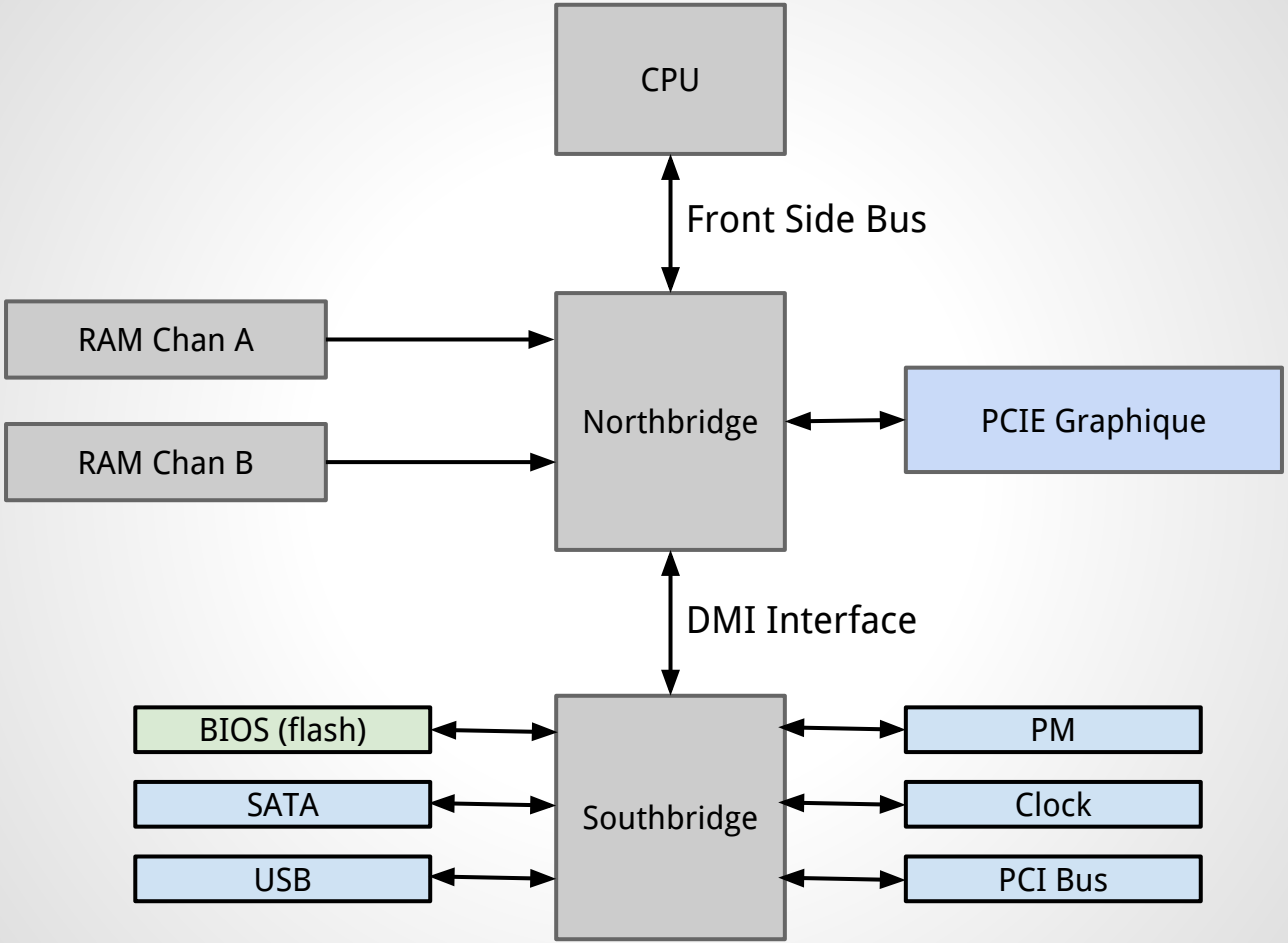
x86 : Initialization & Devices

Gabriel Laskar <gabriel@lse.epita.fr>

LSE Security System


CPU
Front Side Bus
RAM Chan A
RAM Chan B
Northbridge
PCIE Graphique
DMI Interface
BIOS (flash)
SATA
USB
Southbridge
PM
Clock
PCI Bus

x86 power on

- Chips self initialize
- Cpu initialization
- Firmware starting
- Boot code launch OS

PC Initialization - 1

- Execute the CPU reset vector (4Gb - 16b) : 0xfffffff0.
- This address is in the BIOS/UEFI flash memory.
- CPU initialization
  - platform firmware switch to the firmware mode (real, voodooo or flat protected)

LSE Security System

PC Initialization - 2

- Preparation for memory initialization
  - CPU microcode update
  - CPU specific initialization
    - need to setup some kind of stack, before RAM initialization, for that cache-as-RAM is used (CAR)
    - some CPUs need to initialize some specific MSRs
  - Chipset initialization : BAR is initialized, watchdog is disabled
- RAM Initialization

PC Initialization - 3

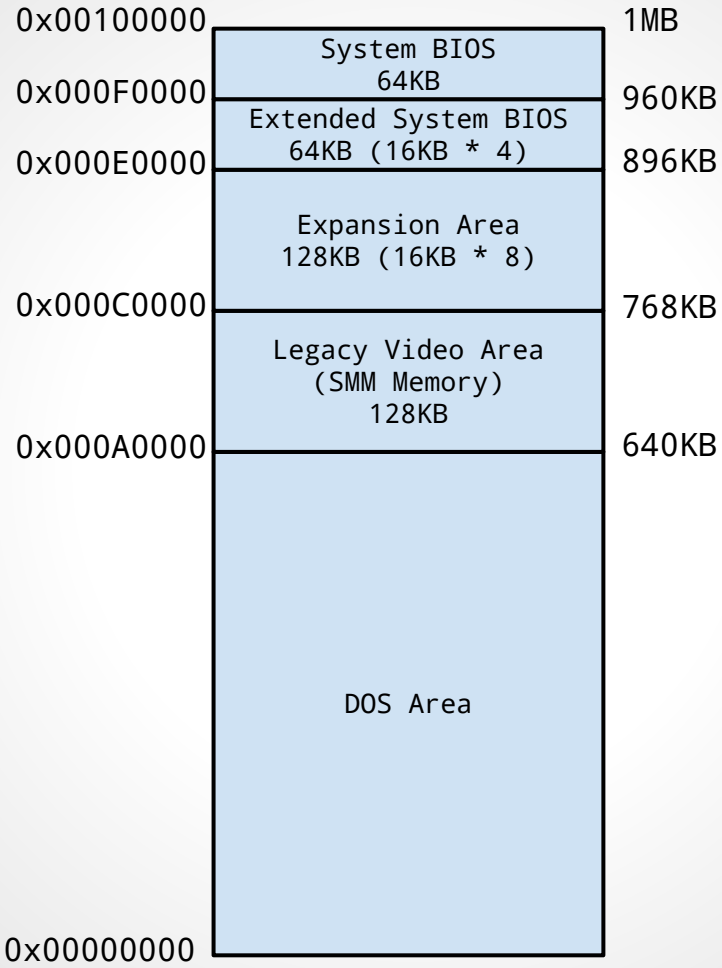
- Relocate firmware code in RAM
  - Memory Test
  - BIOS Shadowing
  - Redirect memory transactions
  - Set up the stack
  - transfer firmware to ram
- Misc platform enabling (clock, gpio)

LSE Security System

PC Initialization - 4

- IRQ enabling and IRQ chips initialization
- Timer initialization
- Memory cache control initialization
- APs initialization (MTRRs must be consistent across all cores)
- Simple IO Device initialization (ps/2, serial)
- PCI Device discovery and initialization
- OS boot loader execution

LSE Security System


0x00100000
1MB
System BIOS
64KB
0x000F0000
960KB
Extended System BIOS
64KB (16KB * 4)
0x000E0000
896KB
Expansion Area
128KB (16KB * 8)
0x000C0000
768KB
Legacy Video Area
(SMM Memory)
128KB
0x000A0000
640KB
DOS Area
0x00000000

BIOS

- 16-bit code
- old, ancient way
- boot MBR partitions (0x55aa)
- user-api is interrupts

BIOS Interrupts

- int $0x10 : Video Services
- int $0x11 : Equipment list
- int $0x12 : lowmem size
- int $0x13 : Disk Services
- int $0x14 : Serial port Services
- int $0x15 : Misc services (0xe820, ...)
- ...

LSE Security System

UEFI

- 32 or 64 bit code
- New “modern” way
- New partition format
- Interface based api

Example

```
#include <efi.h>

EFI_STATUS main(EFI_HANDLE ImageHandle,
                EFI_SYSTEM_TABLE *SystemTable)
{
  SystemTable->ConOut->Outputstring(
    SystemTable->ConOut, L”Hello World\r\n”);
  return EFI_SUCCESS;
}
```

Different types of Applications

- Applications
- Boot services
- Runtime services
- Drivers

Devices

- Registers accessible to CPU :
  - MMIO
  - PIO (in, out)
- Access to Memory (DMA)
- Interrupts (irq, msi)

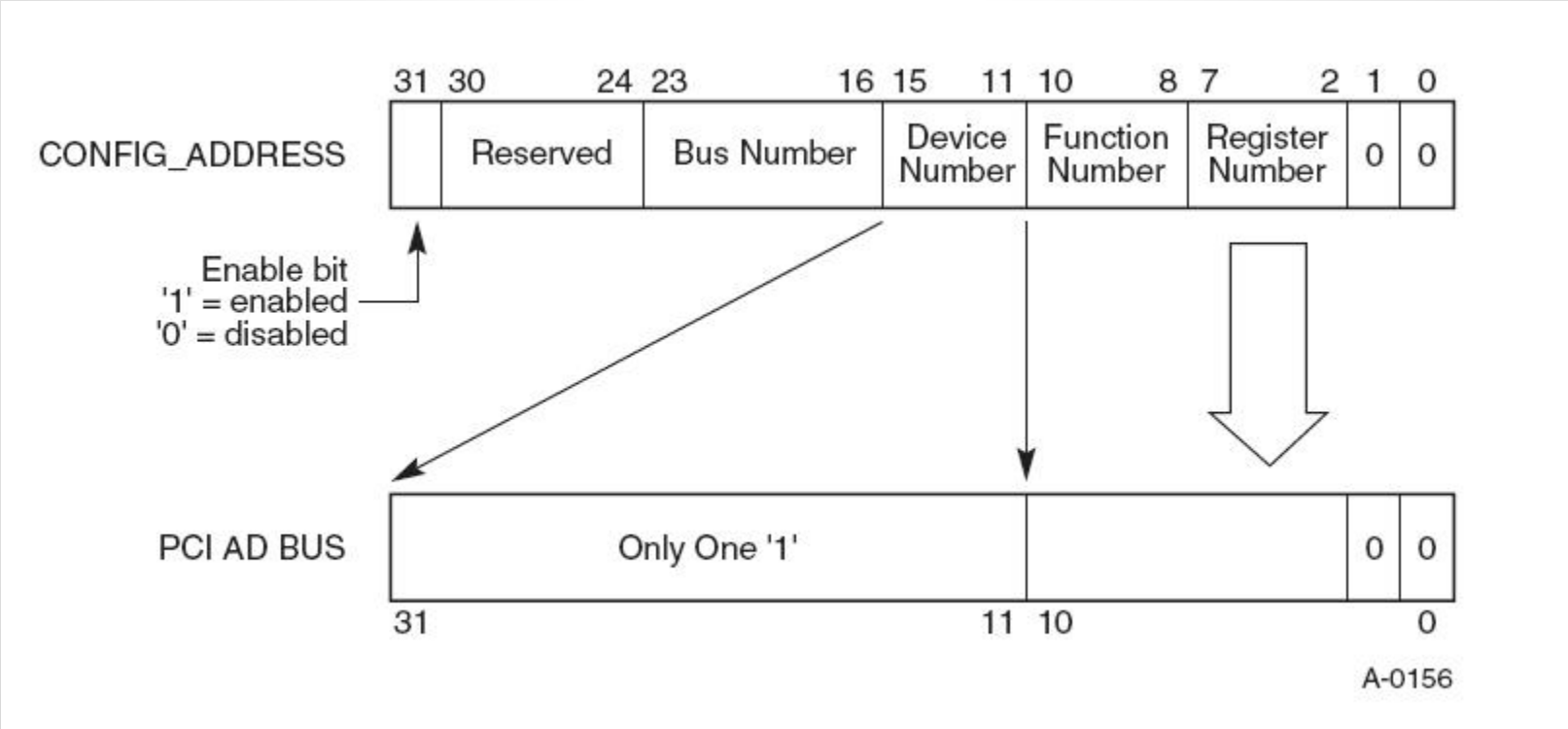
What devices are available ?

- We need some way to discover devices
- When devices are on a bus there is (usually) a way to have their description

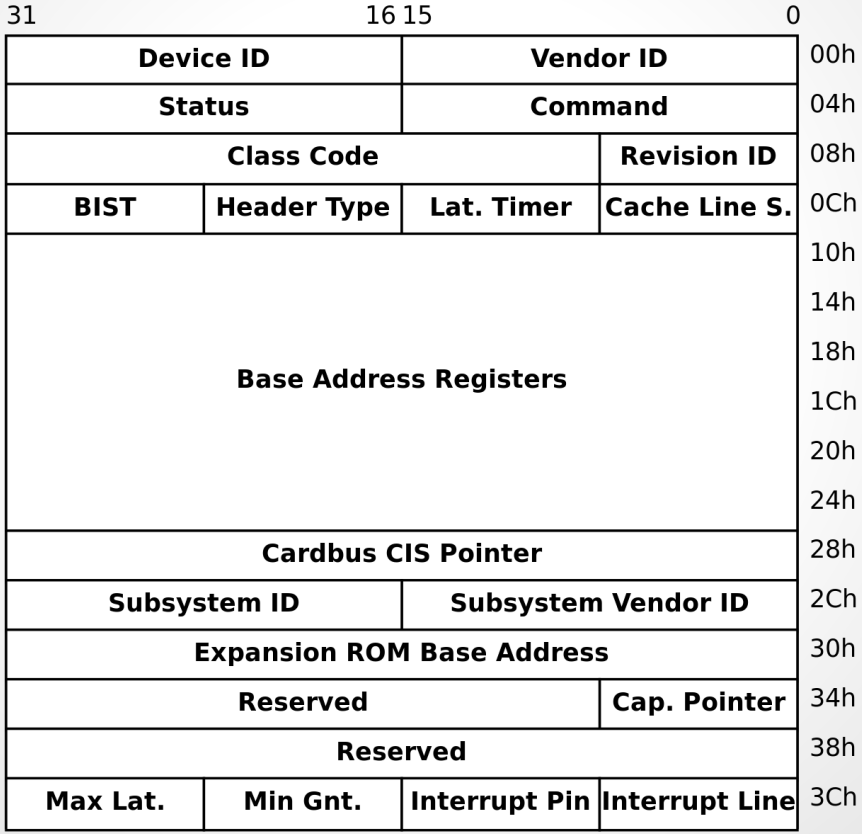
PCI Bus

- Configuration space accessed through IO Ports
- `CONFIG_ADDRESS (0xcf8)`
- `CONFIG_DATA (0xcfc)`

PCI Address Structure



PCI Header


31
16
15
0
Device ID
Vendor ID
00h
Status
Command
04h
Class Code
Revision ID
08h
BIST
Header Type
Lat. Timer
Cache Line S.
0Ch
Base Address Registers
10h
14h
18h
1Ch
20h
24h
Cardbus CIS Pointer
28h
Subsystem ID
Subsystem Vendor ID
2Ch
Expansion ROM Base Address
30h
Reserved
Cap. Pointer
34h
Reserved
38h
Max Lat.
Min Gnt.
Interrupt Pin
Interrupt Line
3Ch

Serial Port

- 8250 compatible (or 16550)
- base port on 0x3f8 (for COM1)
- IRQ 4
- ports mapped onto 8250 registers