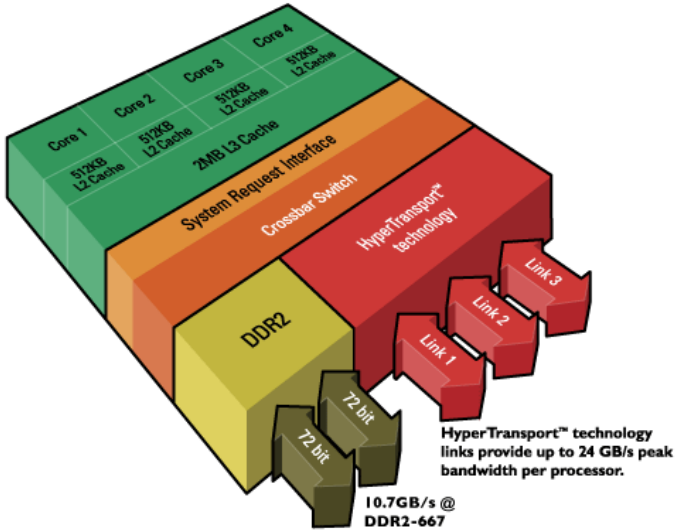
LinuxBIOS Enablement Strategy @AMD

Jiming Sun
10/01/2006

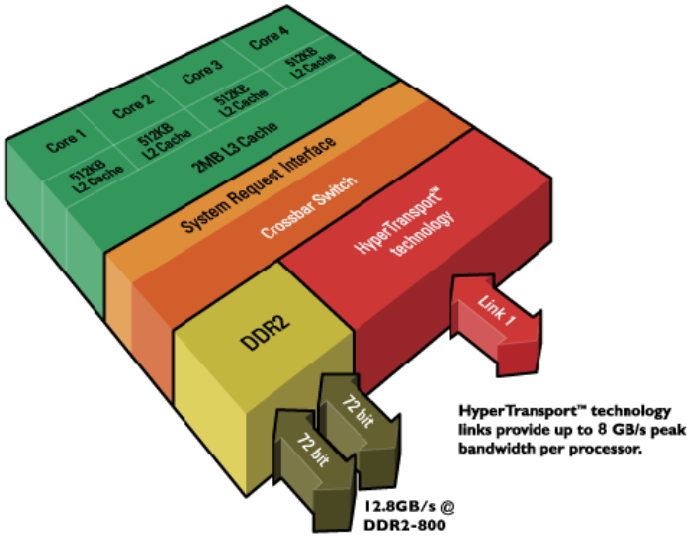
Next-Generation AMD Opteron™ processor support code was released

- Released on 9/29/2006 by YH Lu
- Main features:
  - The code is LinuxBIOS native code. (evolved from previous LinuxBIOS code)
  - DDR2 support
  - Code in AP's Cache
  - Reduce RAM training and ECC clear from xN to x1.1
  - More configurable ACPI support (SRAT, and SLIT support)
- Support was provided to several ODM/OEM customers under NDA before code is released to public

Barcelona (Quad-Core) Features



Quad-Core AMD Opteron™ Processor Design for Socket F (1207)



Quad-Core AMD Opteron™ Processor Design for Socket AM2

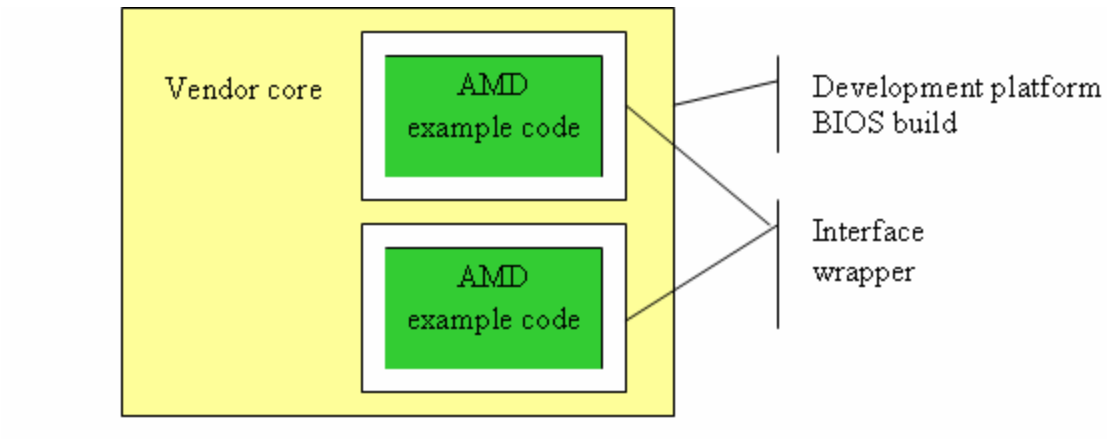
- New Northbridge:
  - New register definition for memory controller: 10-15% difference, amplified by algorithm changes for memory training.
    - *A handful more ddr timing registers to calculate and preload*
  - New bit encoding of HT routing registers
  - A handful more xbar xcs registers to pre-load
  - Added ungang feature
- New Core:
  - New register definition for various control/configuration
- Split Plane support

AGESA

- What is AGESA?
  - AMD Generic Encapsulated Software Architecture
  - There are currently 3 major modules: CPU, Memory, and HyperTransport.
- It has been in existence for 3 years, well accepted by IBV and make-BIOS OEM/ODM; reduce product TTM drastically.
- Current AGESA (REV 2.xx) is in ASM, supporting Next-Generation AMD Opteron™ processors (G and E versions).
- New AGESA for Barcelona (quad-core generation) and later
  - Written in C; built with modern tools.
  - 16-bit and 32-bit assembly code can be released at the same time (output of compilers).
- Customers have the options use AGESA with LinuxBIOS, UEFI/PI implementation, or legacy BIOS under proper license agreements.

AGESA Theory Overview

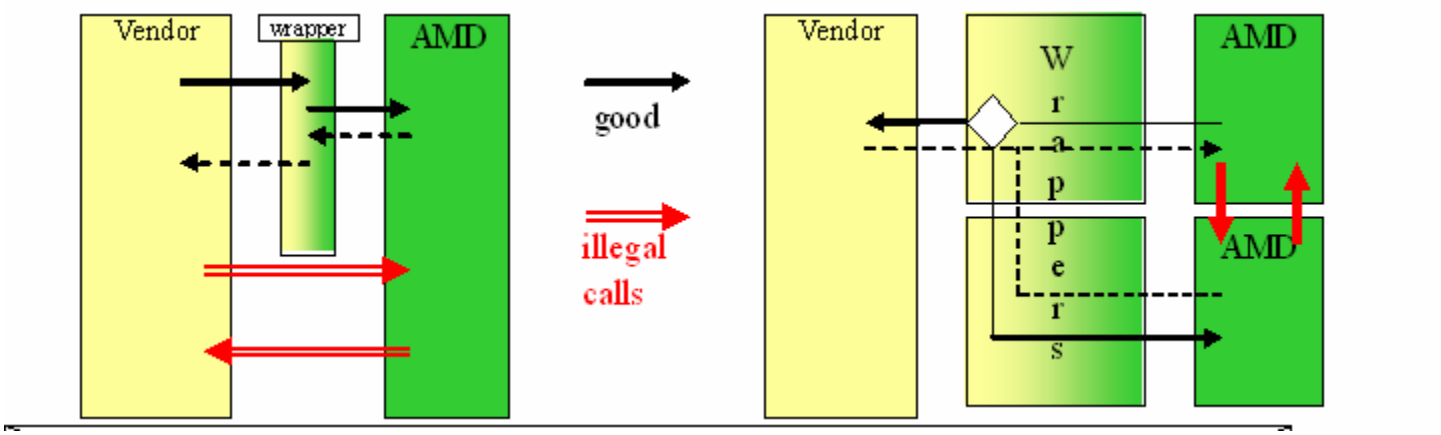
- Create a translation shell, or wrapper, of interface routines that isolate the generic AMD example code from the specifics of the BIOS vendor core code.



- The vendor code will call a routine in the wrapper. the wrapper code will modify the input parameters as necessary to meet the AMD interface definition then calls the AMD generic routine. Upon return, the wrapper code again modifies the return parameters to match the vendor base code needs and returns to the caller.

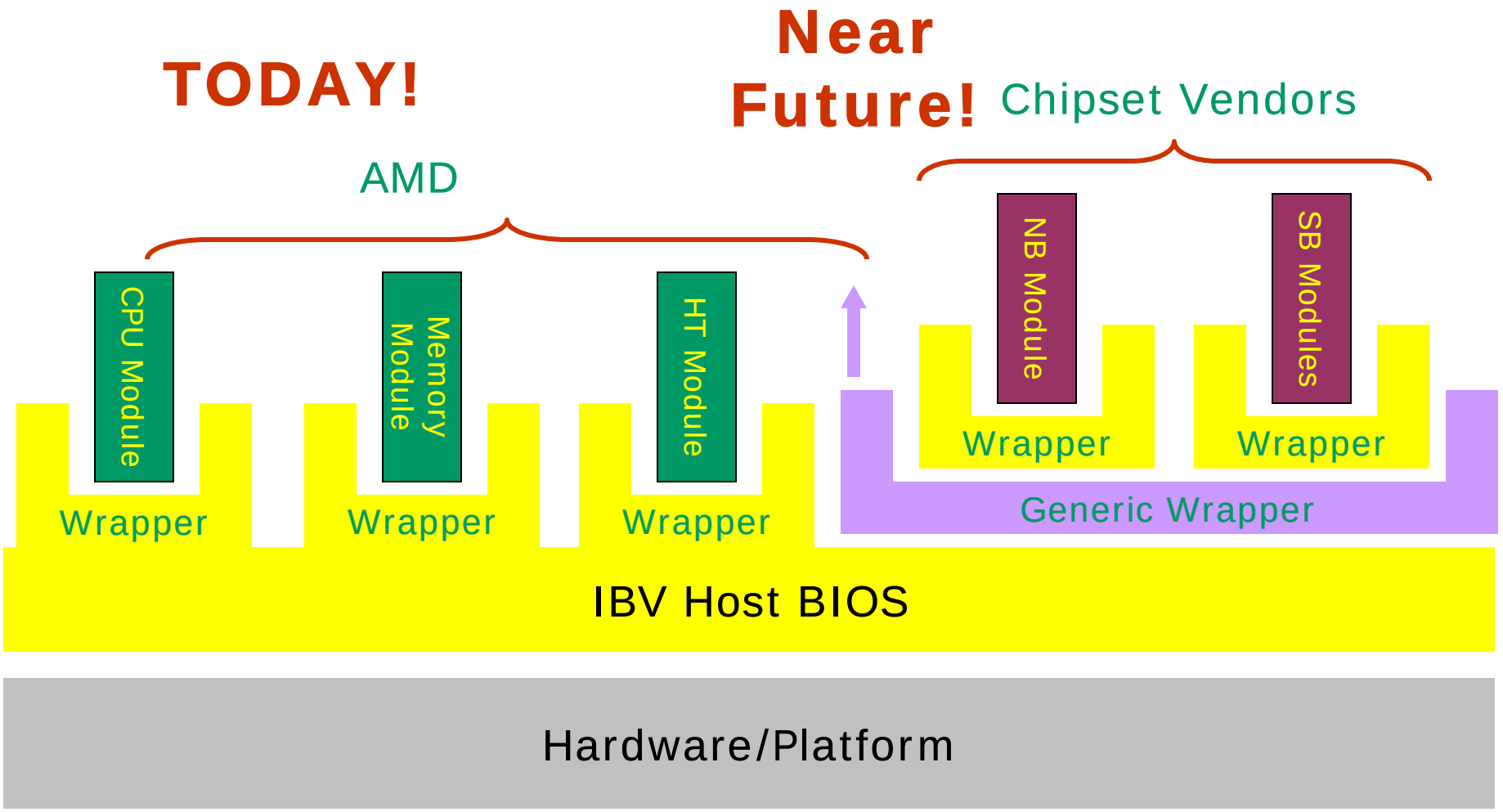
AGESA Design Guidelines

- Each AMD block has its own wrapper and must call out through its wrapper to acquire services



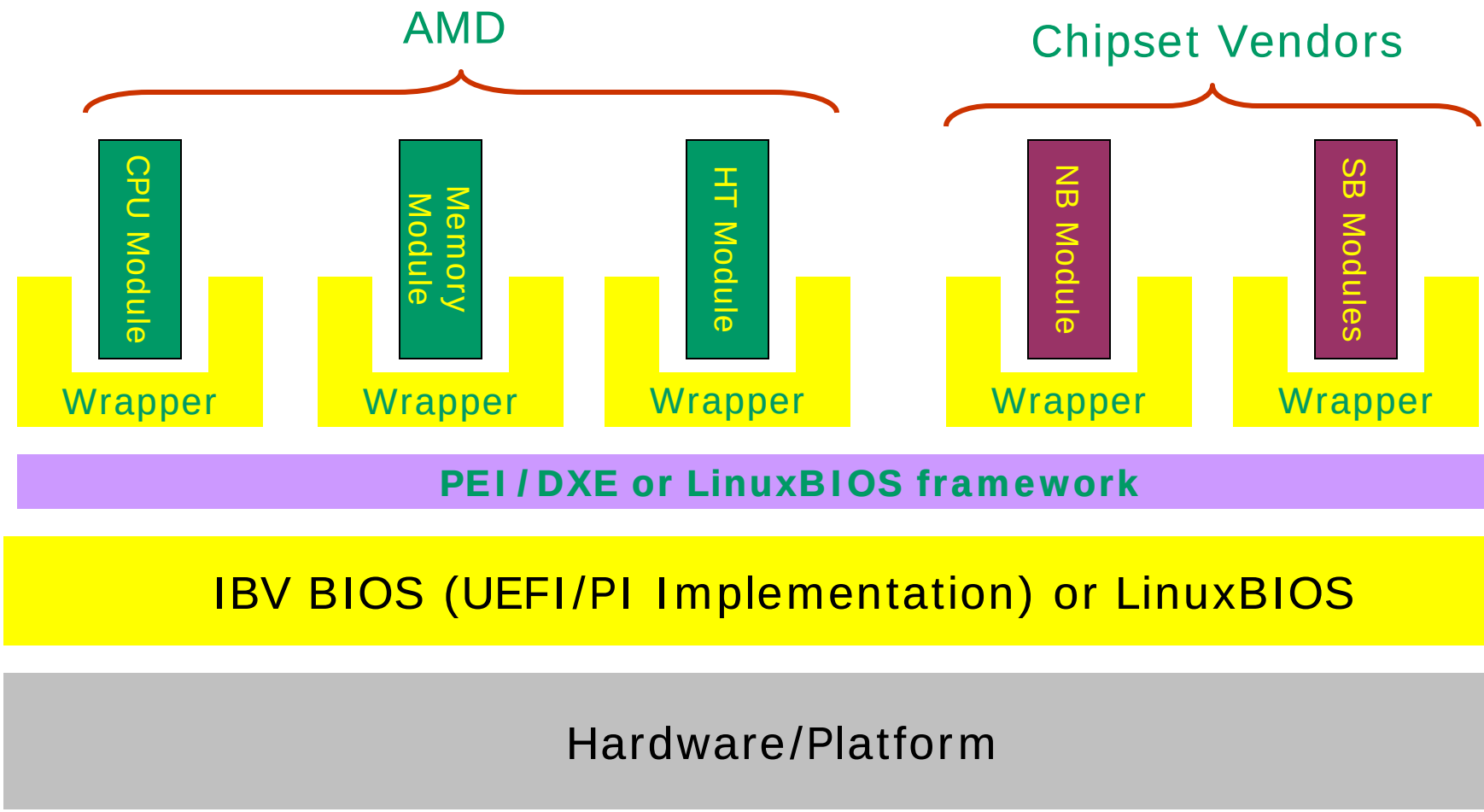
- Wrapper implementer may translate AMD call to a vendor internal routine or it can make a call through another wrapper to a 2nd AMD block that provides the requested service.
- AMD blocks do NOT expect other AMD blocks to be present

AMD wants to see all silicon vendors under AMD ecosystem to provide “encapsulated” code

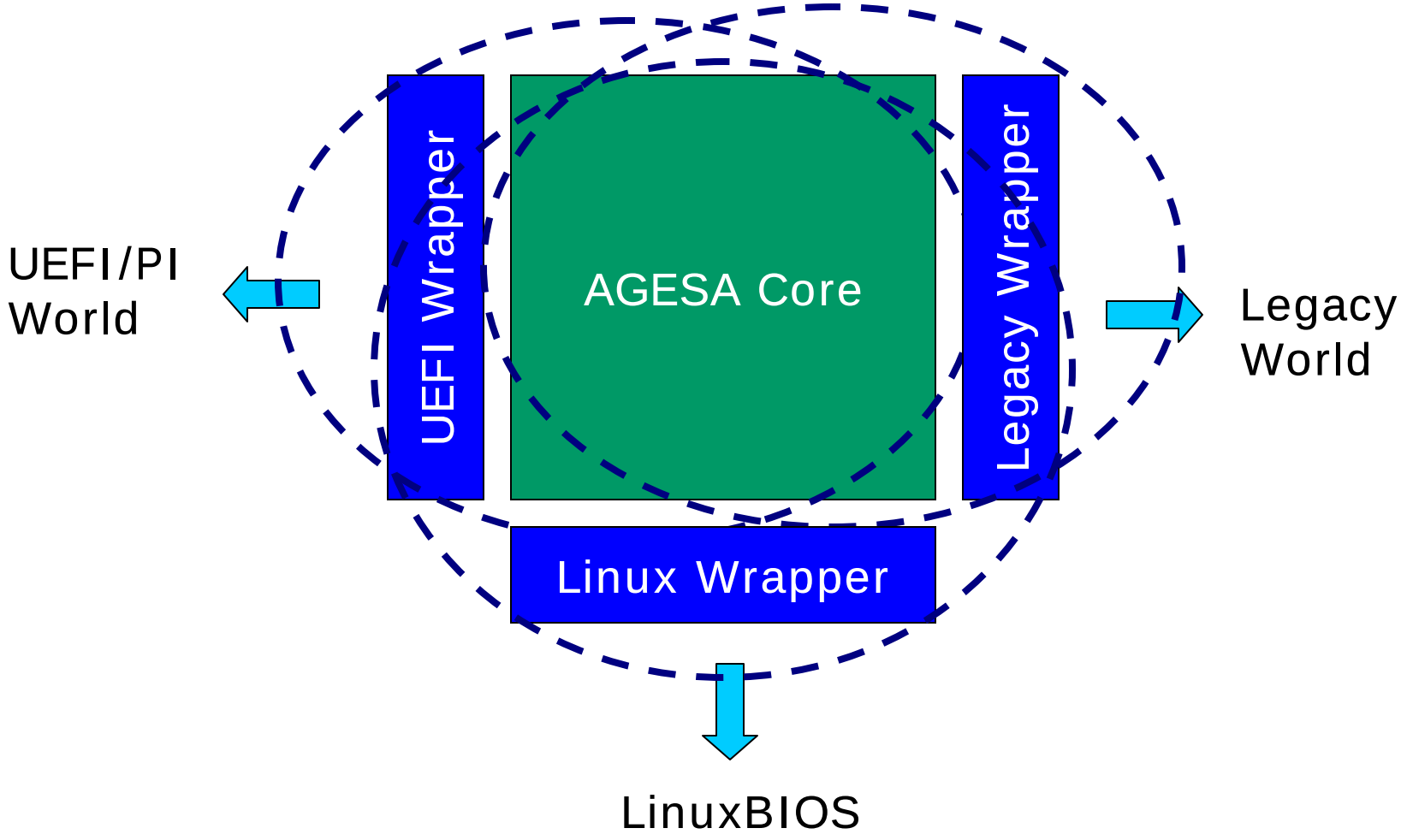


The “wine” should fit into different bottles

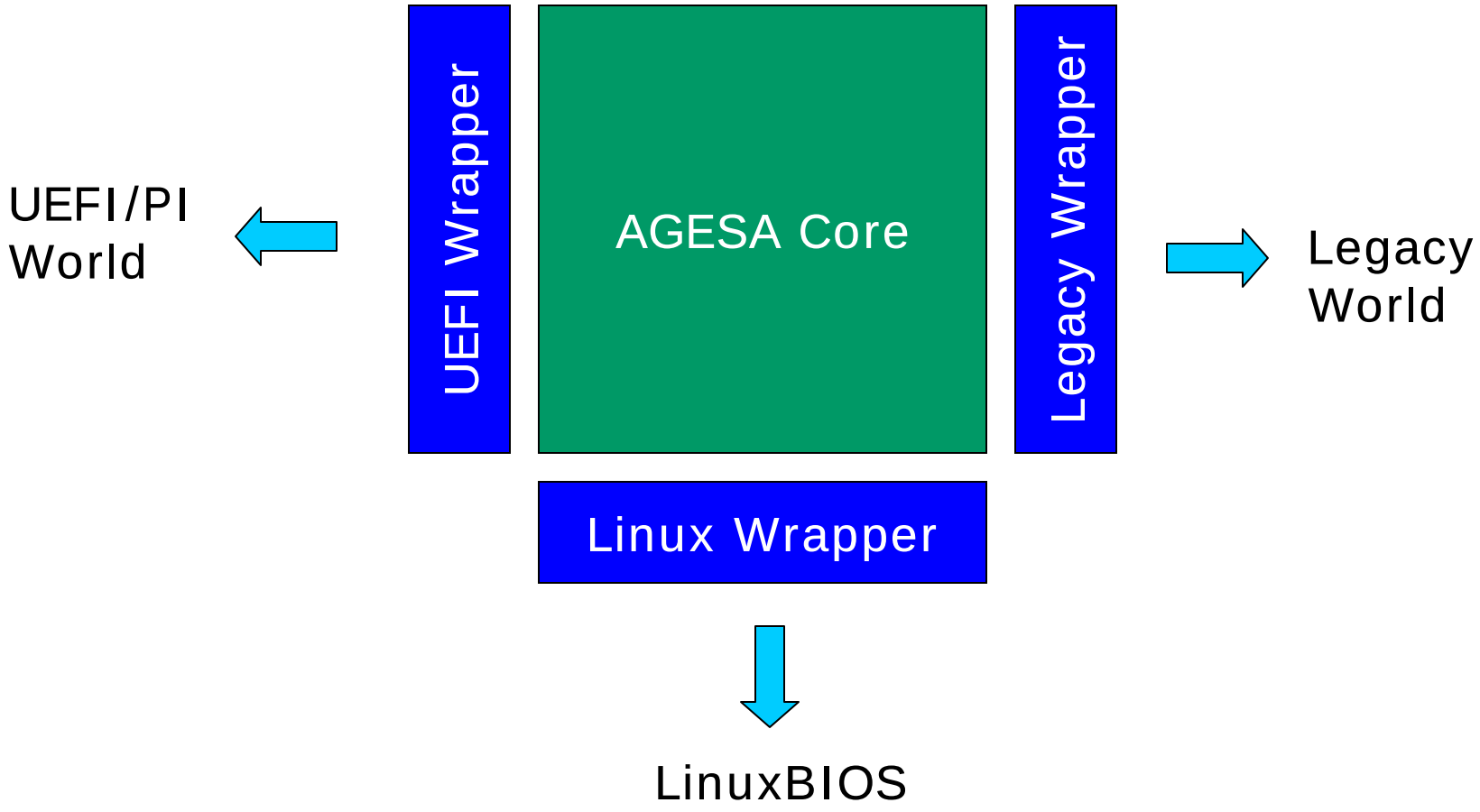
Before Silicon module distribution model is standardized, this is AMD’s method to help customers reduce TTM



AGESA's Evolution (Quad-Core Time Frame)



AGESA's Evolution (Quad-Core Time Frame)



Demo

- Soohoon Lee
  - New to LinuxBIOS
  - New to AGESA
- SimNow usage
- Link LinuxBIOS with AGESA Memory module

A Few Words about SimNow

- AMD BIOS team and strategic customers are using SimNow to develop BIOS before silicon is available and for large MP system simulation
- Currently there is public version SimNow with limited features.
- We encourage LinuxBIOS community to use SimNow for development and testing
  - There are only limited physical MB out there: SimNow can simulate more MB configuration.
- Serious LinuxBIOS developers can sign NDA to get NDA version SimNow to simulate code using MB models. (e.g. LANL, LNXI,...)

Why is it a good thing to use AGESA?

- Well tested and validated by IBV and customers
- Leverage hundreds of engineers' effort instead of a few LinuxBIOS engineers in AMD
- Well encapsulated, yet easy to customize
- Well documented
- Higher stability, higher performance, with more configuration options, and allow fast deployment of fixes.
- Microcode patches and errata fixes will be included whenever applicable.

UEFI

- UEFI will be an industry-specified replacement for the BIOS's legacy (Real Mode) *booting interface*
- It's an open follow-on to Intel's unilateral EFI spec
- AMD supports UEFI and participates actively on The UEFI Forum's Board and in its Work Groups
- We're working to prepare a spec that will be neutral across silicon vendors, firmware vendors and OS's
- Tiano (aka "The Framework") is one possible implementation of (U)EFI; the IBV's and OEM's have others
  - BIOS isn't dead (but its booting interface is worn out)
- This will be a slow transition
  - UEFI will be supported by Vista but not preferred over legacy booting
  - AMD expects the transition to take five years or more
- AMD is working with IBV's and OEM's to prepare and test UEFI solutions for AMD platforms
  - AMD is continuing to support legacy solutions as well

    Customers and the market can decide when the transition happens

The PI (Platform Initialization) Spec

- PI will be a new firmware-to-silicon *interface*
  - An open, industry-specified follow-on to Intel's PEI/DXE specs
- The PI Goal: portable, silicon-support modules that can plug into all industry firmware implementations
- AMD is active in the PIWG; working to define the PI Spec
- The current PEI/DXE specifications do not cover AMD and other chipset architectures very well
  - They need some re-work to make them architecture & vendor neutral
- PIWG is part of The UEFI Forum (for legal convenience) but PI will not be required by UEFI (and vice versa)
- Until PI supports non-Intel silicon architectures, AMD will continue to support customers using its AGESA modules
  - AGESA currently provides a common, portable CPU module
  - Chipset modules still vary by vendor
- AMD customer support organizations will continue to support legacy BIOS customers during the (relatively long) transition to PI and UEFI

UEFI Adoption and Timing

- Windows Vista will use either the legacy interface or UEFI (no advantage for either method)
  - Vista does not force UEFI adoption; it allows experimentation
  - Only future versions of Windows will begin to prefer UEFI over the legacy interface
- UEFI end-user benefits are subtle
- Like all new things, UEFI has costs and risks
- Enthusiasm for UEFI varies by segment and by OEM
- Very difficult to predict speed of adoption
  - AMD will continue to support the legacy firmware interface
  - AMD is also working with the firmware companies to ready UEFI solutions on AMD platforms for early adopters
    Demonstrated at WinHEC (April 2005): AMI BIOS using EFI to boot Windows on an AMD server platform
  - The market will decide when each segment switches to UEFI
  - Complete switch-over likely to take 5-7 years or more

- AMD is not pushing a transition timetable
  - AMD is prepared to support its customers, whatever their timing and whichever path they choose

Comments on UEFI and PI Specs

- UEFI and PI do not dictate an implementation
  - Intel's Tiano "Framework" is one implementation
  - Other BIOS teams are creating other implementations
  - UEFI and PI can also be added to legacy BIOS code bases
  - This flexibility is good; OEM's and IBV's can select their architectural approach and control the timing of their migration
- UEFI and PI will drive other changes in firmware
  - Long-mode, 64-bit infrastructure code is needed to support the new interfaces
  - Shift from assembler to C coding, modern tools and increased modularity
- These are long-term changes
  - It will many years before these specs are ubiquitously supported by all firmware in the industry
  - Timing will be controlled by our customers (and market forces), not imposed
- AMD is working to make the transition safe and painless

- **AMD will support all solutions**
  - **With on-going, simultaneous support for legacy BIOS, LinuxBIOS, and UEFI/PI firmware**
  - **AMD is not urging a transition timetable**
  - **AMD is supporting customers, whichever path they choose**

Problems with Tiano (Intel's implementation)

- From Tiano 8 to Tiano 9, Intel changed vast amount of the code including the architecture itself
- From revision to revision, the new CPU code did not come in "modular" form; they come in a folder with 100+ files
- Since it is not a drop-in-and-work driver distribution model, customers are all busy in keeping up with Intel's changes while keeping legacy programs going.
- Since Intel could not distribute its code in modular format and is still ambiguous of its future plan, no silicon vendor in the industry can distribute silicon code in modular format because standard "sockets" are not there.
- Even if silicon vendors want to formalize a silicon module distribution model, but no one knows if Intel would yank the carpet underneath them by changing the architecture again

Q & A

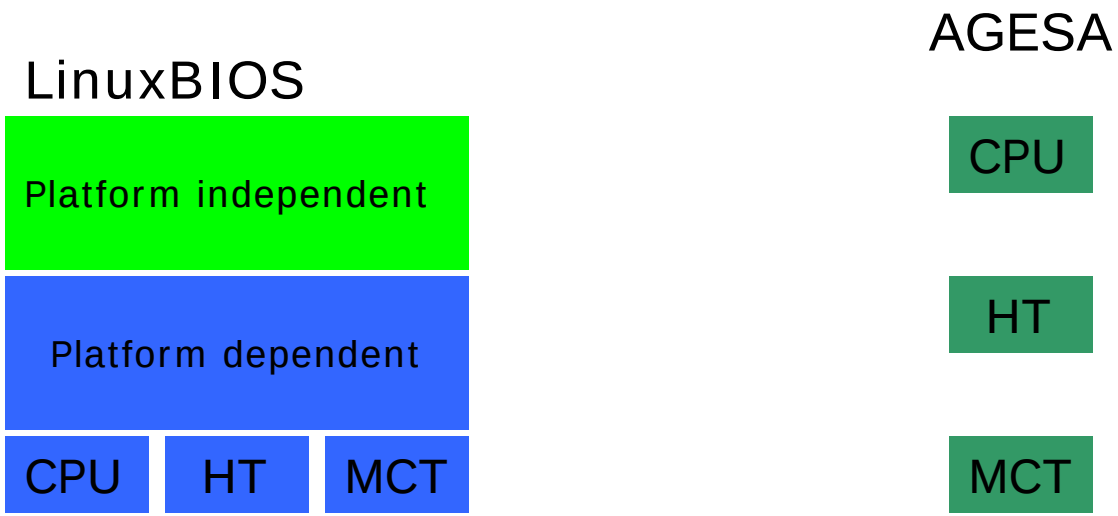
LinuxBIOS + AGESA

Soohoon Lee
CSS-CDS
September 2006
Rev. 0.3

What is it?

- It replaces LinuxBIOS' MCT configuration code with AGESA code.

- Also possible to replace CPU and HT configuration codes.



Simple Four Steps for Implementation

1. Convert AGESA source file.
   - Script asm2h.sh converts mct.inc into mct.h.
   - Merge AGESA asm files you need and run script, fixagesa.sh.
     It modifies specific register usage patterns and segment register
     handlings.

2. Assemble and convert object file.
   - assemble it with MASM with /coff option
   - convert it to linux ELF with objcopy.

3. Patch LinuxBIOS
   - 15 lines of modifications to platform independent files
     2 lines of change to platform dependent file.
   - If this patch gets included in LinuxBIOS later,
     you won't need this step.

4. Compile and go.

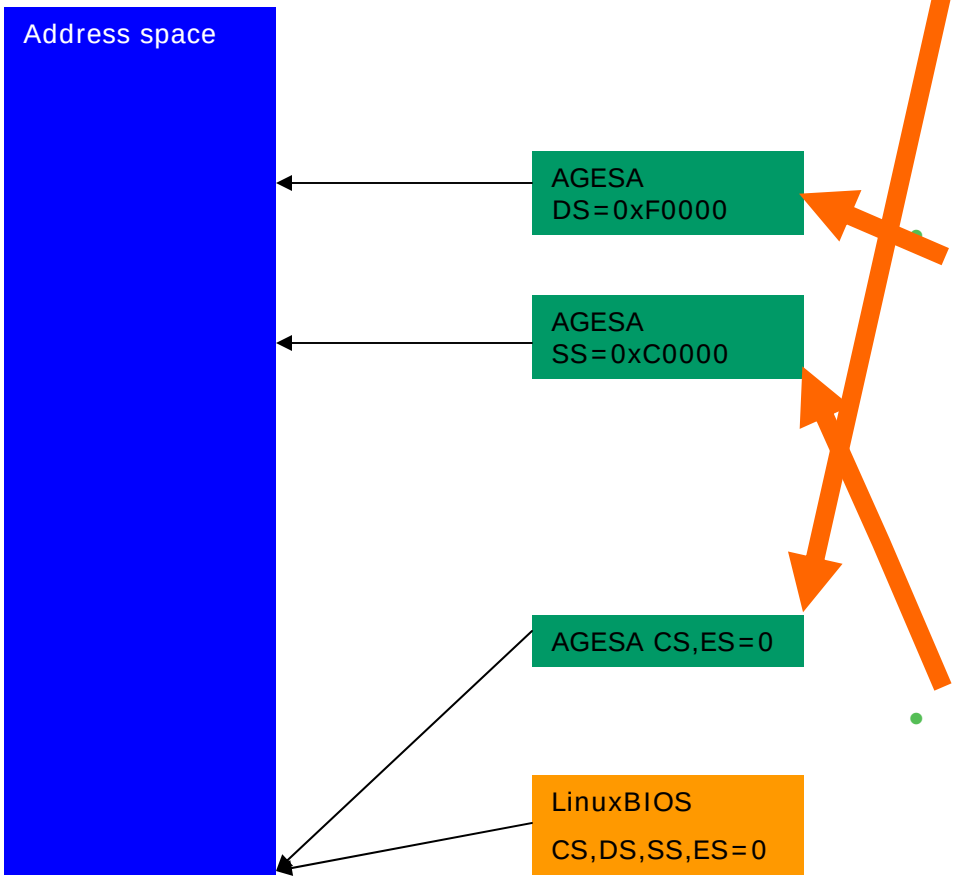
Issues in running AGESA under LinuxBIOS

- Main problem is that their running modes are different.
- There can be many issues with modes but in this case we have two issues,
  - Instructions
  - Memory model
    (address space, seg registers and offset size etc.)
- Instruction wasn't a problem except E/CX in repeat instructions and sse instructions which need 0x67 prefix in real mode.
- But memory model issue required some work.

Operating Mode (1)

- LinuxBIOS runs in protected mode(PM32) with all segment bases set to 0.
- AGESA runs in real mode and uses 16bit offset(RM16).
- Fortunately, AGESA doesn't modify segment registers directly but only moves them around between them.
- For AGESA+LinuxBIOS, two new segment descriptors were added, one for AGESA data and another for AGESA stack segment.
- And they are carefully managed to make them work together.

Operating Mode (2)



- CS desn't need to be changed so its base is always zero and It's 16bit offset safe.
- AGESA uses DS, only when it copies pattern data from CS.
  So it's preset to right value and fixagesa.sh comments out "push cs; pop ds;" from AGESA source.
- SS is pointing C-A-M base address to make it 16bit offset safe.

Object Codes

- Incompatibility in object code formats - MASM coff and Linux ELF format

- MASM coff is not that coff format which Linux understands.
  - Objcopy was used to convert MS coff (pe-i386) format to Linux ELF.
  - Its relocation info for external symbol was also different.
  - It makes AGESA external calls to jump to target address+4.
  - The workaround that I took:
    1. Merge AGESA files to make all AGESA symbols known at assemble time.
    2. Wrapper functions, they are externals to AGESA, handle target+4 jump by adding intermediate function. It also modifies SS and ESP just after C function prolog(4bytes) and before epilog and calls real function.

Sources files and how to (1)

- Run asm2h.sh

```
$ asm2h.sh ../AGESA/MCTB/mct.inc > mct.h
$ cat asm2h.sh
awk '{if ($2 == "EQU") { print "#define", $1, $3; } \
      else if ($2 == "STRUCT") { print "struct", $1, "{"} \
      else if ($2 == "BYTE") { print "unsigned char", $1, ";"} \
      else if ($1 == "BYTE") { print "unsigned char BB"NR ";"} \
      else if ($2 == "WORD") { print "unsigned short", $1, ";"} \
      else if ($2 == "DWORD") { print "unsigned int", $1, ";"} \
      else if ($2 == "ENDS") { print"}__attribute__((pack(4),aligned(8)));" } \
      print "//", $_; }' $1
```

- Merge AGESA files you need

```
$ cat ../AGESA/MCTB/mct*asm \
 ../AGESA/Addendum/MCTB/mctproto.asm \
 ../AGESA/Addendum/MCTB/mctardk0.asm > m.cat
```

Sources files and how to (2)

- Run fixagesa.sh

```
$ fixagesa.sh m.cat > m.asm
$ cat fixagesa.sh
sed -e 's/\<END\>//' \
    -e 's/si,offset/esi,offset/' \
    -e 's/di,offset/edi,offset/' \
    -e 's/push[  ]*cs/;&/' \
    -e 's/pop[   ]*ds/;&/' \
    -e 's/push[  ]*\<si\>/push esi/'  \
    -e 's/push[  ]*\<di\>/push edi/' \
    -e 's/\(pop[       ]*\)\<si\>/\1esi/'  \
    -e 's/\(pop[       ]*\)\<di\>/\1edi/' \
    -e 's/db 67h,/db /' \
    -e 's/\(mov[       ]*\)cx,\([0-9*+][0-9*+]*\)/\1ecx,\2/' \
    -e 's/\(mov[       ]*\)ecx,8000h/\1cx,8000h/' \
    -e 's/\(uses.*\)\<cx\>\(.*\)/\1ecx\2/' \
    -e 's/\(uses.*\)\<si\>\(.*\)/\1esi\2/' \
    -e 's/\(uses.*\)\<di\>\(.*\)/\1edi\2/' \
    -e 's/\<rep\>/shl esi,16\n\t\tshr esi,16\n\t\trep/' \
    -e 's/W\[si/W\[esi/' \
    -e 's/movzx si,/movzx esi,/' \
    -e 's/bx,offset T/ebx,offset T/' \
    -e 's/mctGet_PS_Cfg[0-9]/mctGet_PS_Cfg/' \
..... more expressions to fix extern definitions are omitted for space.....
 | awk 'BEGIN{f=0} !/pop[       ]*ds/{print $0; f=0} /push[     ]*ss/{f=1} /pop[
]*ds/{if(f){split($0,a,";"); print a[1] a[2]} else print $0}'
echo END
```

Sources files and how to (3)

- Assemble

  H:\>ml /c /coff m.asm

- Convert format. (you have to configure binutils to support pe-i386)

  $ objcopy -R '.debug$S' --rename-section mctIBVSegment=.text --input-target=pe-i386 --output-target=elf32-i386 m.obj m.obj.o

- Patch LinuxBISO

```
$cat agesa.patch
diff -r fb.org/src/config/Config.lb fb.agesa/src/config/Config.lb
123,124c123,124
<       depends "crt0.o $(INIT-OBJECTS) $(LINUXBIOS_APC) $(LINUXBIOS_RAM_ROM) ldscript.ld"
<       action  "$(CC) -nostdlib -nostartfiles -static -o $@ -T ldscript.ld crt0.o $(INIT-OBJECTS)"
---
>       depends "crt0.o $(INIT-OBJECTS) $(LINUXBIOS_APC) $(LINUXBIOS_RAM_ROM) ldscript.ld printk_init.o vtxprintf.o uart8250.o"
>       action  "$(CC) -nostdlib -nostartfiles -static -o $@ -T ldscript.ld crt0.o $(INIT-OBJECTS) m.obj.o printk_init.o vtxprintf.o uart8250.o"
diff -r fb.org/src/cpu/x86/32bit/entry32.inc fb.agesa/src/cpu/x86/32bit/entry32.inc
27a28,35
>       /* selgdt 0x18,flat data segment, stack seg for agesa */
>       .word   0xffff, 0x0000
>       .byte   0x0c, 0x93, 0xcf, 0x00
>       /* selgdt 0x20,flat data segment, data seg for agesa */
>       .word   0xffff, 0x0000
>       .byte   0xff, 0x93, 0xcf, 0xff
diff -r fb.org/src/mainboard/amd/serengeti_cheetah/cache_as_ram_auto.c fb.agesa/src/mainboard/amd/serengeti_cheetah/cache_as_ram_auto.c
153a154,155
> #include "northbridge/amd/amdk8/agesa_callback.c"
380c382,383
<       sdram_initialize(sysinfo->nodes, sysinfo->ctrl, sysinfo);
---
>       //sdram_initialize(sysinfo->nodes, sysinfo->ctrl, sysinfo);
>       AGESA_mctAutoInitMCT(sysinfo->nodes, sysinfo->ctrl);
```

Sources files and how to (4)

- AGESA wrapper, entry function

```
void AGESA_mctAutoInitMCT(int controllers, const struct mem_controller *ctrl) {
	struct MCTStatStruc mctstat;
	struct DCTStatStruc dctstat[8];
	char *mct = (char*)&mctstat - 0xc0000, *dct = (char*)dctstat - 0xc0000;
	int r,v,n;
	for (r=0; r<(sizeof(mctstat)/4); r++)  *((int*)&mctstat + r) = 0; // memset(0)
	for (r=0; r<(sizeof(dctstat)/4); r++)   *((int*)dctstat + r) = 0; // memset(0)

	__asm__ __volatile__ (
		"movl %0, %%edi\n\t"
		"movl %1, %%esi\n\t"
		"push %%ax\n\t"
		"mov  $0x18, %%ax \n\t"
		"mov  %%ax, %%ss\n\t"
		"mov  $0x20, %%ax \n\t"
		"mov  %%ax, %%ds\n\t"
		"subl $0xc0000, %%esp \n\t"
		"call mctAutoInitMCT\n\t"
		"addl $0xc0000, %%esp \n\t"
		"mov  $0x10, %%ax \n\t"
		"mov  %%ax, %%ss\n\t"
		"mov  %%ax, %%ds\n\t"
		"pop  %%ax\n\t"
		:
		: "r" (mct), "r" (dct)
		: "esi", "edi"
	);
}
```

Sources files and how to (5)

- AGESA wrapper, callback function utils

```
#define ASMHEAD \
            volatile unsigned int _eax,_ecx,_edx; \
             volatile struct DCTStatStruc *d; \
             volatile struct MCTStatStruc *m; \
             __asm__ __volatile__ ("movl %%eax,%0; movl %%ecx,%1; movl %%edx,%2" : "=m" (_eax), "=m" (_ecx), "=m" (_edx)); \
             __asm__ __volatile__ ("movl %%esi,%0; addl $0xc0000,%0" : "=m" (d)); \
             __asm__ __volatile__ ("movl %%edi,%0; addl $0xc0000,%0" : "=m" (m)); \

#define ASMMID  __asm__ __volatile__ ("movl %0,%%eax; movl %1,%%ecx; movl %2,%%edx" : : "m" (_eax), "m" (_ecx), "m" (_edx));

#define ASMTAIL

#define callbyasm(name) \
      void name##2(); \
      void name() { \
            __asm__ __volatile__ ("nop"); \
            __asm__ __volatile__ ("pushl $0x10; pop %ds; pushl $0x10; pop %ss; addl $0xc0000,%esp"); \
            name##2(); \
            __asm__ __volatile__ ("pushl $0x20; pop %ds; pushl $0x18; pop %ss; subl $0xc0000,%esp"); \
            __asm__ __volatile__ ("ret"); \
      } \
      void name##2()
```

Sources files and how to (6)

- AGESA wrapper, callback function sample.

```
callbyasm(mctRead_SPD) {
	volatile int ret;
	volatile char al;

	ASMHEAD;

	if (!(_ecx % 0x100)) do_printk(0, "mctRead_SPD %x node %d, ecx %x", d, d->Node_ID, _ecx);
	al = ret = smbus_read_byte((_ecx >> 8) & 0xff, _ecx & 0xff);
	if (!(_ecx % 0x100)) do_printk(0, " %x\n", ret);

	ASMMID;

	__asm__ __volatile__ ("mov %0, %%al" : : "m" (al) : "eax");

	if (ret < 0)
		__asm__ __volatile__ ("stc");
	else
		__asm__ __volatile__ ("clc");

	ASMTAIL;
}
```