ADVANCED PROJECT CSE 523-524

Dark Memory Project, COMPAS Lab

Department of Computer Science, Stony Brook University

Document Version No 1.0

Submitted By:

Saptarshi Sen
Master of Science, Computer Science

OUTLINE

1. Introduction
2. Motivation
3. Platform Description
4. Address Mapping Scheme
    - 4.1 Xeon Series Processor Architecture
    - 4.2 Functional Blocks Description
    - 4.3 Interface for Functional Blocks
    - 4.4 Memory Layout
5. Core of Address Mapping Scheme
    - 5.1 Cbox Identification
        - 5.1.1 System Address Decoder
        - 5.1.2 DRAM interleave Register
        - 5.1.3 Illustration of Socket Level addressing
        - 5.1.4 SAD readings for ironicity
    - 5.2 Sbox/Bbox Identification
        - 5.2.1 Target Address Decoder
        - 5.2.2 TAD readings for ironicity
        - 5.2.3 Target Offset Register
        - 5.2.4 TAD offset readings for ironicity
    - 5.3 Mbox Mapper
        - 5.3.1 RIRWAYNESS register
        - 5.3.2 Paging Policy
        - 5.3.3 RIRWAYNESS readings for ironicity
            - 5.3.3.1 Independent Channel Mode
            - 5.3.3.2 Lock-Step Mode
            - 5.3.3.3 Mirror-Mode
        - 5.3.4 RIRCHNILVOFFSET register
        - 5.3.5 RIRCHNILVOFFSET readings for ironicity
    - 5.4 DIMM Table
        - 5.4.1 DIMM readings for ironicity

1 Introduction:

The issue of power consumption is vital to modern day systems: from servers deployed in large-scale data centers to mobile devices, energy consumption is an important part of system design. Power management capabilities are not only embedded in processors but also peripherals. In this project, we explore in detail one such power savings techniques based on a system peripheral, namely, memory: DRAM based devices. We demonstrate the amount of achievable power savings on a Dell M620 based on the DRAM based power optimization technique. The results are significant and it shows how such intelligence when imbibed in kernel's memory management subsystem can enormously benefit the system in terms of energy savings. There has been earlier works based on this idea. But our work differs in the fact that previous works focused only on software implementations without adequately exploring the hardware capabilities offered by modern processors.

2 Motivations:

DRAM chips come packaged in DIMM modules. These rules are governed by a memory controller which forms a part of the North Bridge. Modern processors have the north-bridge integrated into the core and therefore memory controllers are also called integrated memory controllers. The IMC activates power saving states in the DIMM's depending on the degree of idleness (memory transactions). We therefore begin our study by understanding the address mapping scheme used by the IMC so that using software techniques we can minimize the amount of memory transactions in DIMMs or specific regions within the DIMMs so that sleeping states can be artificially induced on them.

The project is closely tied to platform architecture. In Section I, we therefore begin by describing our hardware platform briefly. In Section 2, we describe the approach we followed for our investigation. In Section 3, we present our findings on the granularity of hardware boundaries over which power control can be exercised by platform software. In this section, we consider various platform configurations and show how the granularity is affected by their choice. In Section 4, we describe the power down states associated with the DRAM devices. Besides, we also try the possibility of putting the system to a further low power state by platform specific knowledge. In Section 5, we explore the possibilities whether and how software control can be used to induce these low power states. Work on this section, is in progress.

3 Platform Descriptions:

We have chosen a DellM620 as our platform. We call it ironicity. The configuration of the platform is as follows:

- Intel Xeon E5-2650 Processor with Integrated Memory Controller [x]. belonging to the Sandy Bridge family.

- Two-Socketed Intel C602 Chipset

- 64 GB system RAM with Dual Ranked DIMMs of 8GB capacity each .(See below figure for DIMM layout)

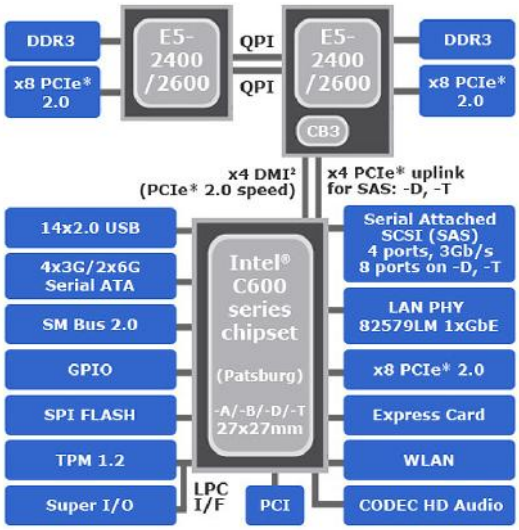


Fig 1 Intel C602 Chipset[3]

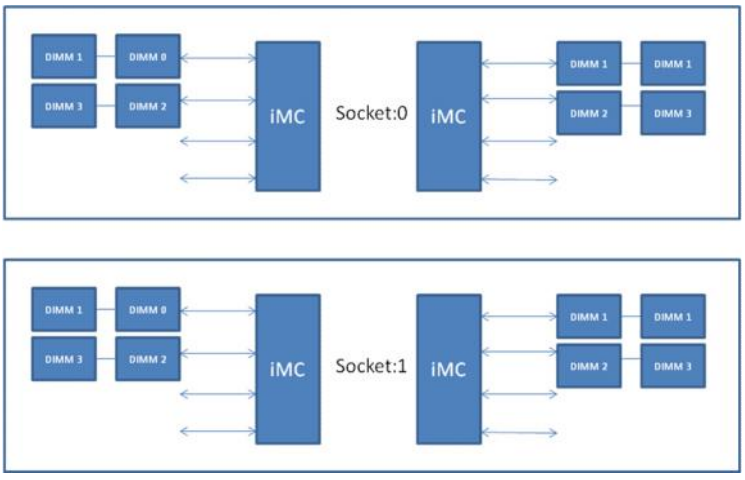


Fig 2 DIMM Layout

4. Introduction to Address Mapping Scheme

4.1 Intel Address Mapping Scheme:

The address mapping scheme for Intel's processors varies across processor families. The simplest address mapping scheme can be found for Atom processors which is hardwired. However in the case of server based processors, there are multiple address mapping per processor supporting various memory configurations targeted at decreasing latency and improve RAS features. Below we explore the address mapping feature for Sandybridge.

4.2 Xeon Series Processor Architecture:

In this section, we understand how the system memory layout is interpreted by the platform. It is wrong to presume that the iMC is solely responsible for the address mapping policies. Below we identify the basic components which are involved in the platform's address mapping policy and the associated hierarchy. This is a functional block diagram of Intel's Xeon E5 processor.

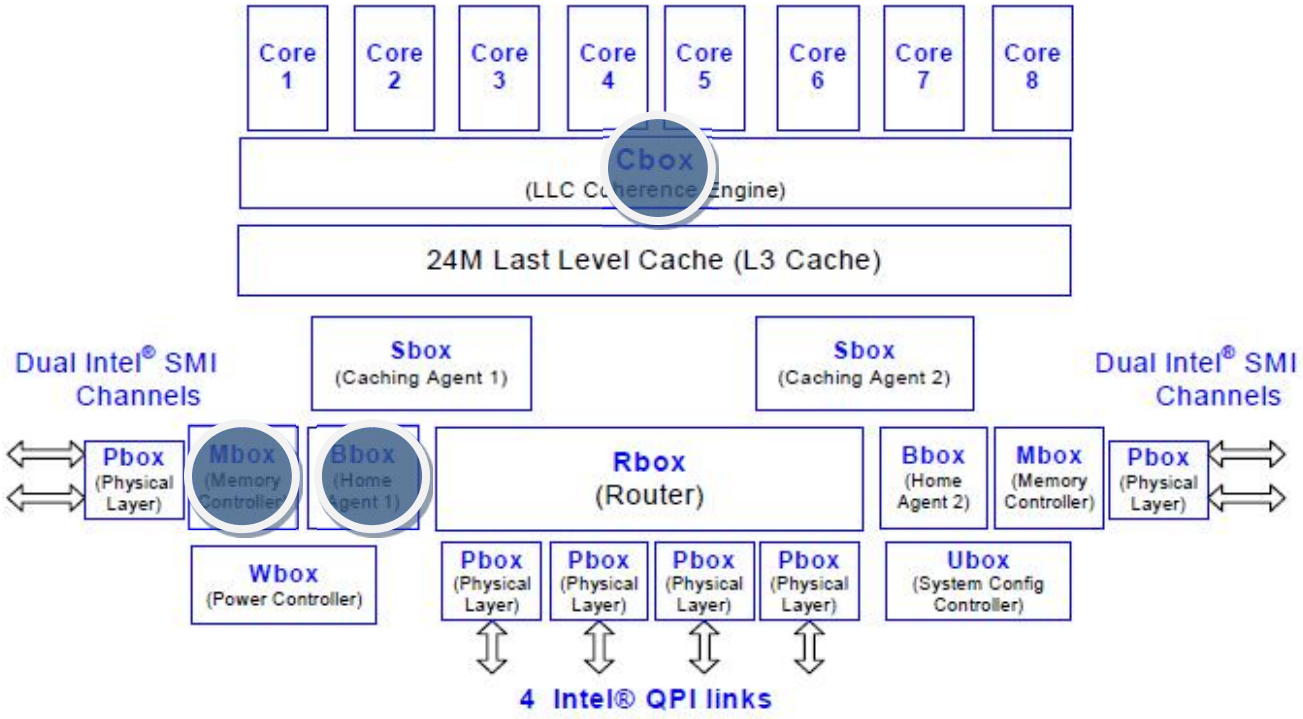


Fig 3 Xeon Architecture

4.3 Functional Blocks Description:

Functional Block	Description
Core	Processing unit
Cbox	Interconnect between a core and last-level cache. This houses the *Source Address Decoder*. The highest in the decoding hierarchy.
Sbox	A bridge between the Core and an Intel Quick path Interconnect Router. It takes core requests and transmits them to the Rbox or forwards them to the Bbox. It snoops responses from the Rbox and forwards them to Cbox or BBox.
Bbox	Also known as the Home Agent, the Bbox maps physical addresses to *Channels*. The Home Agent is the owner of a portion of memory and responsible for satisfying caching agent requests. The Sbox makes memory requests to the Bbox.
Mbox	The Mbox represents the Integrated Memory Controller.
Wbox	This is the Power control Unit. This unit manages power state transitions and voltage/frequency operating points.

Table 1 Functional Blocks Description

The below figure describes the address mapping hierarchy in terms of the functional blocks in the processor.

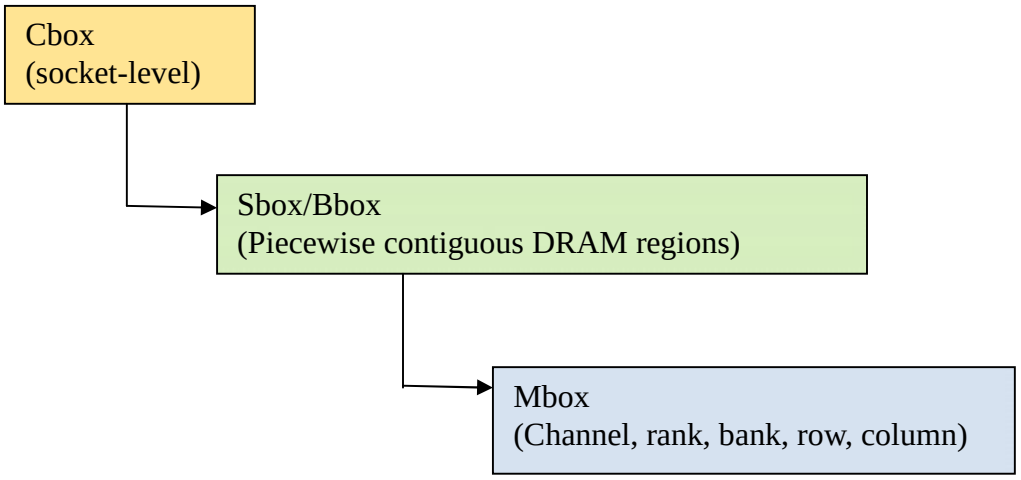


Fig 4 Address mapping hierarchy in terms of Functional Blocks

4.4 Interface for Functional Blocks (PCI Devices):

Each of the Boxes is implemented as a PCI Device and mounted on a local PCI bus. The address mapping information is grouped under registers collectively termed as **the Processor Uncore Registers**. There are two sets of such registers belonging to each of the sockets. The CSR of these devices are already mapped in the memory. Using a standard PCI bus topology search, we discover that these registers reside on Bus No 64 and 127. This information can be obtained by with queries based on the Device ID which are standard across a family.

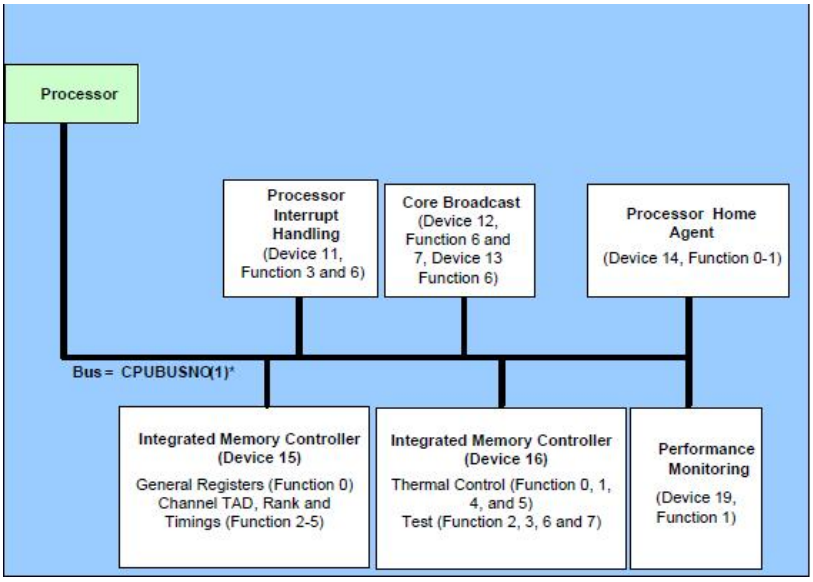


Fig 5 Intel Uncore Registers [1]

Below we enlist bus addresses of the relevant components in the processor belonging to the PCI Uncore devices. These are specific to the processor:

Component	Device ID	Device	Function	Description
PCU	3cc0,3cc1,3cc2,3cd0	10	0-3	
CBox	3cf4 3cf6	12	6-7	System address Decoder
	3cf5	13	6	System address Decoder
Bbox	3ca0,3c46	14	0-1	Home Agent
Mbox	3caa 3cab,3cac,3cad,3cae	15	2-6	Channel Target Address Decoder Registers

Table 2 PCI Device Addresses for Uncore Register Section

4.5 Memory Configuration:

There are two types of decoders which are used in the Processor. They are the DRAM decoder and the IO decoder (Memory Type MMIO, MMCFG). Whenever any memory request comes to the processor through Cbox, parallel searches are made to the two decoders. There is a priority maintained between these two decoders. i.e. result of one can override the other. Generally the IO decode has more priority than the DRAM decoder. If there is any match in the lookup in the IO decoder, the requests are forwarded to the UBox. The IO memory layout as a part of the address space is required for the BIOS to program the CSRs of Bbox so that DRAM regions do not get unused because of the holes in the address space.

This following gives an insight into the possible IO regions in a processors memory layout.

IO region	Description
Low Memory DRAM region	0-640k
VGA/CSEG	MMIO region overlaid with DRAM region This DRAM region is accessible only in SMM mode by the SMI handler
PAM	768K-1M This is a legacy BIOS ROM area in MMIO. It serves a s a faster ROM storage area. It is also simultaneously overlaid with DRAM areas. The 14-17th bit define the PAM segments. Each PAM segment has two configuration bits which decide if the requests map to a PAM target or not.
BIOS	786k-1M This region targets the BIOS flash ROM device. This region must be disabled before the PAM region.
OS region	1M-4G-64M
	TOLM: From 1M till TOLM, we call it the DRAM Low Memory Region. There are no memory holes within this region.
PCIE	Beyond TOLM, we house a PCI-Express configuration register space. This is an MMCFG space. The range of this region is 256 MB and resides below the 4GB memory. This region overrides enabled DRAM address region.
MMIOL	This region is dedicated to MMIO and covers the region immediately adjacent to the configured PCIe segment to 4G-64M. This is a variable length region in the IO decoder.
Global CPU CSRs	4G-64M – 4G-48M. This region is used to access CPU uncore CSRs via MMIO
Global Intel Chipset CSRs	4G-48M – 4G-32M This region is used to access Intel Chipset Uncore CSRs.
ICH	4G-19M – 4G-18M

	This region covers the HPET via MMIO. The accesses are sent to the legacy chipset.
IOAPIC	4G-19M – 4G-18M This is a 1-MB range used to map the IOxAPIC Controller Registers via MMIO
Flash	4G-16M..4G This region is used for BIOS/firmware. The access to this region are forwarded to the chipset if flash is attached. The attribute to this space is MMIO.
Extended Memory Region	This is DRAM mapped above 4GB
Recovered DRAM	DRAM which would have been located at addresses just below 4 GB are relocated in an address region above the 4 GB boundary.
MMIOH	If more MMIO space is required than is available below 4 GB, then DRAM entries are used to configure it above the top of DRAM. These entries are not interleaved.
Additional PCI-E Segments	
Protected Global CPU/Chipset CSRs	2^44-64G - 2^44

Table 3 Description of IO Regions in Intel Processors

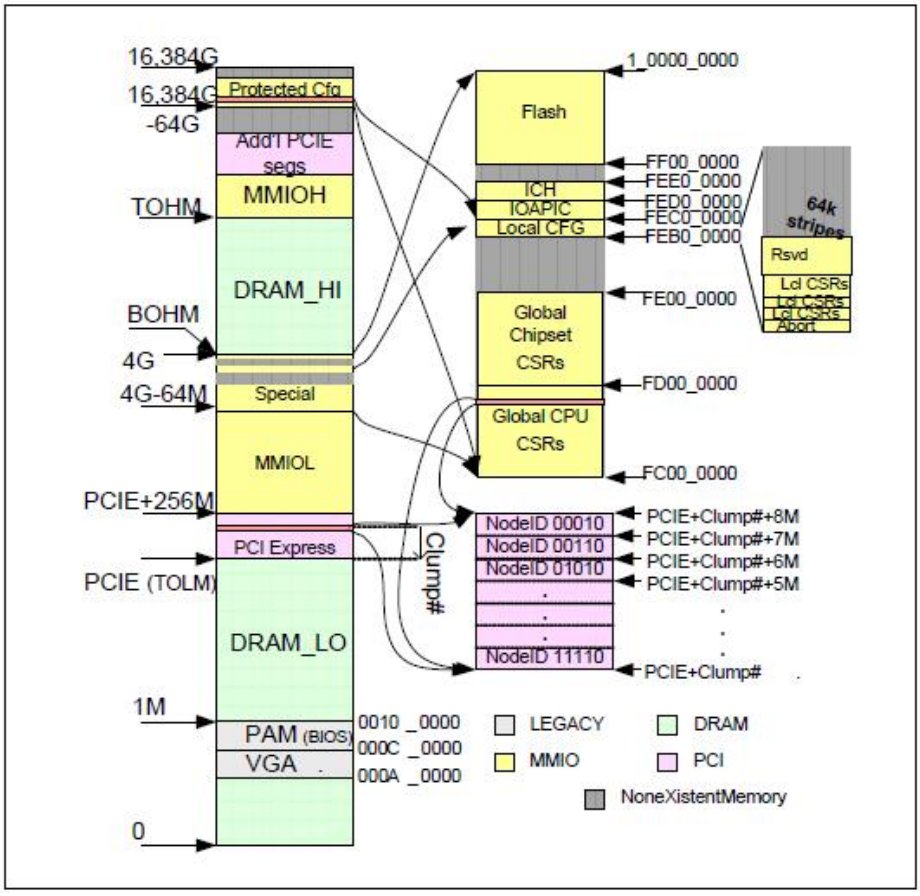


Fig 6 Memory Layout showing DRAM and IO Regions [2]

5. Core of Address Mapping:

The system implements memory addressing scheme based on two parameters:

a) Memory range
b) Nature of Interleaving for that range

Memory configuration settings in the BIOS affect these parameters in a variety of ways which we shall discuss below. In our discussion, we analyze the address mapping scheme for the following memory configuration:-

Interleaving Configuration:

a) NUMA
b) SMP

Channel Configuration:

a) Independent Channel Mode
b) Lock-Step Mode
c) Mirror-Mode Mode

Lock-Step Mode: This is an Advanced ECC-Mode. In lock-step mode, each write is spread across two channels, so on a write 32 bytes are written to one channel and 32 bytes written to the other channel. This means that on a read, 64 bytes have to be read by also reading both channels (32 bytes from each) and combining the data. In this mode, a read and writes are bifurcated to a pair of channels. For Xeon E5, the following lock-step modes are documented as valid:

A) Channel : 0 and Channel: 2
B) Channel : 1 and Channel: 3

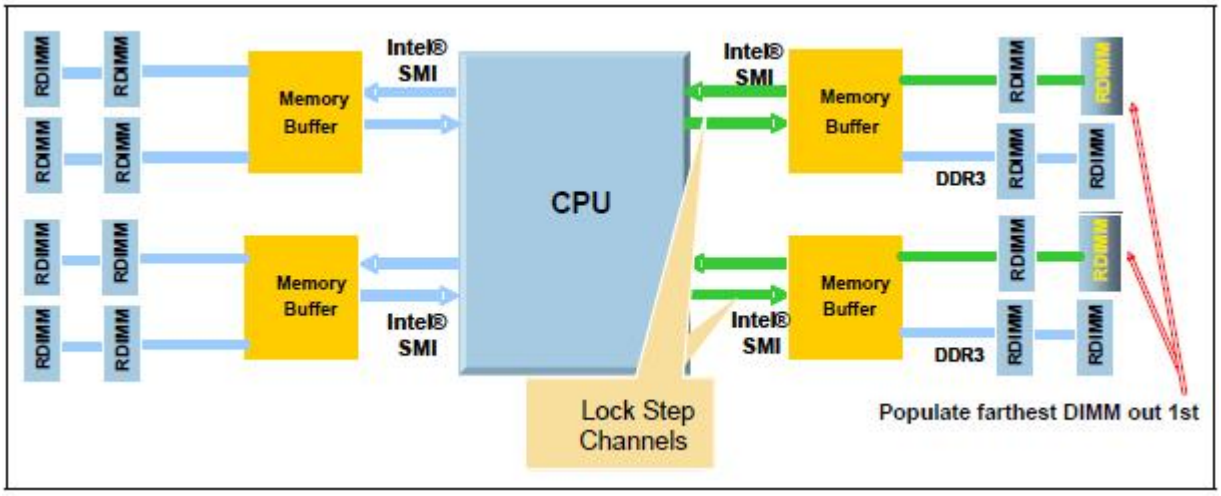


Fig 7 Lock-Step Mode Configuration [2]

Mirror-Mode: In this mode, intra-socket channel mirroring takes place. In mirror mode, on a write, both channels get the same 64-byte write. On a read, the system can pick one of the channels to read 64 bytes from. For Xeon E5, the following mirror modes are documented as valid:

A) Channel 0 mirrored with Channel 2

B) Channel 1 mirrored with Channel 3

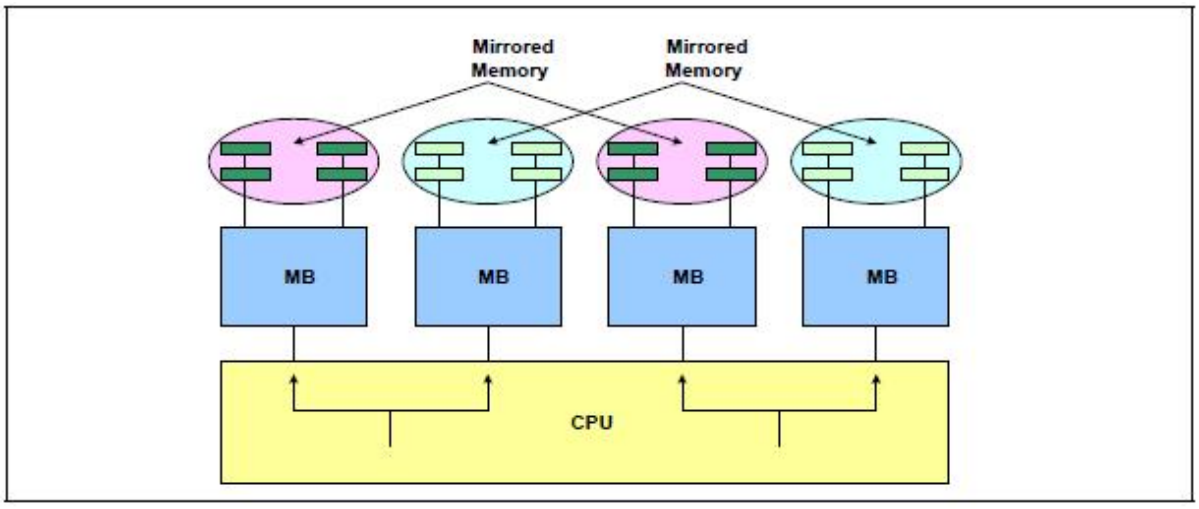


Fig 8 Mirror-Mode Configuration[2]

Configuration Space Registers:

Each functional block has PCI based CSRs. We do a search of the PCI bus to identify and read these CSRs. Below we describe the address decoding scheme based on information obtained from the CSRs of these devices.

5.1 CBox Identification:

5.1.1 System Address Decoder (SAD):

Socket level addressing is based on identification of Cbox. Cbox identification is based on entries in a group of registers collectively called the SAD. Each SAD entry identifies a DRAM region with a unique interleaving. For SandyBridge, there are a maximum of 10 SAD entries of which only a few are valid. SAD entries with repeatable fields are treated as invalid. The following describes a SAD entry which is a 32 bit field.

RV (31-26)	LIMIT (25-6)	RV (5-4)	DRAM_RULE (3-2)	INTERLEAVE (1)	RULE_ENABLE (0)

SAD_LIMIT: This field is the top limit address for that range. The granularity is 256 MB.

DRAM_RULE: The target type for this address range. This can be MMIO, MMCFG, and DRAM.

INTERLEAVE_MODE: This bit selects an interleaving mode between a low order interleave and a hemisphere-interleave mode (described later). Based on the mode, we generate an index which is used as a key to an DRAM interleave register for identifying the SAD Target or the Cbox to which the address belongs. This is the basis for socket level interleaving

This is how the 3-bit field is generated as per the interleaving mode is selected in the SAD entry.

Mode	**Tgtsel**	**Tgtidx[2]**	**Tgtidx[1]**	**Tgtidx[0]**
Mixed	0	addr[8]^addr[18]	addr[7]^addr[17]	addr[6]^addr[16]
Low-Order	1	Addr[8]	Addr[7]	addr[6]

Table 4 Interleaving Mode (Not to be confused as interleaving bits)

RULE_ENABLE: This field tells whether the SAD entry is valid for decoding or not.

Unlike DRAM decoders, IO decoders have fixed targets. The targets in SAD entries are configurable across reboots whereas in the case of IO decoders it is not.

The following is an example of the SAD entries for NUMA configuration in independent channel mode:

SAD RULE	SAD LIMIT
1	33472[MB]
2	66240[MB]

Table 5 SAD Register Sample Contents

5.1.2 DRAM interleave register:

Each SAD entry is associated with a DRAM Interleave register which identifies the Cbox. The following is a description of the interleave register.

Pkg 8	Rv	Pkg 7	Rv	Pkg 6	Rv	Pkg 5	Rv	Pkg 4	Rv	Pkg 3	Rv	Pkg 2	Rv	Pkg 1

Intel Xeon E5 has a maximum of 8 SAD targets. This is because of the fact that this processor can be connected to a maximum of 8 sockets. Although for our platform this number is limited to 2 (this number can be expanded by using an External Node Controller or an IO Hub)

Since we have less than eight nodes, so we have repeating entries in the package.

The following is an example of the DRAM interleaves register entries for NUMA configuration in independent channel mode:

SAD ENTRY	SAD INTERLEAVE
SAD RANGE #1	0
SAD RANGE #2	0x9090909

Table 6 DRAM Interleaving Register Sample Contents

5.1.3 Illustration of socket level addressing:

The address bits which are responsible for socket interleaving is known by collating information from the SAD rule and DRAM interleave register.

Fig 9 Memory Interleaving Configuration

For example, we have two memory configurations, one in which 32 GB is individually allotted to two sockets. This memory configuration is also termed as the NUMA. For this configuration, we have two SAD entries. The first SAD entry has only entries for Package 1 in the SAD Target register irrespective of the interleaving mode used in the SAD entry. Likewise, the second SAD entry has only entries for Package 2 in the SAD target. For the second configuration also known as SMP, there is only a single SAD entry and the SAD Target List consists of repeating alternate entries for Package 0 and Package 1.

However it is from this SAD entry, we can infer that the finest granularity of interleaving possible is 64 bytes.

5.1.4 SAD readings for iconicity:

The pseudo code for identifying Socket Level mapping is in Appendix A

Below we consolidate SAD readings that we obtained in our platform for the following configuration. The sockets have been configured for independent channel mode

a) With memory interleaving (SMP)

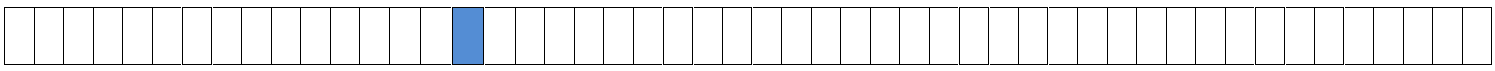
b) With memory interleaving disabled (NUMA)

CONFIGURATION	**SAD**		
NUMA	NO OF RULES : 2		
		SAD_RANGE	Interleaving Bit
	SAD_ENTRY #1	0-33.536 GB	Pa<34>
	SAD_ENTRY #1	33.537-66.304 GB	Pa<34>
SMP	NO OF RULES :1		
	SAD_ENTRY #1	0-66.304 GB	Pa<7>

Table 7 Consolidated SAD Readings for ironicity

NB: Please note that the number of interleaving bits shown here is only valid for ironicity. This shall change if the number of connected sockets increase and secondly, if the amount of physical memory allocated per socket changes. That is why the SAD and DRAM interleave registers are required to decide the interleaving at run-time.

For NUMA, socket-level interleaving bits are shown as follows:-



For SMP, socket-level interleaving bits are shown as follows:-

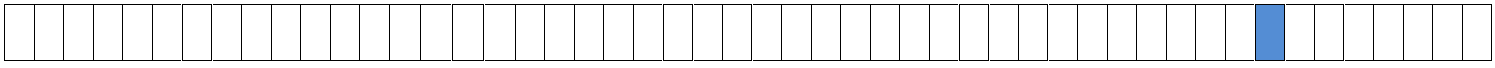


Fig 10 Address Bit positions responsible for Interleaving across sockets

5.2 SBox/Bbox Identification:

Once we identify the Cbox, we identify the next in the memory addressing hierarchy, Sbox/Bbox.

We have seen that each package field in the SAD Interleave Register is a 3 bit field Tgtidx [0:2]. This package ID is different from Node ID. Each Intel CPU has a 5 bit field Node identification listnid[0:4]. This is computed from the package id as follows:

listnid[4] = tgtlist[{tgtidx[2:0], 2b'11}]
listnid[3] = tgtlist[{tgtidx[2:0], 2b'10}]
listnid[2] = tgtlist[{tgtidx[2:0], 2b'01}]
listnid[1] = tgtlist[{tgtidx[2:0], 2b'00}]

Since Sandy Bridge can support a maximum of eight sockets. So, the most significant 3 bits of the Node ID represent the Cbox E.g. 001xx, 000xx. Thus listnid [4:2] does the Cbox identification. Within each Cbox there are four devices, which are used to identify based on the last two lower bit fields.

a) Intel ChipSet (00)
b) Two Sets of Home Agents and Caching Agents, B0/S0 and B1/S1 (01 and 11)
c) The configuration agent, Ubox (10) (IO decoding).

Once the corresponding Cbox is selected based on the Package, the next thing forward is to identify the SBox/Bbox. A combination of listnid[0] and listnid[1] identifies the Sbox/Bbox.
The messages routed through the QPI interconnect use this Node IDs for assignments.

5.2.1 Target Address Decoder:

Once the SBox is identified, we probe the associated Target Address Decoder register (TAD). The set of addresses entering a particular Bbox define an address space with holes (as seen in the memory configuration layout). So it uses the TAD information to glue the interleaved address from the Cbox targeting a particular Bbox together into a contiguous address space for DRAM regions.

The main function of the TAD is twofold:

a) First, to squeeze out the interleave bits due to socket interleaving as well as channel interleaving which were actually used to determine the socket and the home node.
b) Second, to remove the regions in the address space which overlap with IO regions. The holes are also identified as address type NXM (or non-existent memory)

Below we describe the target address decoder register.

TAD_LIMIT	TAD_SKT_WAY	TAD_CH_WAY	TGT_CH_4	TGT_CH_3	TGT_CH_2	TGT_CH_1

TAD LIMIT: The TAD limit is a region of global memory which is contiguous (not channel address space) in the terms that there are possible address holes in that region. Ideally, TAD limits are supposed to present limits on both existent as well as non-existent memory. But register information shows only TAD limits associated with existent memory. For each such limit, there is
a tad offset which is a value representing what is to be subtracted from the system address.

Below is an example how the TAD_LIMITS are computed. Please note that values for imit0 and limi2 are known apriori from the address space layout information. The value has a 64 MB granularity. There are a maximum of 12 TAD Entries for SandyBridge.

ENTRY	**LIMIT**	**BASE ADDRESS**	**INTERLEAVE WAYS**	**NXM**
0	Limit0	Do not Care(DC)	DC	1
1	Limit1	2s complement of((limit0+1) >> (interlv1-1))	b0001	0
2	Limit2	Do not Care	DC	1
3	Limit3	2s complement of((limit2+1)>>(interlev3-1)))+((limit1-limit0)>>(interlv1-1)	b0100	0

Table 8 TAD Limits Computation Example[2]

TAD_SKT_WAY: The socket interleaves way is a log of the number of sockets participating in the interleaving of the memory region specified by the TAD_LIMIT. This is inter-related to the SAD Target Interleave List.

TAD_CH_WAY: The channel interleaves way is a log of the number of channels among which the memory region has to be divided. This is a per socket.

TAD_CH_TGT3/TGT2/TGT1/TGT0: The target channel ID. This register represents the channels participating in the interleaving for that TAD RANGE. A bit set in this field implies that the channel is active.

5.2.2 TAD readings for ironicity:

Below we tabulate relevant information from TAD entries for the following configuration. The sockets have been configured for independent channel mode. a) With memory interleaving (SMP) b) With memory interleaving disabled (NUMA)

Although there can be maximum of 8 TAD entries for SandyBridge, we enlist the valid TAD entries.

NUMA	TAD	SOCKET NO 1				
			TAD_LIMIT	CHN_WAY	SKT_WAY	INTERLEAVE_BIT
		TAD_ENTRY	3.328 GB	2	0	Pa<7,8>
		TAD_ENTRY	33.536 GB	2	0	Pa<7,8>
		SOCKET NO 2				
		TAD_ENTRY	66.304 GB	2	0	Pa<7,8>
SMP	TAD	SOCKET NO 1				
		TAD_ENTRY	3.072 GB	2	1	Pa<8,9>
		TAD_ENTRY	3.328 GB	2	1	Pa<8,9>
		TAD_ENTRY	66.048 GB	2	1	Pa<8,9>
		TAD_ENTRY	66.304 GB	2	1	Pa<8,9>
		SOCKET NO 2				
		TAD_ENTRY	3.072 GB	2	1	Pa<8,9>
		TAD_ENTRY	3.328 GB	2	1	Pa<8,9>
		TAD_ENTRY	66.048 GB	2	1	Pa<8,9>
		TAD_ENTRY	66.304 GB	2	1	Pa<8,9>

Table 9 TAD Entries for a Socket in two different configurations

5.2.3 Target Offset Register (TAD offset Register):

Unlike SAD interleave register, there is no interleave register for TAD entries. However there is an associated TAD OFFSET register whose purpose is quite different to that of interleaving. The TAD offset register contains an offset value per TAD range which has to be subtracted from the system address. The requirement of the offset comes in the context of overlapping of IO address space. Besides, the computation of the offset is not explicitly explained in the manual but however relevant insight can be found on this from the table discussed in section. The address thus obtained forms a contiguous address space.

To squeeze out the socket interleaving in a tad range and convert the address into local channel address space, we use the tad offset for that range in the following way.

addr = adrr - tad_offset[tad_range]

The channel identification is done by dividing addr with total_way(= number of channels participating in interleave * number of sockets participating in interleave)

channel_no = addr % total_way

5.2.4 TAD OFFSET readings for ironicity

The pseudo code for identifying Channel Level mapping is in Appendix A.

The following table lists TAD OFFSETS that we have obtained on our platform for two memory configurations. This holds for channel configured in independent mode.

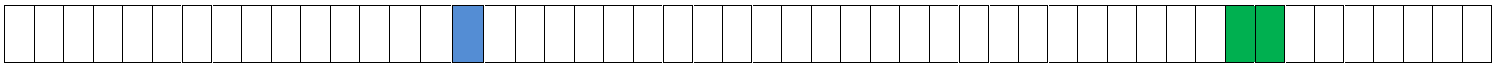
NUMA	TAD LIMIT	SOCKET 1	
		TAD #0 OFFSET	CH 0 : 0 GB
			CH 1 : 0 GB
			CH 2 : -

			CH 3 : -
		TAD #1 OFFSET	CH 0 : 0.768 GB
			CH 1 : 0.768 GB
			CH 2 : -
			CH 3 : -
		SOCKET 2	
		TAD #0 OFFSET	CH 0 : 33.536 GB
			CH 1 : 33.536 GB
			CH 2 : -
			CH 3 : -
SMP	TAD LIMIT	SOCKET 1	
		TAD #0 OFFSET	CH 0 : 0 GB
			CH 1 : 0 GB
			CH 2 : -
			CH 3 : -
		TAD #1 OFFSET	CH 0 : 1.5 GB
			CH 1 : 1.5 GB
			CH2 : -
			CH3 : -
		TAD #2 OFFSET	CH 0 : 0.512 GB
			CH 1 : 0.512 GB
			CH 2 : -
			CH 3 : -
		TAD #3 OFFSET	CH 0 : 33.536 GB
			CH 1 : 33.536 GB
			CH 2 : -
			CH 3 : -
		SOCKET 2	
		TAD #0 OFFSET	CH 0 : 0 GB

			CH 1 : 0 GB
			CH 2 : -
			CH 3 : -
		TAD #1 OFFSET	CH 0 : 1.536GB
			CH 1 : 1.536 GB
			CH 2 : -
			CH 3 : -
		TAD #2 OFFSET	CH 0 : 0.512 GB
			CH 1 : 0.512 GB
			CH 2 : -
			CH 3 : -
		TAD #3 OFFSET	CH 0 : 33.536 GB
			CH 1 : 33.536 GB
			CH 2 : -
			CH 3 : -

Table 10 TAD Offset Readings for two different memory configuration

For NUMA, channel interleaving bits are shown as follows:-



For SMP, channel interleaving bits are shown as follows:-

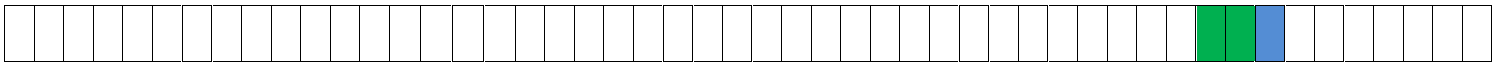


Fig 11 Address Bit positions responsible for Interleaving across channels

NB: Please note that the position of the channel interleaving bits is valid only for ironicity. It shall change depending on the number of sockets and number of channels per socket. The only invariant is that the smallest level of interleaving granularity is 64 bytes which is in line with the cache line size.

5.3 Mbox Mapper:

The Bbox issues read and write requests to the Mbox. The Mbox is the Integrated Memory Controller which is responsible for further address decoding.

This is done as follows:

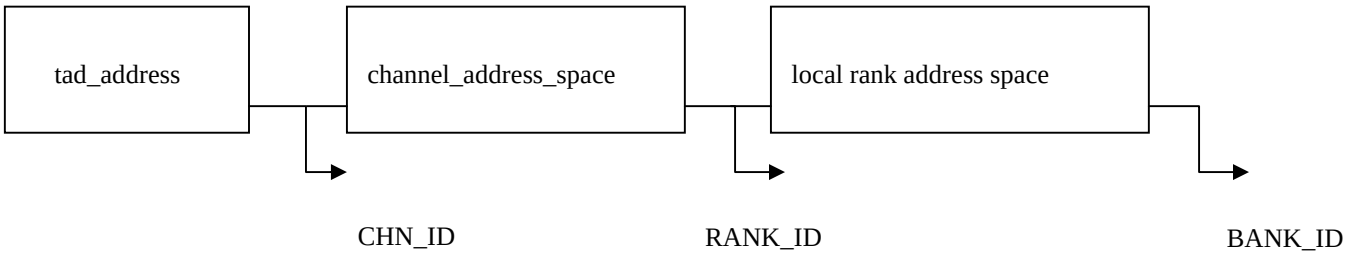


Fig 12 Address Decoding Logic in terms of address space

5.3.1 RIRWAYNESS register

Each of the memory controllers are connected via two SMI (or fully buffered) links to DIMMs. For our platform, each SMI Link can connect up to a maximum of three DIMMs. The scheduler views each DDR bus as a single DIMM. Thus ranks present on a DDR bus seem equivalent to logical ranks. This interleaving among the logical ranks is done based the RIRWAYNESSLIMIT Register described below.

RIR_VAL	RV	RIR_WAY	RV	RIR_LIMIT	RV

RIR_VAL: This is a one-bit field which if this is a valid rank rule.

RIR_WAY: The rank interleaves wayness is the log of the number of ranks this address range shall be interleaved with. The ranks described here is the total number of logical ranks per channel. There can be a maximum of 8 Ranks per channel

RIR_LIMIT: The limiting address for that range in the channel addresses space. The granularity is in the value of 512 MB. SandyBridge has a maximum of 5 Rank Limits.

5.3.2 Paging Policy:

Apart from the number of logical ranks participating in the interleaving, the granularity of rank

interleaving is dependent on the paging policy also. The IMC maintains page tables, and maintains per page paging policy – open or closed, depending on the access frequency of the page. Page Policy and Timing Parameter Register represents this.

The default paging mode for our platform can be obtained from the DIMM MEMORY TECHNOLOGY REGISTER.

For closed paging mode, we do the following:

rank_addr = channel_addrr >> rir_wayness

For open paging mode, we do the following:

rank_addr = (channel_addrr << 7) >> rir_wayness

On our platform, we have an open page policy.

5.3.3 RIRWAYNESS register readings for ironicity:

The pseudo code for identifying Rank Level mapping for different Channel Configurations are given in Appendix A.

5.3.3.1 Independent channel mode:

NUMA	RIR	SOCKET NO 1			
			RIR_LIMIT	RIR_WAYS	INTERLEAVE BITS
		RIR_ENTRY	15872 MB	2	Pa<16,17,18> (Open Page) Pa<9,10,11> (Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		SOCKET NO 2			
		RIR_ENTRY	15872 MB	2	Pa<16,17,18> (Open Page) Pa<9,10,11> (Closed Page)
		RIR_ENTRY	-		-

		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
SMP	RIR	SOCKET NO 1			
		RIR_ENTRY	15872 MB	2	Pa<17,18,19> (Open Page) Pa<10,11,12>(Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		SOCKET NO 2			
		RIR_ENTRY	15872 MB	2	Pa<17,18,19> (Open Page) Pa<10,11,12>(Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		

Table 11 RIRWAYNESS register entries for two different memory configurations in Independent Mode

5.3.3.2 Lock-Step Mode:

The RIR entries for the Lock-step mode are tabulated as follows:

NUMA	RIR	SOCKET NO 1			
			RIR_LIMIT	RIR_WAYS	INTERLEAVE BITS
		RIR_ENTRY	32256 MB	2	Pa<16,17,18> (Open Page) Pa<9,10,11> (Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		SOCKET NO 2			
		RIR_ENTRY	32256 MB	2	Pa<16,17,18> (Open Page) Pa<9,10,11> (Closed Page)
		RIR_ENTRY	-		-
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
SMP	RIR	SOCKET NO 1			
		RIR_ENTRY	32256 MB	2	Pa<17,18,19> (Open Page) Pa<10,11,12>(Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		SOCKET NO 2			
		RIR_ENTRY	32256 MB	2	Pa<17,18,19> (Open Page) Pa<10,11,12>(Closed Page)
		RIR_ENTRY	-		

		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		

Table 12 RIRWAYNESS register entries for two different memory configurations in Lock-Step Mode

5.3.3.3 Mirror-Mode:

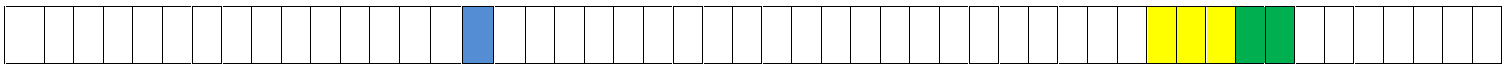
The RIR entries for the mirror mode are tabulated as follows:

NUMA	RIR	SOCKET NO 1			
			RIR_LIMIT	RIR_WAYS	INTERLEAVE BITS
		RIR_ENTRY	15872 MB	2	Pa<16,17,18> (Open Page) Pa<9,10,11> (Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		SOCKET NO 2			
		RIR_ENTRY	15872 MB	2	Pa<16,17,18> (Open Page) Pa<9,10,11> (Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
SMP	RIR	SOCKET NO 1			
		RIR_ENTRY	15872 MB	2	Pa<17,18,19> (Open Page) Pa<10,11,12>(Closed Page)
		RIR_ENTRY	-		

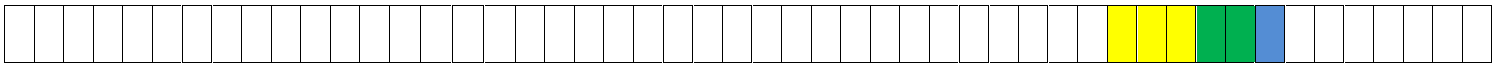
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		SOCKET NO 2			
		RIR_ENTRY	15872 MB	2	Pa<17,18,19> (Open Page) Pa<10,11,12>(Closed Page)
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		
		RIR_ENTRY	-		

Table 13 RIRWAYNESS register entries for two different memory configuration in Mirror-Mode

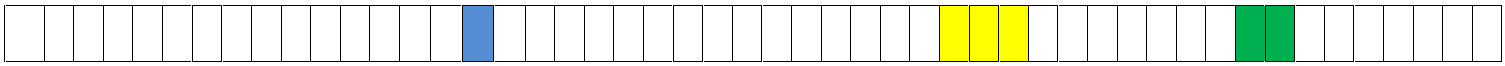
For NUMA, rank interleaving bits in Closed Page Policy are shown as follows:-



For SMP, rank interleaving bits in Closed Page Policy are shown as follows:-



For NUMA, rank interleaving bits in Open Page Policy are shown as follows:-



For SMP, rank interleaving bits in Open Page Policy are shown as follows:-

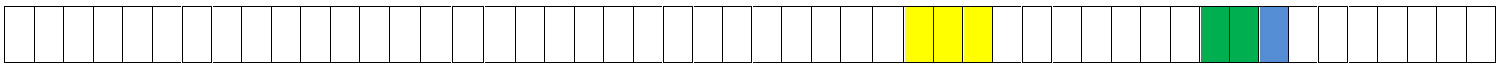


Fig 13 Address Bit positions responsible for Interleaving across ranks

NB: Please note that the position of the rank interleaving bits is valid only for ironicity. It shall change depending on the number of sockets and number of channels per socket. The only invariant is that the smallest level of interleaving granularity is 64 bytes which is in line with the cache line size.

5.3.4 RIRCHNILVOFFSET:

Like TAD entries, there is no interleaving register for RIR entries. The rank interleaving is uniform based on the number of participating ranks which is indicated by the RIR_WAY field. The logical rank is identified as follows:

rank_no = rank_addr % (1<<RIR_WAY)

Based on the logical rank, we compute the physical rank from the RIRCHNILVOFFSET register.
The RIRCHNILVOFFSET register contains the mapping of the logical rank to physical rank.
For each RIR_LIMIT, we have a maximum of 8 possible RIR mappings. From the one-to-one mapping of logical rank to the physical rank, we obtain the physical rank no.

RV	RIR_RNK_TGT	RV	RIR_OFFSET	RV

RIR_RNK_TGT0: This field represents the physical rank number corresponding to the logical rank to which this mapping belongs.
RIR_OFFSET0: The RIR_OFFSET is equivalent to a TAD_OFFSET. But it is always zero.

5.3.4 RIRCHNILVOFFSET readings for ironicity:

Below we tabulate the register entries obtained per RIR range.

NUMA	RIR_LIMIT#1	RIR_OFFSET#0	LOGICAL RANK 0 : 0 GB	LOGICAL RANK 0 : PHY RANK :0
		RIR_OFFSET#0	LOGICAL RANK 1 : 0 GB	LOGICAL RANK 1 : PHY RANK :4
		RIR_OFFSET#0	LOGICAL RANK 2 : 0 GB	LOGICAL RANK 2 : PHY RANK :1
		RIR_OFFSET#0	LOGICAL RANK 3 : 0 GB	LOGICAL RANK 3 : PHY RANK :5
		RIR_OFFSET#0	LOGICAL RANK 4 : -	LOGICAL RANK 4 : -
		RIR_OFFSET#0	LOGICAL RANK 5 : -	LOGICAL RANK 5 : -
		RIR_OFFSET#0	LOGICAL RANK 6 : -	LOGICAL RANK 6 : -
		RIR_OFFSET#0	LOGICAL RANK 7 : -	LOGICAL RANK 7 : -
	RIR_LIMIT#2	RIR_OFFSET#1	-	-

		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
	RIR_LIMIT#3	RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
	RIR_LIMIT#4	RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
	RIR_LIMIT#5	RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-

SMP	RIR_LIMIT#1	RIR_OFFSET#0	LOGICAL RANK 0 : 0 GB	LOGICAL RANK 0 : PHY RANK :0
		RIR_OFFSET#0	LOGICAL RANK 1 : 0 GB	LOGICAL RANK 1 : PHY RANK :4
		RIR_OFFSET#0	LOGICAL RANK 2 : 0 GB	LOGICAL RANK 2 : PHY RANK :1
		RIR_OFFSET#0	LOGICAL RANK 3 : 0 GB	LOGICAL RANK 3 : PHY RANK :5
		RIR_OFFSET#0	LOGICAL RANK 4 : -	LOGICAL RANK 4 : -
		RIR_OFFSET#0	LOGICAL RANK 5 : -	LOGICAL RANK 5 : -
		RIR_OFFSET#0	LOGICAL RANK 6 : -	LOGICAL RANK 6 : -
		RIR_OFFSET#0	LOGICAL RANK 7 : -	LOGICAL RANK 7 : -
	RIR_LIMIT#2	RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
		RIR_OFFSET#1	-	-
	RIR_LIMIT#3	RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
		RIR_OFFSET#2	-	-
	RIR_LIMIT#4	RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
		RIR_OFFSET#3	-	-
	RIR_LIMIT#5	RIR_OFFSET#4	-	-

		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-
		RIR_OFFSET#4	-	-

Table 13 RIRCHNILVOFFSET register entries for two different memory configurations

5.4 DIMM_TABLE:

We can associate the corresponding DIMM on which the address resides because there exists a sequential mapping between the logical ranks and the DIMM. Although this information is not clearly evident from the information in the manual. But we have found this after due observation.

The pseudo code for of DIMM mapping is given in appendix A.

5.4.1 DIMM readings for ironicity:

For example the physical ranks which map to corresponding DIMMs are shown in the address layout.

DIMM NO	**PHYSICAL_RANKNO**
1	0 and 4
2	1 and 5

Table 15 DIMM to physical-rank identification

5.5 BANK, ROWS and COLUMNS:

The Intel manual for Xeon does not describe any registers from which we can infer how the address mapping is implemented beyond ranks. We get only information regarding the number of banks and the number of row and address bits which are involved in the decoding. But for our purpose, rank mappings are sufficient to provide us the required granularity that we need to establish for software control. This is based on the fact as we shall see later that power saving mechanisms is based on rank-wise policies.

6. DATA STRUCTURE:

Here we discuss the data structures that we have used which capture the essence of the information that we obtain from various registers as discussed in the previous sections.

Data Structure	**Description**
sad_table [MAX_SOCKETS][MAX_SAD_RANGES]	All Cbox of interconnected sockets have the same SAD entries. Each SAD entry specify a new interleaving scheme for that range of physical address.
sad_interleave_table [MAX_SOCKETS][MAX_SAD_RANGES]	For a SAD entry, there is a SAD target based on the interleaving obtained from the physical address bits. The SAD target represents the concerned socket.
tad_table[MAX_SOCKETS][MAX_TAD_RANGES]	The TAD table converts the address range to channel address space. The channel address space removes address space holes due to MMIO/MMCFG regions and interleaved regions. This is done at the Bbox level. The TAD ranges are divided based on the granularity of interleaving and also the contiguity of address space. In short, this entry gives the channel interleaving index for a tad range.
tad_offset [MAX_SOCKETS][MAX_CHANNELS][MAX_TAD_RANGES]	The tad_offset table maps the logical channel number to physical channel number. The logical channel number for a specific range is obtained from the interleaving index from tad entry.
rnk_table [MAX_SOCKETS][MAX_CHANNELS][MAX_RIR_RANGES]	The RANK table converts the local channel addresses to rank addresses space. Like SAD and TAD entry, each RANK table entry is identified based on the granularity of interleaving among ranks. We obtain the logical rank number from the RANK table entry.
rnk_rir [MAX_SOCKETS][MAX_CHANNELS][MAX_RIR_RANGES][MAX_RANKS]	The RNK_RIR table maps the logical rank to physical rank number for that specified RANK range.

dimm_table [MAX_SOCKETS][MAX_CHANNELS][MAX_DIMMS]	The DIMM table corresponds to information about each DIMM. The information includes whether DIMM was populated. The number of bits per column and per row respectively.
memory_technology [MAX_SOCKETS]	The Memory technology register contains information about the Memory configuration mode as well as the paging policy

Table 16 Data Structure used for capturing various table information

6.1 Address mapping scheme Summary:

The decoding logic is diagrammatically explained as follows:

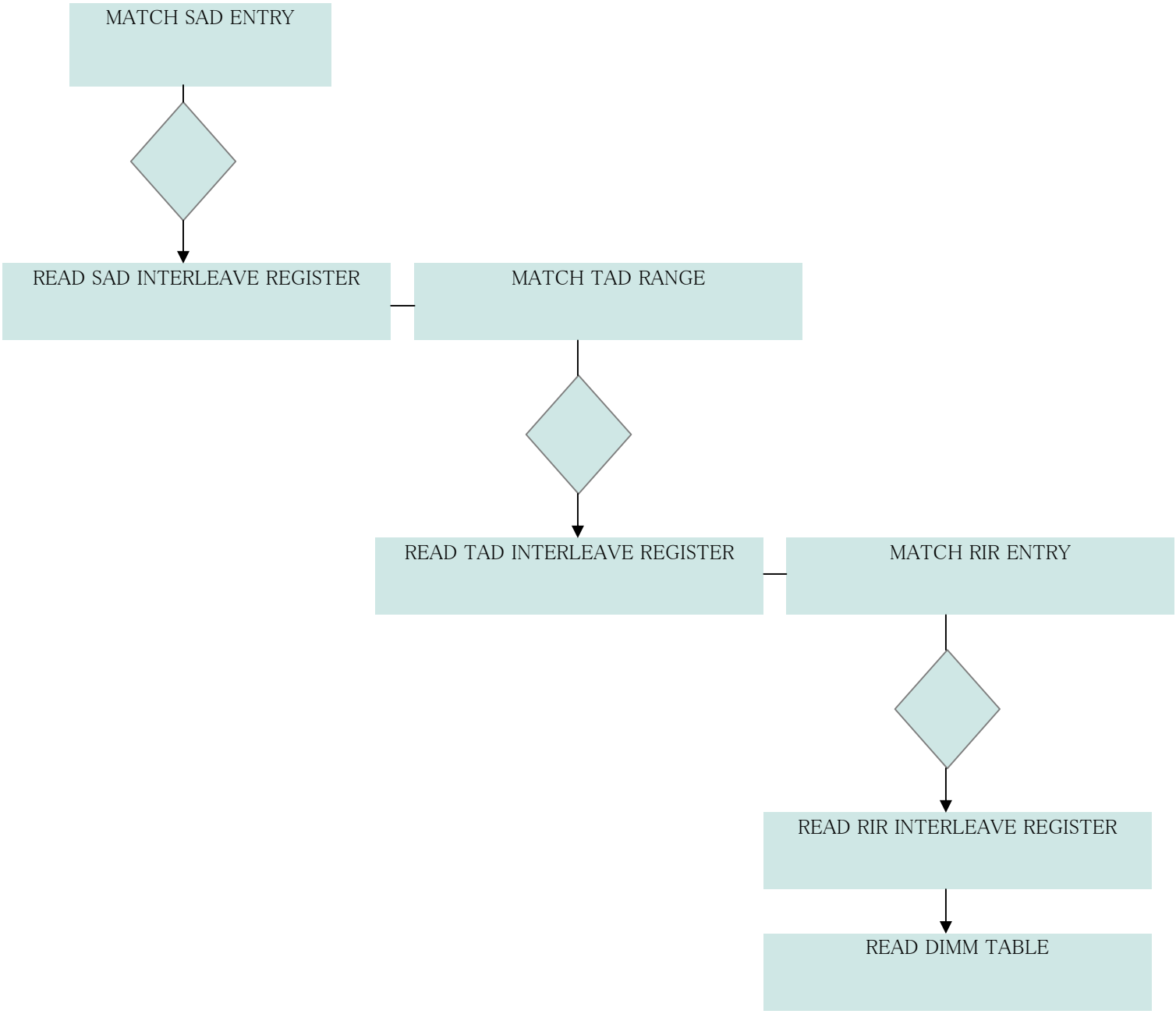


Fig 14 Flow Chart for Address Decoding Logic

7 Implementation:

We have developed a memory allocator based on the hardware information. The allocator has been developed on top of the buddy-allocator in Linux kernel. Pages in the allocator have been grouped under two separate lists belonging to each socket. We have written a synthetic memory traffic generator which we configure to write addresses belonging to a specific socket, channel and rank. We have released both the code under GPL.

8 Evaluations:

We had two options to verify the address mapping scheme.

a) Use error injection registers and applying reverse translations.

b) Use Performance Monitoring utility in SandyBridge.

We found the second option easy and accurate to use.

SandyBridge exposes a set of Performance Monitoring Device Registers. They are used to monitor a wide class of system events within a core. Intel has released a tool suite to observe these events. This tool suite has utilities for observing memory access, power consumption, and various other sets of events. We use the pcm-memory and pcm-power utilities to verify our findings on address mapping.

8.1 Results:

The amount of traffic that we saw in the figure confirms the mapping at the socket and channel level.

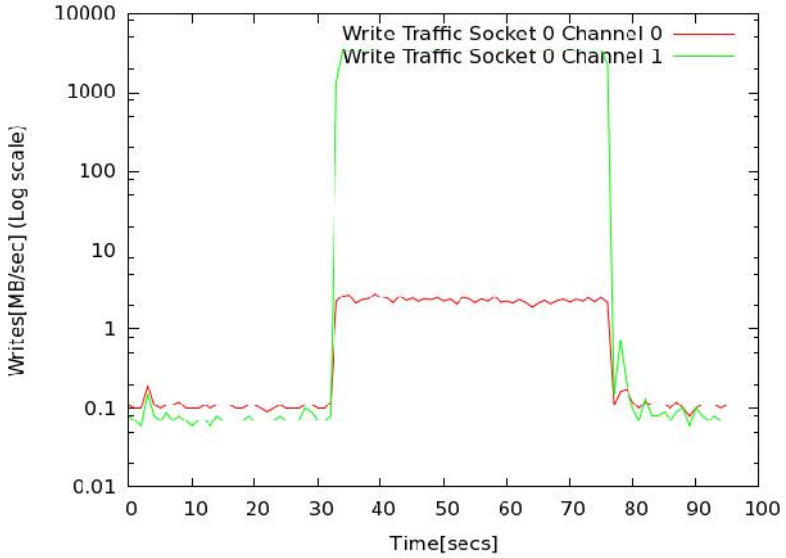


Fig 15 Write Traffic (Log Scale)

The proof for rank-level verification could not be done with the pcm-memory tool because Rank traffic cannot be monitored by pcm-memory. So we used pcm-power to verify the rank mappings. In the following figure, we can infer considerable traffic going to rank 1 of channel 0 and socket 0 because the percentage of CKEOFF represents the amount of idle time in the rank.

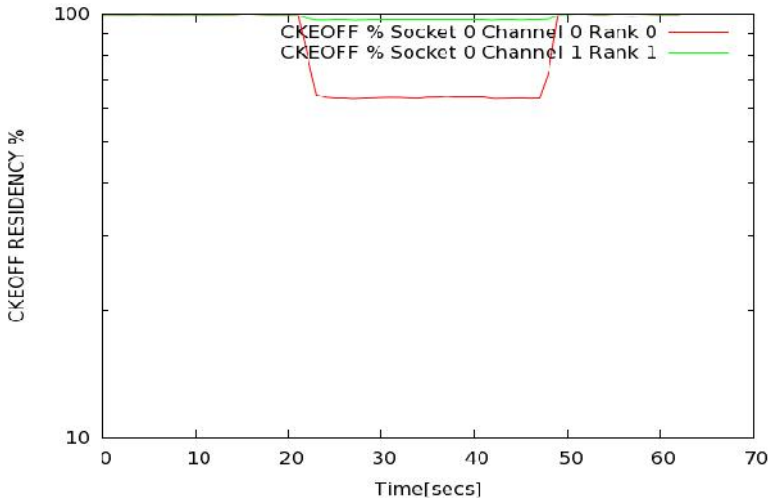


Fig 16 Rank Wise CKEOFF

9 Dynamic Power Management:

To prove the amount of power savings we have to introduce mechanisms by means of which we can dynamically disable power to the granularity of ranks. Although rank-level power management policies exist in the memory controller but auto-power down mechanism on idleness is currently not supported by the hardware. We begin by studying the available power states in SandyBridge.

9.1 Low-Power States in Xeon IMC:

For Xeon E5 DIMMs can be put to two low power states. This information we can get from the PWR_MODE register which is a part of the WBox.

a) CKE Power-down: In this mode, iMC does opportunistic refresh after idle time. Internal DDR clock is disabled in this mode. This mode can be further categorized into three sub low-power states :-

1. Active Power Down a.k.a. APD (55% saving when compared to normal idle state)
2. Pre-Charge Power-down Fast Exit a.k.a PPDF (60%)
3. Pre-Charge Power-down Slow Exit a.k.a PPDS (87%)

b) Self-Refresh: In this mode, iMC relinquishes control over refresh to DRAM controller. This is the lowest possible power state. Self-refresh mechanism is done rank-wise. Ranks enter self-refresh if there are no transactions for a SF_IDLE_CNTR period of time. Self-Refresh Timing Constraint Register controls the self-refresh behavior.

RV (31-29)	RV (23-21)	SF_EN (20)	SF_IDLE_CNTR (19-0)

9.2 Survey of Other Hardware Supported Low Power State:

Mobile DRAMs:

There are DRAM technologies which provide such a feature, namely Partial Array Self Refresh and Deep Power-down. But unfortunately these are only available as mobile memories. Appendix B gives an insight on the amount of power savings that could be gained by using such memory cells.

ACPI Based Power Down:

Xeon 7500 family of processors have a third low power state in which the device is powered off. This is done based on ACPI. This capability can be invoked by either putting the system to a sleeping state where power to memory is suspended; the other mechanism is called hot-plug. The latter requires support from the a) processor as well as b) requires appropriate slots in the chipset. In server technology these special slots come in the form of memory riser cards. Dell R810 has the hot-plug capability. The mechanism by which this is achieved is explained in Appendix D

OverWriting BIOS control fields :

The important point to note is that the power savings option can work on the granularity of ranks.
So a rank-wise power shutdown could be a better option where in which the internal DRAM power generation is disabled, and there are no self-refresh happening; and the unwanted contents of the cell can be lost.

We finally wanted to see if there exists any register control by means of which we can disable the rank in the idle state by gating supply when not necessary and later re-enable on requirement.
The iMC exposes sets of registers which can be used to signal or make hardware configuration changes just like how it is done during boot in BIOS. But unfortunately after boot, most of the much needed registers are locked. Even if some are not locked, overwriting them take no effect because some can be written only once, some require some additional INIT bit to be reset for the effect to take place. Tinkering with such bits cause immediate CPU reset and most of the time a reboot shows error related to PCI hardware. But However through study we have found a technique by means of which the change can be brought to effect without causing a CPU reset. But unfortunately we could not test the same in Intel X5 2650 because the manual provided restricted information as regards to some LOCK DISABLE BITS. The technique is described in Appendix C.

10. Work in Progress:

10.1 Current Work

Hardware Control: Currently, we are trying to observe do there exist any register control fields by means of which we can optimize the power savings even in the self-refresh state. We are also concurrently trying to disable memory devices to whatever granularity supported by the hardware.

Software Control: A major chunk of the work has to be done in introducing the notion of hardware boundaries into the memory management mechanism. There has been previous work in this domain. This work targeted the zoned-buddy allocator in Linux. They had arbitrarily decided on the hardware boundaries over which power decisions could be made as 512 MB. They termed these as memory regions. They introduced a parallel data structure based on this in the buddy-allocator code. They sorted the pages on being freed based on these boundaries. They also use the compaction code in their design to collate pages based on the memory regions.

Currently, our work has been to identify how pages are related to physical memory in the Linux kernel. We have found that there exists three memory models available and the association can change because of this. They are:

a) Flat Memory Model.
b) Discontinuous Memory Model.
c) Sparse Memory Model (NUMA based system).

All these memory models are backed by a data structure called as the memblocks. A memblock is a contiguous region of physical memory with constant start and end physical addresses. For sparsemem, we have found that they match the TAD entries that we obtain from our previous discussion. This is only possible to verify during boot because after boot this data structure is destroyed as it was created only by the bootmem allocator.

I think that changes in the Linux kernel shall had to be made in two ways :

a) Populating the hardware based tables as section in Section X had to be done in the /arch/x86/mm section
b) The page allocation and compaction code changes had to be done in the /mm section

10.2 Power Saving Measurements:

We present our power saving measurements based on kernbench. It is a CPU throughput benchmark. Socket interleaving was disabled for the experiment. Thus, 32 GB each of the available 64 GB physical memory was allotted to each of the sockets. The experiment was run with two kernel configurations. In the First configuration, memory allocation was enabled from both the sockets. We call it the Free Allocation Setup. In the Second Configuration, only socket 0 was allotted 32 GB whereas memory allocation from the second socket socket 1 was disabled. We call it the Restricted Allocation Setup. The benchmark results have been tabulated below :

SetUp	Kernel Version : linux-3.10-rc6(memblock_modified)				
Free	Load Type	Elapsed Time	User Time	System Time	% CPU
	Half Load	668.520000	5201.93	1136.28	948
	Optimal Load	576.6	5230.49	1190.38	1037.5
	DRAM Joules	12403.1005			
	Kernel Version : linux-3.10-rc6(normal)				
Restricted	Load Type	Elapsed Time	User Time	System Time	% CPU
	Half Load	670.2	5272.660000	1126.170000	954.000000
	Optimal Load	578	5269.2	1174.74	1038
	DRAM Joules	16019.157			

Table 17 kernbench bencmark results

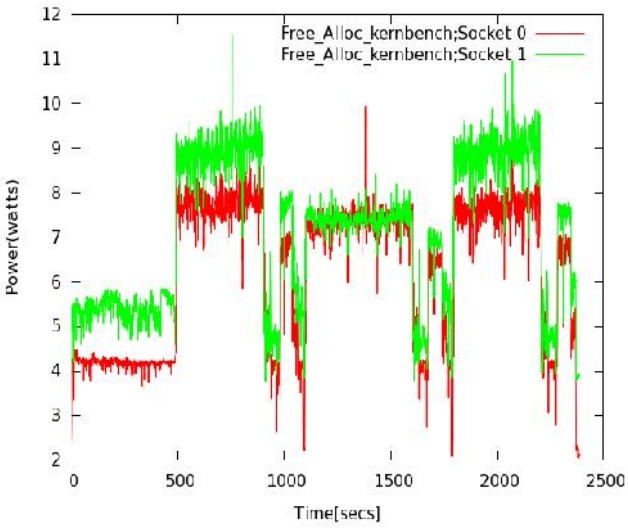


Fig 14 Power vs Time
(Free memory allocation)

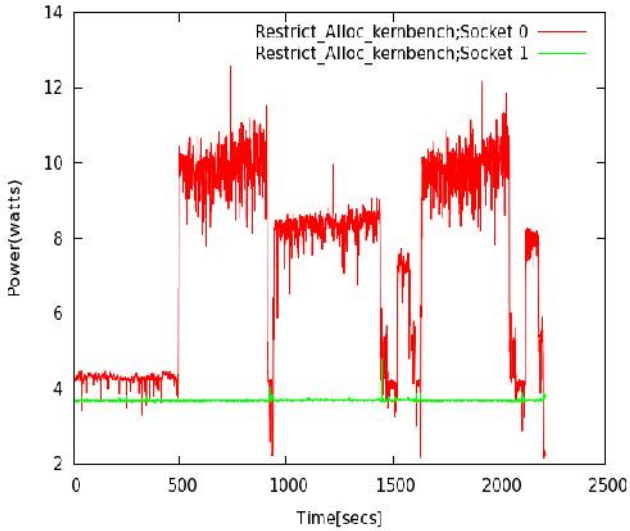


Fig 15 Power vs Time
(Restricted memory Allocation)

	Energy Savings	Lowest Power State
Power Aware Page Allocation	22 %	Self-Refresh
Power Aware Page Allocation + Auto Power down capability	51% (estimated)	ACPI S4 State Power down

11. Conclusion:

This project is still under progress. But efforts till now have shown that we can make considerable power savings if we can darken the unused memory based on software and hardware control features. We have established the following till date:

a) address mapping scheme in that Intel processors;
b) that there exits rank-wise power saving options
c) rank-wise interleaving range for them to be used as the basic block for power intelligent memory management.

Bibliography:

Description	Document Name
1. Intel Xeon E5 2600	http://www.intel.la/content/dam/doc/datasheet/core-i7-lga-2011-datasheet-vol-2.pdf
2 Intel Xeon 7500	http://www.intel.com/Assets/en_US/PDF/datasheet/323341.pdf
3 Intel C600	http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/c600-series-chipset-datasheet.pdf
4 Intel Performance Monitor	http://www.intel.com/content/dam/www/public/us/en/documents/design-guides/xeon-e5-2600-uncore-guide.pdf
5 Hot-Plug	https://www.kernel.org/doc/Documentation/memory-hotplug.txt
6 Hot-Plug Drivers	http://events.linuxfoundation.org/sites/events/files/lcjp13_chen.pdf
7 ACPI spec	http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf
8 Hot-Plug Hardware	http://www.intel.com/Assets/PDF/whitepaper/323479.pdf
9 Hot-Plug Hardware	http://www.intel.com/content/dam/doc/product-brief/e8500-muli-processing-chipset -brief.pdf
10 Hot-Plug Hardware	http://www.dell.com/downloads/global/products/pedge/en/poweredge-server-11gen-whitepaper-en.pdf
11 Book	Memory Systems by Bruce Jacob
12 ACPI driver	msdn.microsoft.com

Appendix A

Psuedo Code:

```
int isSocket(long long unsigned int phy_addr) {

      for (k = 0 ; k < NO_SOCKETS ; k++) {

       for (i = 0 ; i < MAX_SAD_RANGES ; i++) {

          if (!RULE_ENABLE(sad_table[k][i]))
              break;

          prev_limit = curr_limit;
          curr_limit =  SAD_LIMIT(sad_table[k][i]) << 26;

            if( (prev_limit < phy_addr ) && ( curr_limit > phy_addr )) {

              if(SOCKET_INTERLEAVING_MODE(sad_table[k][i]))
                sockilv_indx = MID_HASH_INTERLEAVE(phy_addr);
              else
                sockilv_indx = LOW_INTERLEAVE(phy_addr);

              switch(sockilv_indx) {
                case 0 :  socket_no   =  SAD_TARGET_PKG1(sad_interleave_table[k][i]);
                          break;
                case 1 :  socket_no   =  SAD_TARGET_PKG2(sad_interleave_table[k][i]);
                          break;
                case 2 :  socket_no   =  SAD_TARGET_PKG3(sad_interleave_table[k][i]);
                          break;
                case 3 :  socket_no   =  SAD_TARGET_PKG4(sad_interleave_table[k][i]);
                          break;
                case 4 :  socket_no   =  SAD_TARGET_PKG5(sad_interleave_table[k][i]);
                          break;
                case 5 :  socket_no   =  SAD_TARGET_PKG6(sad_interleave_table[k][i]);
                          break;
                case 6 :  socket_no   =  SAD_TARGET_PKG7(sad_interleave_table[k][i]);
                          break;
                case 7 :  socket_no   =  SAD_TARGET_PKG8(sad_interleave_table[k][i]);
                          break;
                default :
                          break;
              }
              goto ok;
          }
       }
      }
      ok:
      return socket_no; }
```

Psuedo Code :

```
int isChannel(long long unsigned int phy_addr, int socket_no,long long unsigned int *channel_addr ) {

          for (i = 0 ; i < MAX_TAD_RANGES ; i++) {
           prev_limit = curr_limit;
           curr_limit =  TAD_LIMIT(tad_table[socket_no][i]);

           if (prev_limit == curr_limit)
              break;
           if( (prev_limit < phy_addr ) && ( curr_limit > phy_addr )) {

           sck_ways  = SOCK_WAYS(tad_table[socket_no][i]);
           chn_ways  = CHN_WAYS(tad_table[socket_no][i]);

              /*correction added */
           phy_addr -= TAD_OFFSET(tad_offset[socket_no][0][i]);
           temp_bits=phy_addr & 0x7f;

              /* decode channels */
             if(chn_ways==2)
             chn_addr = phy_addr >> 6;

           else
             chn_addr = phy_addr >> (6 + sck_ways);

           channel_no = chn_addr % (1 << chn_ways) ;

           switch(channel_no) {
              case 0 :
                    channel_no = CHN_ID0(tad_table[socket_no][i]);
                    break;
              case 1 :
                    channel_no = CHN_ID1(tad_table[socket_no][i]);
                    break;
              case 2 :
                    channel_no = CHN_ID2(tad_table[socket_no][i]);
                    break;
              case 3 :
                    channel_no = CHN_ID3(tad_table[socket_no][i]);
                    break;
              default :
                    break;
            }
          }
         }
        chn_addr = (chn_addr << 6) | temp_bits
       *channel_addr = chn_addr;
        return channel_no;
}
```

Psuedo Code :

```
int isRank(long long unsigned int chn_addr, int socket_no, int channel_no, long long unsigned int *rank_local_addr) {

        if(CLOSED_PAGE(memory_technology[socket_no]))
          rank_addr = chn_addr >> 6;
        else
          rank_addr = chn_addr >> 13;

        for (i = 0; i < MAX_RIR_RANGES ; i++) {
            if(!RIR_VAL(rnk_table[socket_no][channel_no][i]))
               break;
            prev_rank_limit = curr_rank_limit;
            curr_rank_limit = (RIR_LIMIT(rnk_table[socket_no][channel_no][i]) << 29);

            if((rank_addr >= prev_rank_limit) && (rank_addr < curr_rank_limit)) {
              rnk_ways    =  RIR_WAYS(rnk_table[socket_no][channel_no][i]);
              rank_no     =  rank_addr % (1 << rnk_ways);
              phy_rank_no =  RANK_TGT(rnk_rir[socket_no][channel_no][i][rank_no]);
              break;
             }
         }
         *rank_local_addr = rank_addr;

         return phy_rank_no;
}

int isDimm(long long unsigned int phy_addr, int socket_no, int channel_no, int phy_rank_no) {


        for( i = 0 ; i < MAX_DIMMS ;i++) {
            if (DIMM_POP(dimm_table[socket_no][channel_no][i])) {
               num_ranks = NUM_RANKS(dimm_table[socket_no][channel_no][i]);
               if((phy_rank_no < (i+1)*num_ranks) && (phy_rank_no > i*num_ranks)) {
                  dimm_no = i;
                  break;
               }
         }
        }
        return dimm_no;
}
```

Appendix B (Mobile RAM Power Options)

The following power saving features is used in case of Mobile memories. They are as follows:

First is the a) partial array self-refresh b) Deep Power Down.

a) Partial Array self-refresh (PASR): This is an intelligent self-refresh mechanism in which all the banks are not put to self-refresh. It selectively refreshes certain banks while leaving others inactive

b) Deep Power down (DPD):

In Deep Power-down, the internal power supply is switched off and all the refresh operations are suspended. As a result, data will not be retained after entering the DPD mode. Under normal co-operations, a single active bank can consume less than 80mA. When the refresh is active, the power consumption is about three times as much as during normal operation, however in DPD mode, the current use is reduced to about 10uA.

But it happens that SDRAM's supporting such technologies is expensive. Only Micron Technology ship SDRAMs based on PASR and DPD. However they are mainly used in mobile RAM's. I have not found mention of any DIMM modules shipped with such SDRAMs.

Such Special DRAM chips are manufactured by Elpida Inc now also known as Micron Technology.

Appendix C (Overwriting Locked registers in Intel)

In this we have tried manual disabling of rank to observe power savings. Since Intel's current hardware does not offer support for such run-time changes in hardware configuration at such a low granularity. So it was a challenge for us to find out if it was possible. Although we could not meet with success in this case because of the fact that we could not move beyond a point because of lack of information regarding some register level details. Although we had requested for privileged permission from Intel Inc. but our request was denied. However we have learnt some important
concepts of programming at such a level in the process.

Since we need to change the hardware specific configuration parameters, we found that most of the concerned bits were locked. The lock types as applied to CSRs can be seen from the Register Type Section in the Intel Processor Manual. The lock type that we faced were RW-LB meaning that
Such a register field is RW locked but in the event of a bypass bit being set it can be overwritten.

In our case we targeted a register called as the DIMM_MEMORY_TECHNOLOGY register.
This contained rank disable bits which has the capability of disabling refresh-operations, scrubbing
and sparring operations on the memory region. But however the bits were RW-LB. Overwriting them would cause immediate CPU reset.

LOCK_DISABLING in Intel Processors:

What we needed for this is the information about the concerned LOCK_DISABLE bit which
had to be set prior trying the overwrite. The Xeon manual's description has mentioned only one such LOCK_DISABLE bit for the DIMMTEMP_STAT configuration register which is also RW locked.
It is locked by CLTT_DEBUG_DISABLE_LOCK. When the CLTT_DEBUG_DISABLE lock is cleared, software can write to this byte. The corresponding lock bit is available in the CHN_TEMP_CFG register. However for our purpose we should also know the concerned register which has the lock disable bit.

Even if we are able to overcome the hurdle, there is a thing to add to it. There are certain CSRs writing which does not necessarily guarantee that the corresponding effect shall take place.
There is a MC_INIT_STATE_G register which has a bit field called MCR_DONE field. For changes to take effect, we have to overwrite the locked registers prior the MCR_DONE bit is done and most importantly the MCR_DONE bit is placed under BIOS control. This we have verified on Intel Xeon 5500 platform

Finally, to further add to the overwriting of locked bits, since writing some can cause CPU resets which is undesired in our case. So there is a register called the MC_CFG_CTL register.

Prior overwriting if we set the corresponding bit in MC_CFG_CTL register, it does not cause any CPU reset. Again for the corresponding Xeon Manual, we did not find information on such a register. These lock control registers are collectively termed as non-core registers. We find mention of these register in Intel Xeon 5500 series processor.

APPENDIX D (ACPI options)

A brief Introduction to ACPI:

Prior to ACPI, we had a legacy based power management option. This is called the APM Based Power Management. In this scheme, the Platform Controller Hub (PCH) has a timer which can be set by enabling the Intel SMI (System Management Interrupt) Control and Enable Register. The Intel SMI handler when invoked checks the _STS Register for any system activity. If none of the system bits are set, the Intel SMI handler can increment a software counter. When the counter reaches a sufficient number of consecutive minutes with no activity, the Intel SMI handler puts the system into a low power state. (Section 5.14.10 [3])

ACPI:

The PCH is the central component in power control. This is tied to Intel architecture.

ACPI exercises more granularities over power control using the PCH. The ACPI standard identifies a system with power states and makes necessary transitions between them to optimize the usage of power. The system wide sleeping states are indicated as S0-S5.

A system wide S-State is identified based on the power states of its components. For example, the compute intensive core is identified with C states. There are a total of C0-C6 states as per the specification. The higher the number the more the power saving and longer the latency. What happens when a C state transition is made is that a low power thread is signaled and invoked.
Any C State transition always goes through C0 state.

Xeon Supports C0, C1 and C3 States. The core state transitions can be observed using the Xeon Performance Monitoring Registers and are known as state transition cycles. Each C-state is associated with a Sleeping State or the S –State.

Peripherals are associated with D0-D3 States. For ACPI compliant devices, the controllers come
with register set with power down functionality supporting D0 and D3 States. In the Intel C602 Chipset, SATA controllers, Audio Devices and LAN have this capability. This register set is not associated with the iMC.

Terminologies:

SMI: Scalable Memory Interconnect

SCI: System Control Interrupt

PME: PCI Message Event

SMI Space: System Management Mode
GPE Registers: ACPI register set; GPE registers are used to cause SCI interrupts.

Mechanics of ACPI Based Low Power State Control:

To explain how these register sets are used for the SATA controller is as follows: Whenever the device driver sets a D3 State to the Controller, it also simultaneously invokes an ACPI method. This resets the device and cuts down its power. (Section 5.17.8[3]). During the wakeup, the port wake pin is asserted; this causes a PCI Message Event (PME). When the PME message is received, PCH sets the PCI_EXP_STS bit. This causes an SCI using the GPE_STS register if the corresponding PCI_EXP_EN bit is set which is responsible for invoking the required state transition.

Mechanics of Hot-Plug Operation:

Besides low power device states, SATA Controller also support Hot-Plug Operation. This means that it has got a SATA port with mechanical presence switch with Hot Plug Enabled feature set. Whenever the port status changes, a corresponding bit in the GPE Register is set. Setting this bit invokes the corresponding handler. Infact the chipset manual supports the fact that software can take advantage of the power savings in the low power states while enabling hot-plug operation. (Section 5.17.5[3]). Below we describe the code flow in ACPI based Hot Plug Operation:

Event Flow for Insertion [12]

When a device is inserted into the system, the following takes place:

- **User:** The user inserts the hardware in the slot.
- **Hot-Plug PCI controller (HPPC):** Asserts a GPE.
- **Core chip set:** Raises an SCI.
- **ACPI driver:** Clears the GPE event and runs the _Lxx method associated with the GPE.
- **_Lxx method:** Reads status bits from the HPPC to determine that the event was an insertion event and which slot the event took place on.
- **_Lxx method:** Executes Notify (,0) on the PCI bus that the slot is on.
- **ACPI driver:** Executes _STA methods for the devices specified in Notify () in the previous step.
- **_STA:** Returns 0x0a; that is, the device is present but not enabled.
- **ACPI driver:** Tells the PCI driver to enumerate the bus.
- **PCI driver:** Reads configuration space to identify the device.
- **PCI driver/PNP subsystem:** Loads and starts the drivers for all functions of the device.
- **Device drivers:** Request the functions be turned on.
- **ACPI driver:** Executes _PS0 methods, if present.
- **PCI driver:** Writes to configuration space to turn on the device according to PCI Power Management specifications.
- **Device driver:** Begins using the device normally.

Event Flow for Removal[12]

When the user requests to eject a device, the following steps take place. These steps assume that the hardware has an eject button associated with each device. If it does not, then the user must use the Add/Remove Hardware Wizard to eject a device. In this case, the process would start at step 8 in this list:

- **User:** The user presses the eject button for the slot.
- **HPPC:** Asserts a GPE.
- **Core chip set:** Raises an SCI.
- **ACPI driver:** Clears the GPE event and runs the _Lxx method associated with the GPE.
- **_Lxx method:** Reads status bits from the HPPC to determine that the event was an eject request and which slot the event took place on.
- **_Lxx method**: Executes Notify(,3) function 0 of the slot that is requesting eject.
- **ACPI driver:** Requests the Plug and Play system to eject the device.
- **PNP:** Queries all drivers for all functions of the device, and if successful, tells the drivers to unload.
- **PCI driver**: Writes to configuration space to turn off the device per PCI Power Management specifications.
- **ACPI driver**: Executes _PS3 methods, if present.
- **ACPI driver:** Executes _EJ0.
- **_EJ0:** Engages motors, solenoids, lights, and so on to eject the device. When the device has been ejected and status bits updated accordingly, it returns.
- **ACPI driver:** Executes _STA to verify the device ejected successfully.
- **_STA:** Returns 0x00 (device not present). Had the device not ejected successfully, _STA would have returned 0x05 (that is, present but not functioning).
- **ACPI driver**: Complete cleanup. Had _STA returned 0x05, the ACPI driver would have told the operating system that there was a problem and an error dialog would pop up on the user's screen

The main challenge with this approach is to tweak the code to support pseudo injections and removals.

Power Disabling DIMM Granularity:

Since for obtaining lowest possible power state, the main hurdle is to minimize the leakage current.
We wanted to use the Hot-Plug Capability. But unfortunately, the PCI Slots to which the DIMMs
are connected do not have the Hot-Plug capability. This information can be obtained by reading the PCI Express Ports Capabilities registers (SLTCON, SLTSTS and SLTCAP). The device having the capability is identified by their Slot Number indicated in the register. Our investigation concluded with the finding that such a capability is supported in Intel Xeon 7500 series processor []. There are three systems with this capability which we found was commercially available Dell R810, M610. [NB: PCI based Hot-Plug is different than ACPI based Hot-Plug]

Power Disabling Socket-Level Granularity:

The PCH supports the following Sleeping States:

Sleep Type	Comment
S1	System Lowers the processor's power consumption. No Snooping is possible in this state.
S3	Asserts SLP_S3# .The SLP_S3# signals control to the non-critical circuits. Power is only retained to devices needed to wake from this sleeping state, as well as to the memory
S4	Asserts SLP_S3# and SLP_S4#. The SlP_S4# signal shuts off the power to the memory subsystem
S5	Same power state as S4. Asserts SLP__S3#, SLP_S4# and SLP_S5#

This is another ACPI based power disabling option in which we have a coarser granularity. We target to bring the system to S4 ACPI State in which PCH cuts off power supply to the memory-subsystem based on a SLP_S4 signal. This SLP_S4 signal can be invoked by in many ways. However for our purpose, the following ways seem viable. (Section 5.14.6.3)

Action	**Transition**
Set GPIO27 in GPE0_EN register	Wake up from S4/S5 State to S0
Configuring a GPIO signal as a control method for a Sleep Button	Trigger Transition to S4 State from S0

This possibility is under exploration now

Without ACPI:

HP based servers have developed a solution in which they disable power to failed DIMMs by Making changes to the SMI handler. We can also explore in this direction.

Appendix E (Sample output displaying memory addressing CSR)

ADDRESS_MAPPING TABLES FOR INTEL_SANDYBRIDGE **********

Socket No :0

SAD RANGE #0 33472[MB] INTERLEAVE 0

sad_table[0][0] 82c3 0

SAD RANGE #1 66240[MB] INTERLEAVE 9090909

sad_table[0][1] 102c3 9090909

****

TAD LIMIT 0 3327[MB], SOCK WAYS :0 CHN_WAYS 1 TAD_OFFSET : 0

TAD LIMIT 1 33535[MB], SOCK WAYS :0 CHN_WAYS 1 TAD_OFFSET : 768

****

CHANNEL NO :0

RIR LIMIT 0 :15872[MB] RIR_WAYS :2

PHY RIR NO :0 : RIR_OFFSET[MB] 0

PHY RIR NO :4 : RIR_OFFSET[MB] 0

PHY RIR NO :1 : RIR_OFFSET[MB] 0

PHY RIR NO :5 : RIR_OFFSET[MB] 0

CHANNEL NO :1

RIR LIMIT 0 :15872[MB] RIR_WAYS :2

PHY RIR NO :0 : RIR_OFFSET[MB] 0

PHY RIR NO :4 : RIR_OFFSET[MB] 0

PHY RIR NO :1 : RIR_OFFSET[MB] 0

PHY RIR NO :5 : RIR_OFFSET[MB] 0

****

CHANNEL_NO :0 DIMM No :0 Present 1

CHANNEL_NO :0 DIMM No :1 Present 1

CHANNEL_NO :1 DIMM No :0 Present 1

CHANNEL_NO :1 DIMM No :1 Present 1

Memory Technology: OPEN

memory_technology[0] :104

Socket No :1

SAD RANGE #0 33472[MB] INTERLEAVE 0

sad_table[1][0] 82c3 0

SAD RANGE #1 66240[MB] INTERLEAVE 9090909

sad_table[1][1] 102c3 9090909

****

TAD LIMIT 0 66303[MB], SOCK WAYS :0 CHN_WAYS 1 TAD_OFFSET : 33536

****

CHANNEL NO :0

RIR LIMIT 0 :15872[MB] RIR_WAYS :2

PHY RIR NO :0 : RIR_OFFSET[MB] 0

PHY RIR NO :4 : RIR_OFFSET[MB] 0

PHY RIR NO :1 : RIR_OFFSET[MB] 0

PHY RIR NO :5 : RIR_OFFSET[MB] 0

CHANNEL NO :1

RIR LIMIT 0 :15872[MB] RIR_WAYS :2

PHY RIR NO :0 : RIR_OFFSET[MB] 0

PHY RIR NO :4 : RIR_OFFSET[MB] 0

PHY RIR NO :1 : RIR_OFFSET[MB] 0

PHY RIR NO :5 : RIR_OFFSET[MB] 0

****

CHANNEL_NO :0 DIMM No :0 Present 1

CHANNEL_NO :0 DIMM No :1 Present 1

CHANNEL_NO :1 DIMM No :0 Present 1

CHANNEL_NO :1 DIMM No :1 Present 1

Memory Technology: OPEN

memory_technology[1] :104

SBridge_driver registration OK