A Framework for Using Processor Cache as RAM (CAR)

Eswaramoorthi Nallusamy
University of New Mexico

October 10, 2005

Acknowledgement

Prof. David A. Bader,
University of New Mexico

Dr. Ronald G. Minnich and his team,
Advanced Computing Labs,
Los Alamos National Laboratory

Overview

- Processor’s cache-as-RAM (CAR)
- Architectural Components affecting CAR
- Cache-as-RAM initialization
- Conclusion

Processor’s Cache as RAM (CAR)

- Using processor’s cache as RAM until the RAM is initialized
- CAR allows the memory init code to be written in ANSI C and compiled using GCC
  - Solves all the limitations of previous solutions comprehensively
  - Porting LinuxBIOS to different motherboards is quick
- Feasibility of Cache-as-RAM
  - Cache memory is ubiquitous across processor architectures & families
  - Cache subsystem can be initialized with few instructions
  - Cache init code doesn’t depend on any mother board components
  - Generally doesn’t require changes across a processor family
  - Can be ported easily among processor families

Architectural Components affecting CAR

- Protected Mode
- Cache Subsystem
- Inter-Processor Interrupt Mechanism
- HyperThreading Feature

Protected Mode

- Native mode of operation
- Provides rich set of architectural features including cache subsystem
- Requires, at least, a Global Descriptor Table (GDT) with entries for Code Segment (CS), Data Segment (DS) & Stack Segment (SS)
- Each GDT entry describes properties of a memory segment
- Can be entered by setting Control Register 0 (CR0) flags

Cache Subsystem

- Different levels of caches (L1, L2)
- Cache coherency maintained using Modified-Exclusive-Shared-Invalid (MESI) protocol
- Cache control mechanisms
  - Cache Disable (CD) flag in CR0 – system wide
  - Memory Type Range Registers (MTRRs) – memory region
  - Page Attribute Table (PAT) Register – individual pages

Cache Operating Modes

CD	Caching Mode & Read/Write Policy
0	*Normal Cache Mode*. Highest Performance cache operation. - Read hits access the cache; read misses may cause replacement. - Write hits update the cache; write misses cause cache line fills. - Invalidation is allowed. - External snoop traffic is supported.
1	*No-fill Cache Mode.* Memory coherency is maintained. - Read hits access the cache; read misses do not cause replacement. - Write hits update the cache; write misses access memory. - Invalidation is allowed. - External snoop traffic is supported.

Caching Methods

- Uncacheable/Strong Uncacheable
  - Memory locations not cached
- Write Combining
  - Memory locations not cached
  - Writes are combined to reduce memory traffic
- Write Through
  - Memory locations are cached
  - Reads come from cache line; may cause cache fills
  - Writes update the system memory

Caching Methods

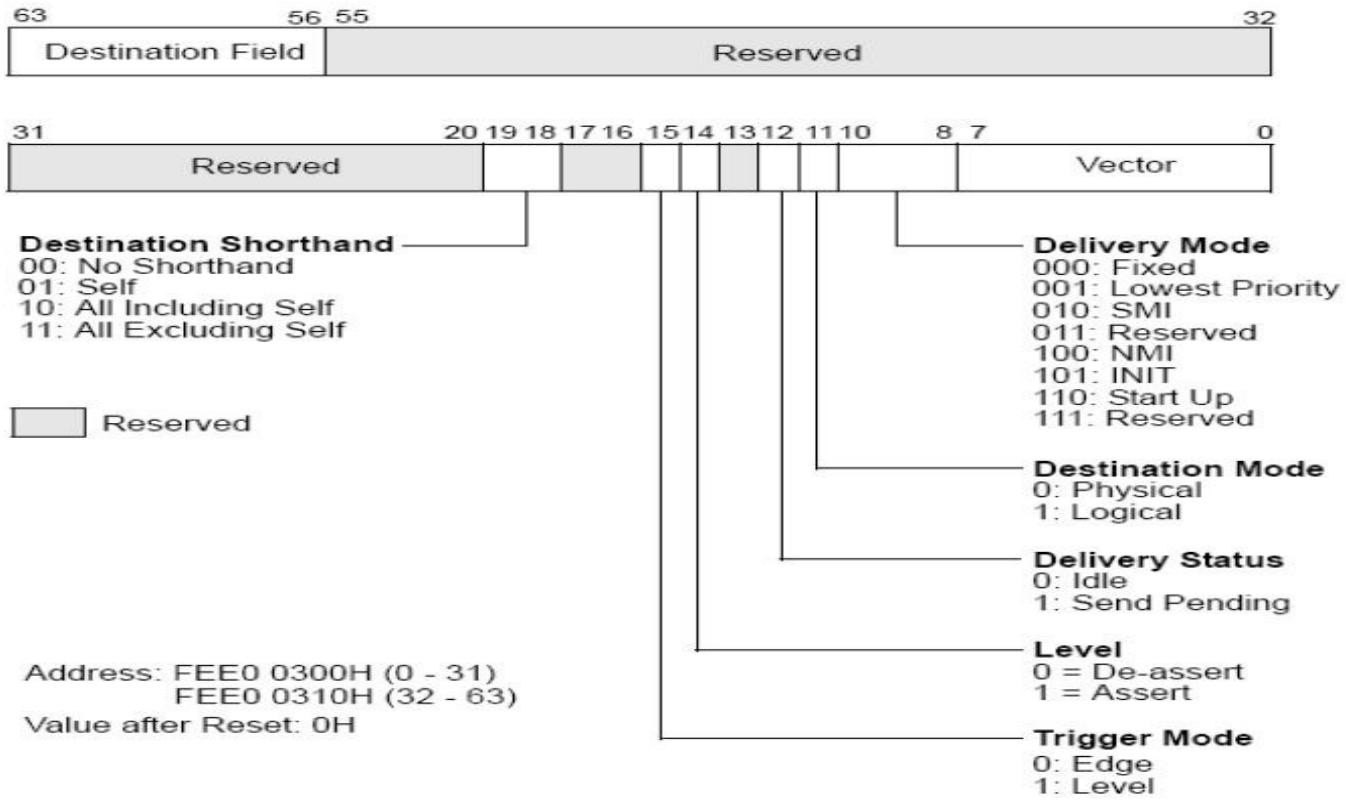
- Write Back
  - Memory locations are cached
  - Reads come from cache line; may cause cache fills
  - Writes update cache line
  - Modified cache lines written back only when cache lines need to de-allocated or when cache coherency need to be maintained

- Write Protected
  - Memory locations are cached
  - Reads come from cache line; may cause fills
  - Writes update system memory and the cache line is invalidated on all processors

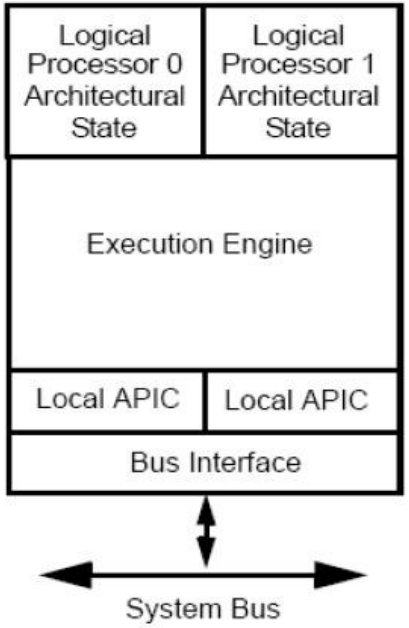
Inter-Processor Interrupt (IPI)

- Used for interrupt based communication among processors in a SMP system
- Each processor is assigned a unique ID
- IPI is raised by writing appropriate values into Interrupt Command Register (ICR)

Interrupt Command Register (ICR)



HyperThreading Feature



CAR Principle of Operation

- For the cache to act as RAM
  - The cache subsystem must retain the data in the CAR region irrespective of internal cache accesses falling within and outside the CAR region.

    (Assuming there is no external memory access activity)
  - Any memory accesses within the CAR region must not appear on the Host Bus

    as memory transactions.

CAR Principle of Operation Contd...

- Normal Cache Mode (CR0.CD=0)
  - Read/Write misses cause cache line fills
  - Cache lines replaced when there are no free cache line fills
- No-fill Mode (CR0.CD=1)
  - Read/Write misses access memory
  - Cache lines are never replaced
- An MTRR describing CAR region’s caching method is required
- Only Write-Back caching method is capable of confining Read/Write access within cache subsystem

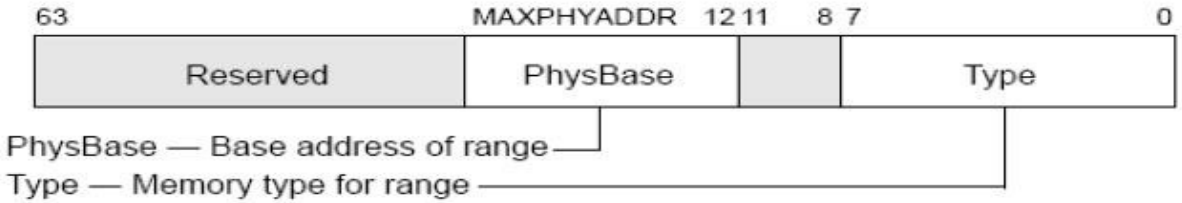
CAR Principle of Operation Contd…

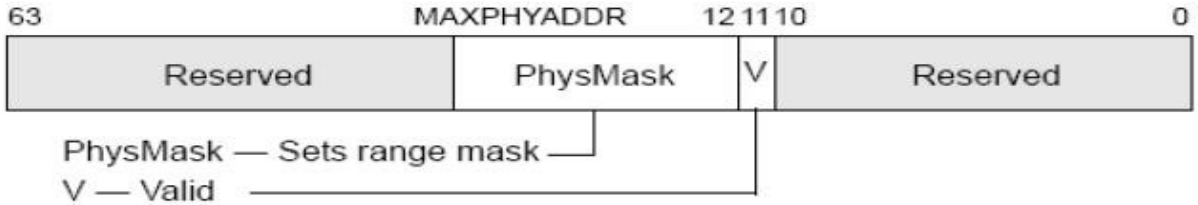
- Thus with no system memory/internal cache activity in other processors the cache subsystem operating in No-Fill Mode having at least one Write-Back CAR region behaves as RAM.
- As only the Normal Cache Mode is capable of filling the cache lines, we must fill the CAR region’s cache lines in Normal Cache Mode before entering No-Fill Mode.
- Only the Boot Strap Processor’s (BSP) internal cache is initialized to behave as CAR.

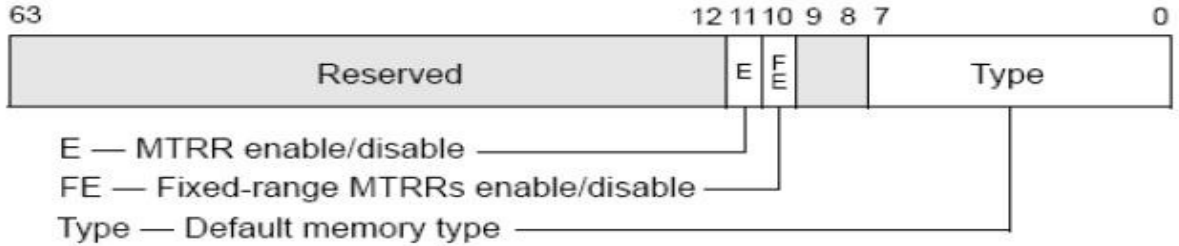
GDT and CR0 setup

- An NVRAM based flat-model GDT to access the processor address space is setup during LinuxBIOS build process
- GDTR is initialized to point to the GDT in NVRAM
- BSP enters protected mode by setting CR0.PE flag

MTRR setup


63
MAXPHYADDR
12 11
8 7
0
Reserved
PhysBase
Type
PhysBase — Base address of range
Type — Memory type for range


63
MAXPHYADDR
12 11 10
0
Reserved
PhysMask
V
Reserved
PhysMask — Sets range mask
V — Valid


63
12 11 10 9 8 7
0
Reserved
E
FE
Type
E — MTRR enable/disable
FE — Fixed-range MTRRs enable/disable
Type — Default memory type

MTRR setup Contd...

```
#define   CACHE_AS_RAM_BASE   0xF2000000
#define   CACHE_AS_RAM_SIZE   0x2000
#define   MEMORY_TYPE_WRITEBACK   0x06
#define   MTRR_PAIR_VALID   0x800
#define   MTRR_ENABLE   0x800
#define   MTRR_PHYBASE_LOW   (CACHE_AS_RAM_BASE | MEMORY_TYPE_WRITEBACK)
#define   MTRR_PHYBASE_HIGH   0x00
#define   MTRR_PHYMASK_LOW   ((~((CACHE_AS_RAM_SIZE)-0x1)) | MTRR_PAIR_VALID)
#define   MTRR_PHYMASK_HIGH   0xF
#define   IA32_MTRR_PHYBASE_REG0   0x200
#define   IA32_MTRR_PHYMASK_REG0   0x201
#define   IA32_MTRR_DEF_TYPE_REG   0x2FF
#define   IA32_MISC_ENABLE_REG   0x1A0



// Setup MTRR base
movl      $MTRR_PHYBASE_LOW, %eax
xorl      %edx, %edx
movl      $IA32_MTRR_PHYBASE_REG0, %ecx
wrmsr

// Setup MTRR mask
movl      $MTRR_PHYMASK_LOW, %eax
movl      $MTRR_PHYMASK_HIGH, %edx
movl      $IA32_MTRR_PHYMASK_REG0, %ecx
wrmsr

//Enable the MTRR subsystem
movl      $IA32_MTRR_DEF_TYPE_REG, %ecx
rdmsr
orl       $MTRR_ENABLE, %eax
wrmsr
```

Cache Subsystem Initialization

- Establish valid tags for the cache lines in the CAR region
  - Enter Normal Cache Mode (CR0.CD=0)
  - Cache lines in CAR region are marked valid by reading memory locations within CAR region
- Enter No-Fill Mode (CR0.CD=1) to avoid cache line fills/flushes
- The cache lines are analogous to uninitialized memory.
- Initialize the cache lines with some recognizable pattern for ease of debugging

Cache Subsystem Contd...

```
// Enter Normal Cache Mode
movl    %cr0, %eax
andl    $0x9FFFFFFF, %eax
invd
movl    %eax, %cr0

// Establish tags for the CAR region in the cache.
cld
movl    $CACHE_AS_RAM_BASE, %esi
movl    $CACHE_AS_RAM_SIZE/4, %ecx
rep     lodsl

// Enter No-Fill Cache Mode
movl    %cr0, %eax
orl     $0x40000000, %eax
movl    %eax, %cr0

// Initialize the CAR with recognizable patterns
movl    $CACHE_AS_RAM_BASE, %edi
movl    $CACHE_AS_RAM_SIZE/4, %ecx
movl    $0xA55A5A5A, %eax
rep     stosl

// Setup the stack
movl    $(CACHE_AS_RAM_BASE+CACHE_AS_RAM_SIZE), %esp
jmp     CAR_Init_done
```

HyperThreaded Processor Initialization

- For a HT enabled processor
  - Cache subsystem is common to both the logical processors
  - CR0 is unique to each logical processors
  - The ultimate cache operating mode is decided by the “OR” result of CR0.CD flag of each logical processors
- Only one logical processor is designated as BSP using MP init protocol
- A logical processor can’t directly access other logical processor’s resources
  - An IPI is used by the BSP to clear the CR0.CD flag of the logical processor
- An IPI handler is provided to clear the logical processor’s CR0.CD

HT Processor Initialization Contd...

```
// IPI Handler
.align   0x1000
.code16
.global   LogicalAP_SIPI

LogicalAP_SIPI:
        movl     %cr0, %eax
        andl     $0x9FFFFFFF, %eax
        movl     %eax, %cr0
        movl     $0x250, %ecx
        movl     $0x01, %eax
        xorl     %edx, %edx
        wrmsr

// Halt this AP
        cli
Halt_LogicalAP:
        hlt
```

HT Processor Initialization Contd...

```
// Check whether the processor has HT capability
            movl        $01, %eax
            cpuid
            btl         $28, %edx
            jnc         NotHtProcessor
            bswapl      %ebx
            cmpb        $01, %bh
            jbe         NotHtProcessor

// Use some common register as semaphore
            movl        $0x250, %ecx
            xorl        %eax, %eax
            xorl        %edx, %edx
            wrmsr

Retry_SIPI:
            movl        $0xFEE00310, %edi
            movb        $0x01, %al
            bswapl      %eax
            movl        %eax, %es:(%edi)
            movl        $0xFEE00300, %edi
            movl        $0x000006F9, %eax
            movl        %eax, %es:(%edi)

            movl        $0xffff, %ecx
SIPI_Delay:
            pause
            decl        %ecx
            jnz         SIPI_Delay
            movl        $0xFEE00300, %edi
            movl        %es:(%edi), %eax
            andl        $0x00001000, %eax
            jnz         Retry_SIPI

LogicalAP_SIPINotdone:
            movl        $0x250, %ecx
            rdmsr
            orl         %eax, %eax
            jz          LogicalAP_SIPINotdone
```

Conclusion

- Limitations of LinuxBIOS imposed by ROMCC is addressed
- Memory Init code can be written in ANSI C and compiled using standard C compilers
- Memory Init object code size is reduced by factor of Four