BAR sizes from extended cap then update the fake BAR
- Mixed true and fake PCI functions under the same PCI device in order to save memory

Interrupts

Routing

- Interrupt Routing information are platform specific and obtained from system firmware via PCI MMCFG space and SFI tables
- IOAPIC redirection tables used to establish mapping between IRQ# and vectors

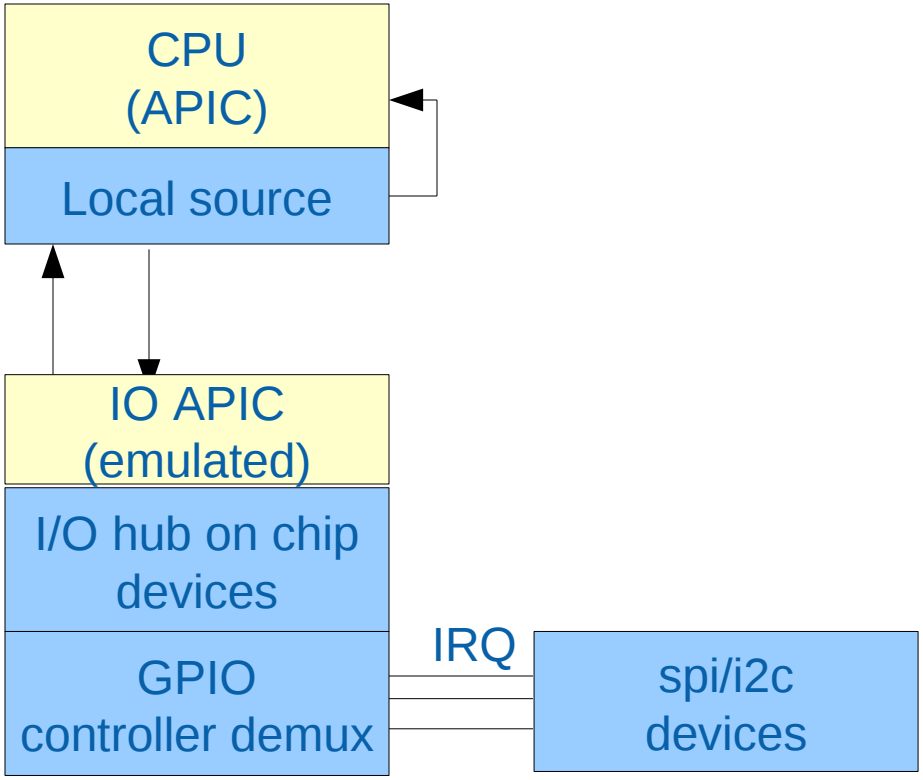
Delivery methods

- CPU Local interrupts (local APIC timers, thermal events)
- True PCI MSI (graphics)
- FSB delivery by FW emulated MSIs (on-chip devices)
- Logical delivery via GPIO IRQ chip
  - Use chained irq handler

Abstraction added

- legacy_pic

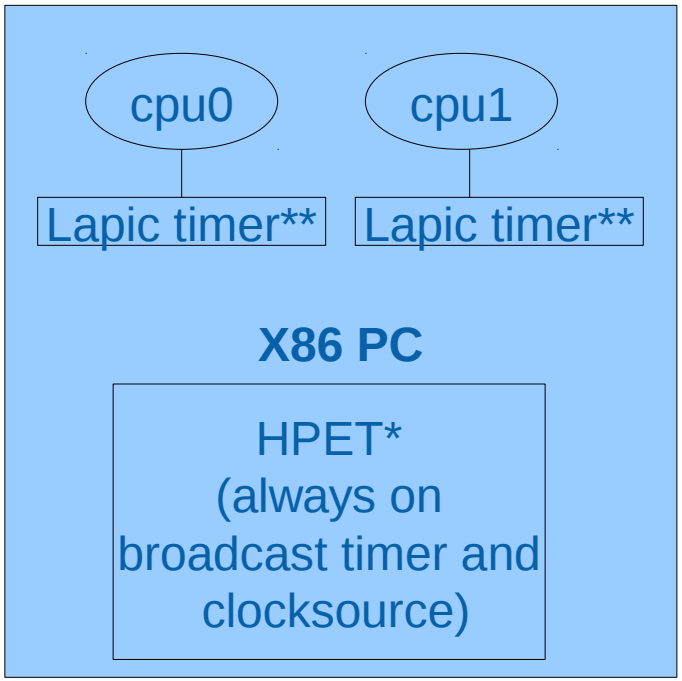
Interrupt delivery



Timers

On X86PC, typically broadcast timer is needed since local APIC timers stop working in C2 and deeper states

- Typical PC has per-cpu local APIC timer plus HPET
- Moorestown can use per-cpu local APIC timer plus APB timer

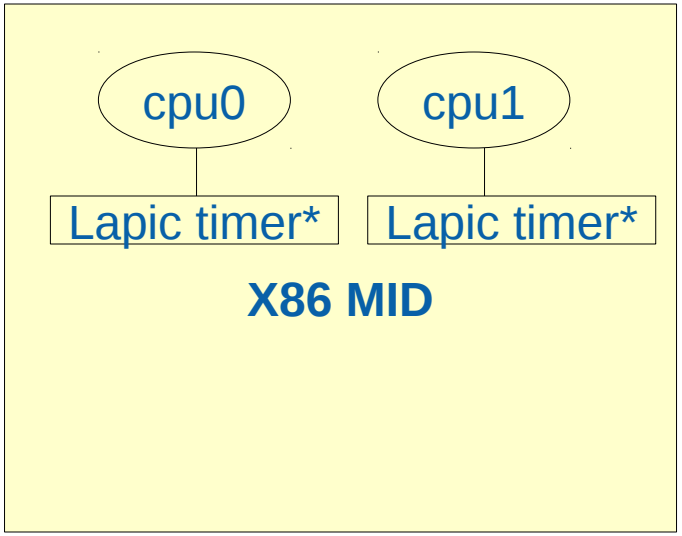


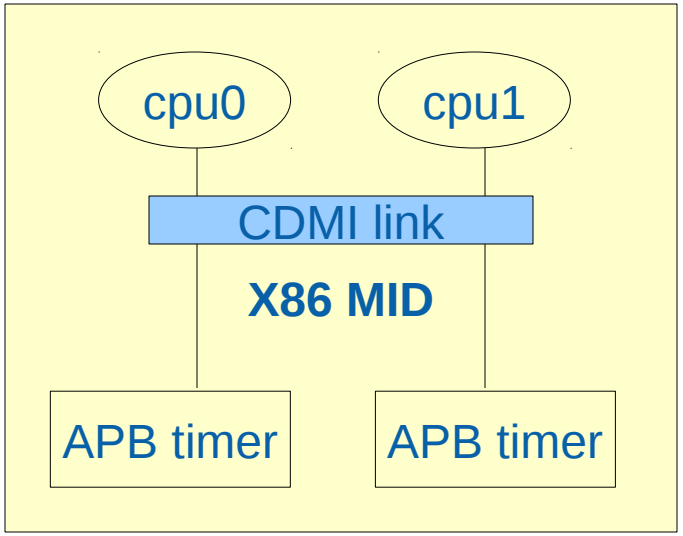
* High precision event timer
** stop at C2 or deeper CPU states

X86 MID system timer options

No more broadcast timer!

- Per CPU APB timer
- Per CPU local APIC timers (always on)





* if always-on (ARAT)

Code Change

Step 1. Clean up existing code to make room for smooth integration of Moorestown specific changes

Step 2. Add Moorestown specific enabling code

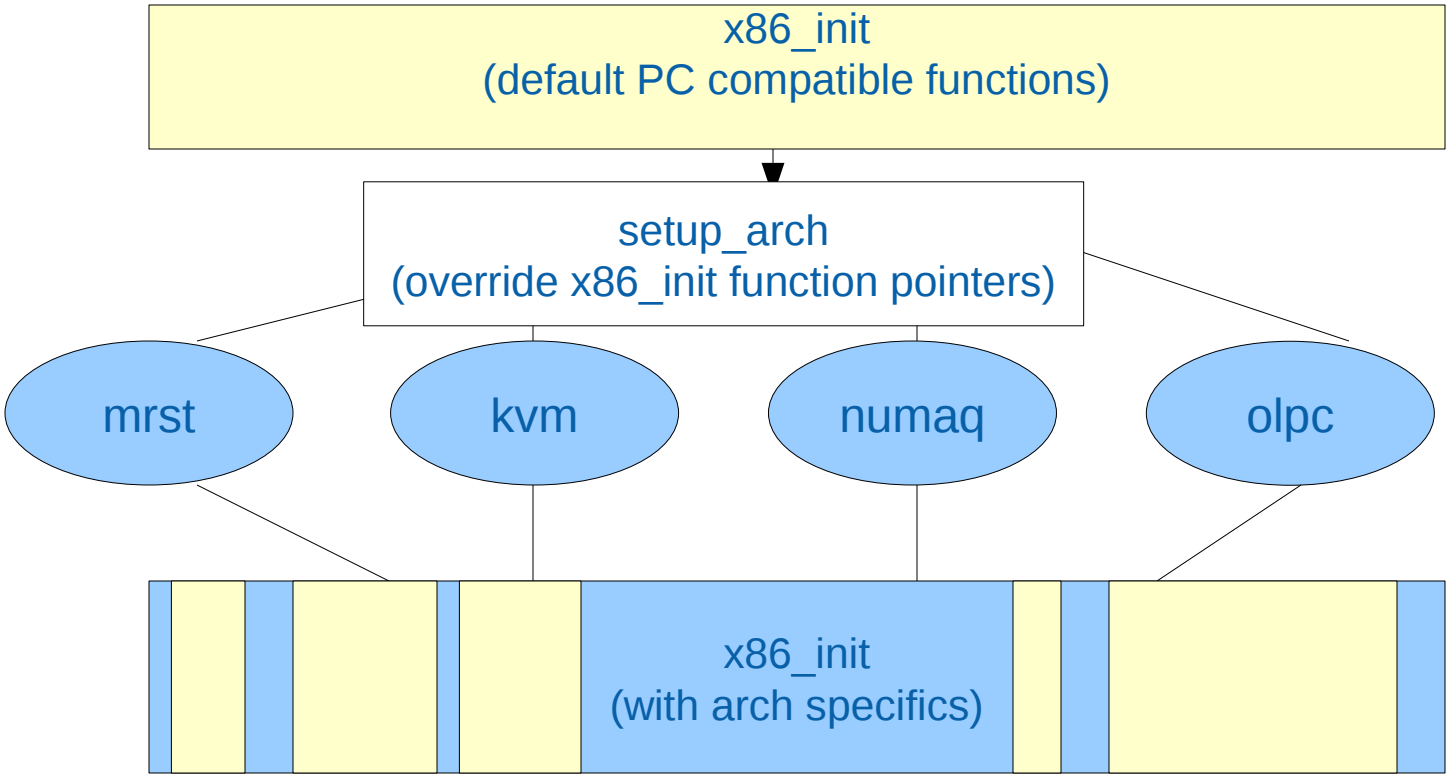
Integration with existing x86 code

First attempt is to add a set of platform feature flags so that we can workaround non-PC compatible code. Spreads out to too many places. e.g.

```
-         probe_roms();
+         if (platform_has(X86_PLATFORM_FEATURE_BIOS))
+                   probe_roms();
… ...
-         io_apic_irqs = ~PIC_IRQS;
+         if (!platform_has(X86_PLATFORM_FEATURE_8259))
+                   io_apic_irqs = ~0;
+         else
+                   io_apic_irqs = ~PIC_IRQS;
```

A much better approach: x86_init

Thomas Gleixner added x86_init layer (x86_cpuinit and x86_platform included) to abstract the common init functions for all x86 platforms.



More abstractions used

Abstractions such as x86_init, machine_ops, and legacy_pic are used to deal with platform variations at runtime.

- BIOS related
- Timers
- RTC
- Machine power on/off
- Legacy PIC

Benefit of using abstractions

Maintain arch/x86 as a unified architecture

Less run-time checking (similar to have self-modifying code for each sub-arch), example:

```
-       pcibios_fixup_irqs();

+       x86_init.pci.fixup_irqs();
```

The corner cases

There are cases sensitive to the ordering, not so easily fit in abstractions based on existing flow. e.g.

IOAPIC early setup

- Standard PC relies on legacy devices during early boot, IOAPIC is normally set up later
- Moorestown needs IOAPIC to be ready early for system timer interrupts.

Ideally, we can move IOAPIC setup early on most modern PCs such that there is no need to do special setup for x86 MID.

Current Status

Moorestown x86 enabling patches has been merged upstream since 2.6.34-rc

Questions?

Backup

Resources

- http://simplefirmware.org/

Legal Information


intel®
Leap ahead™