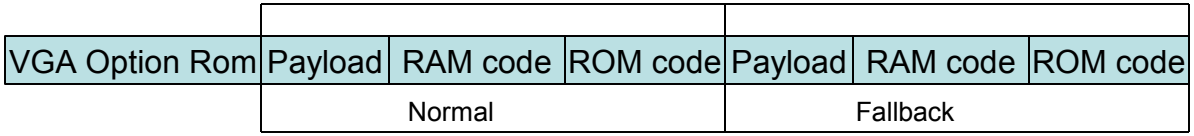
LinuxBIOS Three Images solution

yinghai.lu@amd.com

08/08/2006

Problem: For _start too lower error in fallback



Fallback: need to keep _start in ROM code in [4G-64K, 4G).

When ROM code is too big or ROM_SECTION_SIZE is too big. The _start will slip out of last 64k.

Workarounds:

a. Put _start after ram init code. And use jump after _start.

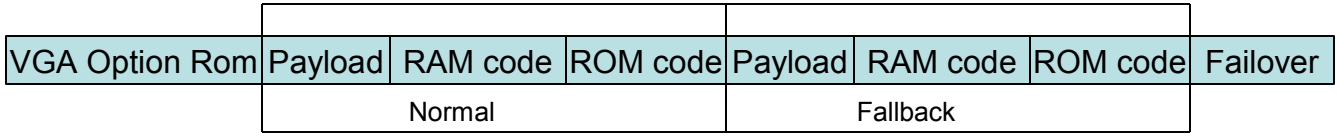
- in MB Config.lb, put entry16.inc after auto.inc, and use arch/i386/lib/jmp_auto_out.inc to execute code after auto.inc originally (copy linuxbios_ram and jmp)

b. or Cut ROM code in last 64K (ldscript_failover.lb)

- *_x = .;*
- *. = (_x < (_ROMBASE - 0x10000 + ROM_IMAGE_SIZE)) ? (_ROMBASE - 0x10000 + ROM_IMAGE_SIZE) : _x;*

LinuxBIOS Three Images solution

Final Solution:



Move out failover from fallback to separate image.

Failover: 4K:enable >64k rom access. Decide to jump to Fallback or Normal

Fallback: doesn't need to keep ROM code in [4G-64K, 4G).

- further will treat fallback as normal. and has one byte on global variables area to state if CMOS is used. The byte is set by failover.

LinuxBIOS CAR

yinghai.lu@amd.com

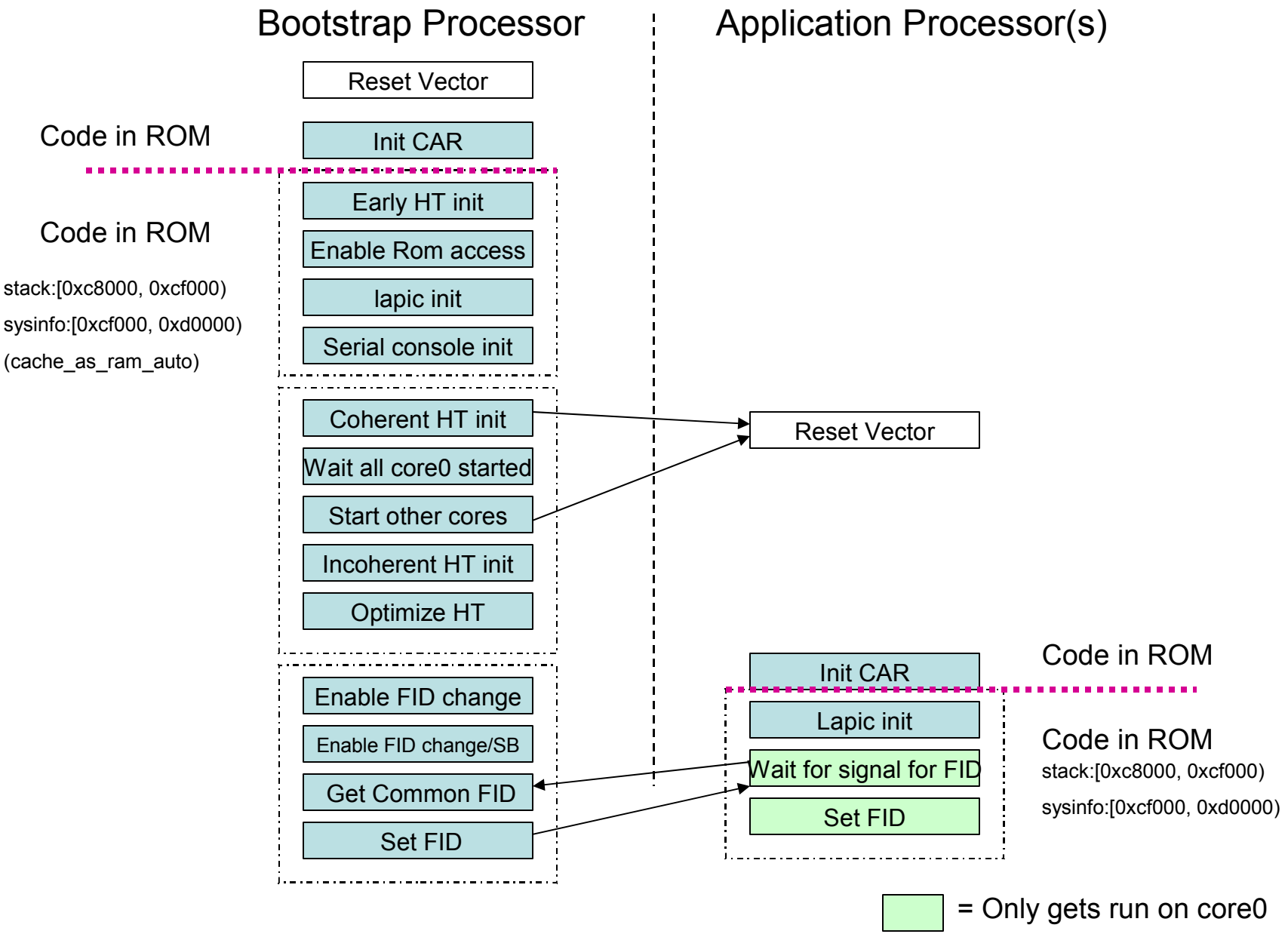
06/07/2006

It will reduce mem training and clear ECC from x*N* to x*1.1* for SMP system

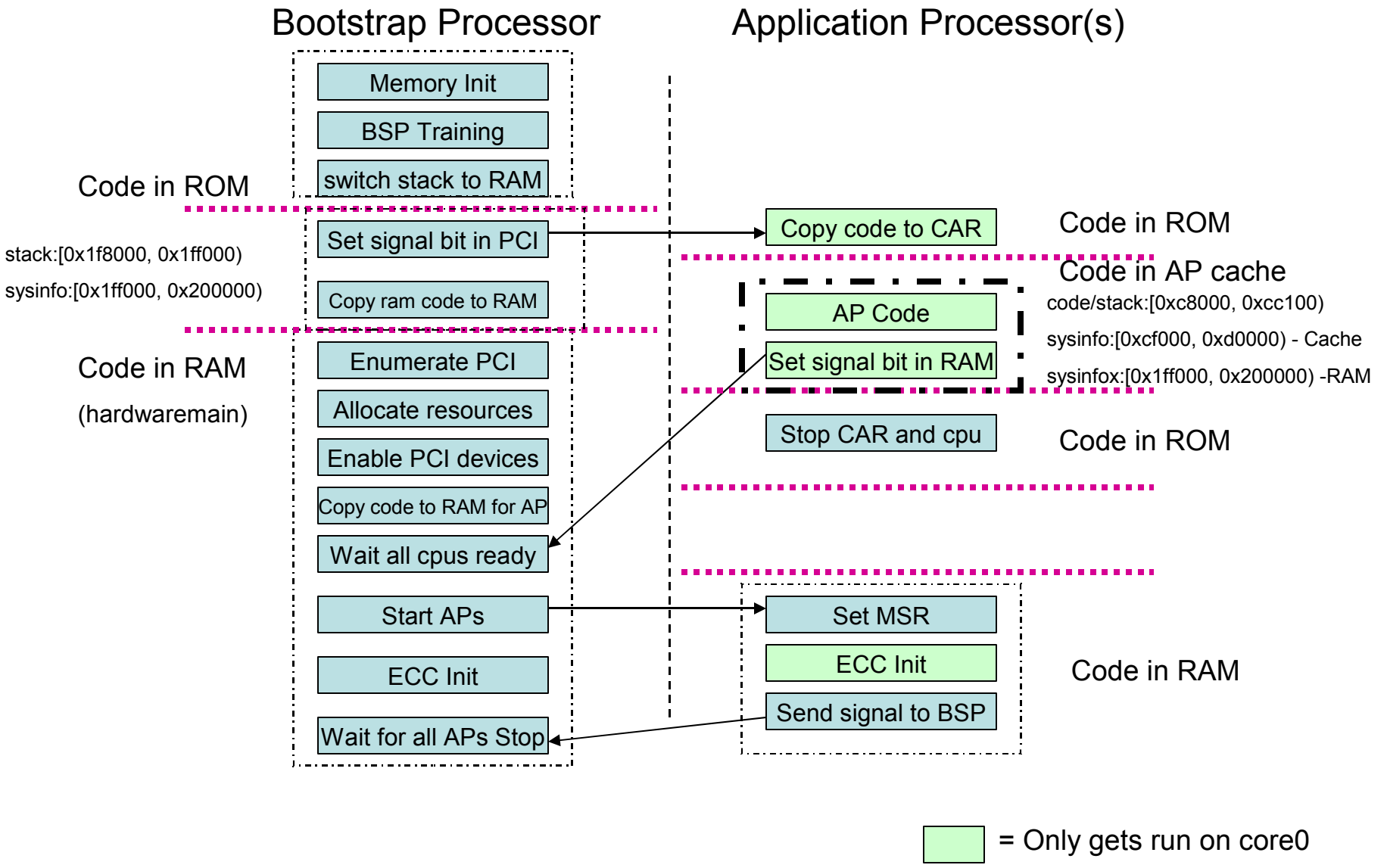
3 Phase operation

- Phase 1: Before Memory Init, --- CAR
- Phase 2: mem init/pci init
  - Memory Init
  - Training mem on first node with mem with BSP
  - Code in RAM/BSP (linuxbios_ram code for pci init…)
  - Code in CAR/AP (training with mem in AP)
- Phase 3: after ram init and pci init

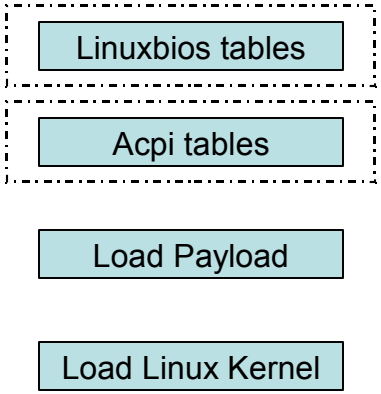
Phase 1 – CAR


Bootstrap Processor
Application Processor(s)
Reset Vector
Code in ROM
Init CAR
Early HT init
Code in ROM
Enable Rom access
lapic init
stack:[0xc8000, 0xcf000)
sysinfo:[0xcf000, 0xd0000)
Serial console init
(cache_as_ram_auto)
Coherent HT init
Reset Vector
Wait all core0 started
Start other cores
Incoherent HT init
Optimize HT
Code in ROM
Init CAR
Enable FID change
Lapic init
Code in ROM
Enable FID change/SB
Wait for signal for FID
stack:[0xc8000, 0xcf000)
Get Common FID
Set FID
sysinfo:[0xcf000, 0xd0000)
Set FID
= Only gets run on core0

Phase 2 – mem init, and mem training and linuxbios ram



Phase 3 – after RAM and PCI



LinuxBIOS CAR init and stop

yinghai.lu@amd.com

06/07/2006

1: CAR init: cache_as_ram.inc

2: post_cache_as_ram.c

   a. copy stack and global area

   b. disable_cache_as_ram

   c. copy_and_run.c

3. cache_as_ram_auto.c

LinuxBIOS CAR init and stop

1: CAR init:

a. use fixed mtrr to set [0xc0000, 0xd0000) to WB IO type

b. clear the region.

c. set the esp

d. call cache_as_ram_main

LinuxBIOS CAR init and stop

2: post_cache_as_ram:

a. use var mtrr to set mem access to [0, CONFIG_LB_MEM_TOPK)

b. copy data from CAR to RAM, (stack and global data to area near CONFIG_LB_MEM_TOPK)

c. set the esp to RAM

d. call disable_cache_as_ram

e. call copy_and_run

LinuxBIOS CAR init and stop

disable_cache_as_ram

a. disable the cache

b. clear the fixed mtrr

c. disable fixed mtrr, it will enabled by linuxbios_ram stage code again.

LinuxBIOS CAR init and stop

copy_and_run:

a. use unrv2b in C to uncompress linuxbios_ram code from flash to RAM.

b. jmp to linuxbios_ram (hardware_main will be called)

LinuxBIOS CAR init and stop

cache_as_ram.inc will be used by crt0.s, and
cache_as_ram_auto.c will be compiled to
cache_as_ram_auto.c and inclued by crt0.s or to init.o
and linked with crt0.o.

LinuxBIOS CAR init and stop

cache_as_ram_auto.c

a. start point is cache_as_ram_main (called by cache_ram_ram.inc).

b. call real_main.

c. real_main will init console/HT/RAM, and at last it will call post_cache_as_ram.