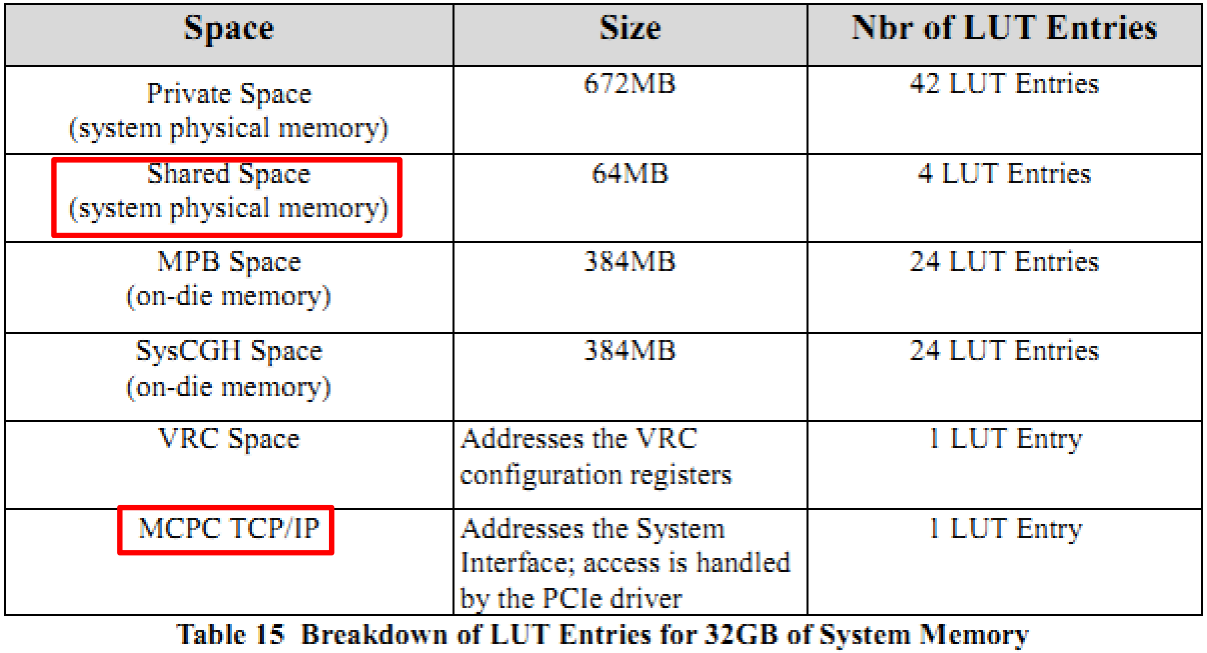
SCC : rckpc

2011.11.07

이 용 준

RCKPC COMMUNICATION

Breakdown of SCC LUT Entries

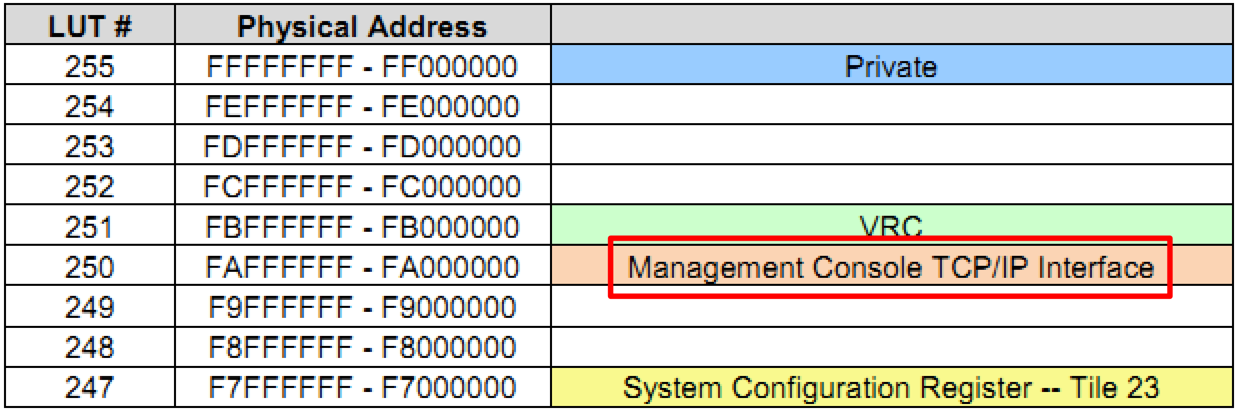


Space	Size	Nbr of LUT Entries
Private Space (system physical memory)	672MB	42 LUT Entries
Shared Space (system physical memory)	64MB	4 LUT Entries
MPB Space (on-die memory)	384MB	24 LUT Entries
SysCGH Space (on-die memory)	384MB	24 LUT Entries
VRC Space	Addresses the VRC configuration registers	1 LUT Entry
MCPC TCP/IP	Addresses the System Interface; access is handled by the PCIe driver	1 LUT Entry

Table 15 Breakdown of LUT Entries for 32GB of System Memory

System Interface

- **System Interface**
  - The System Interface (MCPC TCP/IP) : 16MB
    - Addressable by the core



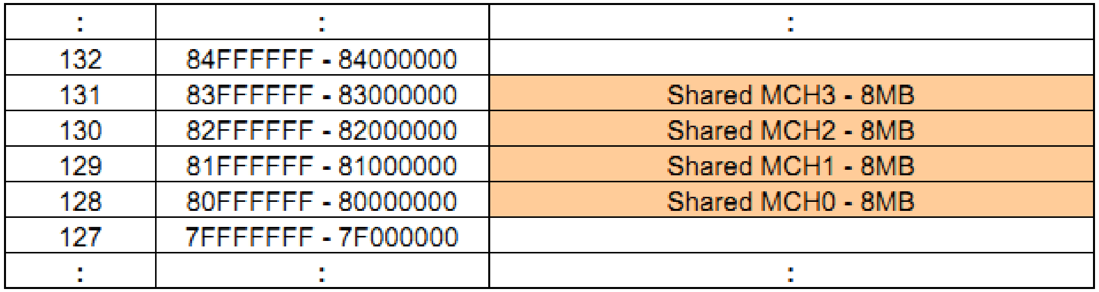
LUT #	Physical Address	
255	FFFFFFFF - FF000000	Private
254	FEFFFFFF - FE000000	
253	FDFFFFFF - FD000000	
252	FCFFFFFF - FC000000	
251	FBFFFFFF - FB000000	VRC
250	FAFFFFFF - FA000000	Management Console TCP/IP Interface
249	F9FFFFFF - F9000000	
248	F8FFFFFF - F8000000	
247	F7FFFFFF - F7000000	System Configuration Register -- Tile 23

crb & Rckpc

- **Communication between MCPC(the host) & SCC cores**
  - MCPC : crb
    - mcpc_driver/crbif_net.c
    - A network interface for transferring data with SCC cores
  - SCC cores : Rckpc
    - Linuxkernel/linux-2.6.16-mcemu/drivers/net/rckpc.c
    - A layer for transferring data to/from MCPC

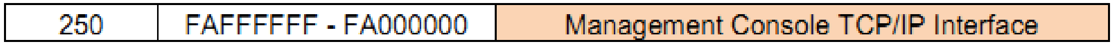
Rckpc

- **From Rckpc's point of view**
  - `MAX_PACKET_SIZE` : 4096
    - Maximum size of data transfers
  - `rxPacketSlots` : 16
    - 1 slot : `MAX_PACKET_SIZE` = 4096
  - `rxBaseAddress` : 0x80200000



:	:	:
132	84FFFFFF - 84000000	
131	83FFFFFF - 83000000	Shared MCH3 - 8MB
130	82FFFFFF - 82000000	Shared MCH2 - 8MB
129	81FFFFFF - 81000000	Shared MCH1 - 8MB
128	80FFFFFF - 80000000	Shared MCH0 - 8MB
127	7FFFFFFF - 7F000000	
:	:	:

    - Start address of the network buffer in the shared memory space
    - 4 consecutive memory tiles are reserved for the 4 quadrants
    - Each core uses `rxPacketSlots*MAX_PACKET_SIZE` bytes
  - `txMailbox` : 0xFA000000



250	FAFFFFFF - FA000000	Management Console TCP/IP Interface

    - Address of the Tx mailbox

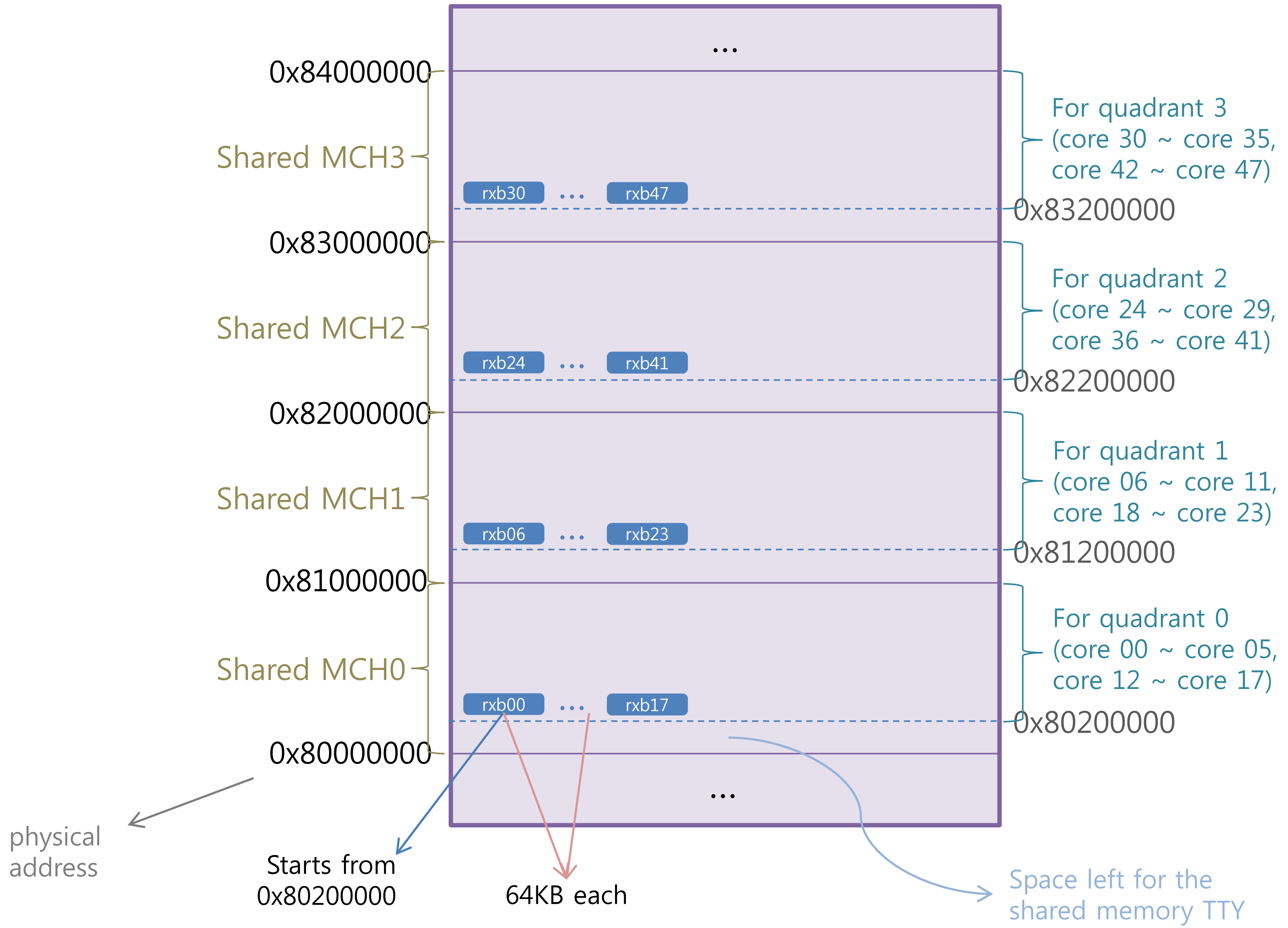
Rckpc

- **struct rckpc_priv @ rckpc.c**
  - struct rckpc_priv

```
struct rckpc_priv {
                        …
    void* rxb;           // Receive Buffer
    void* mailbox;       // Host mailbox
};
```

  - rxb
    - Receive buffer
    - Mapped space for rxPacketSlots
      - rxPacketSlots*MAX_PACKET_SIZE = 16*4K = 64K per core
    - Marked as MPB so it can be easily invalidated
    - Write-through flag is also set
      - So every write is immediately propagated to main memory
  - mailbox
    - Mapped for txMailbox
    - Offsets of the host mailbox registers
      - MBX_NULL : 0x00
      - MBX_CONFIG : 0x20
      - MBX_PACKETSTART : 0x40
      - MBX_PACKETDATA : 0x60
      - MBX_RXDONE : 0x80

Utilization of Shared Memory



SCC Core Rckpc Send

- **SCC core Rckpc send (`rckpc_tx()`)**
  - Copy the packet data to the mailbox, which is the system IF.
    - If the data is smaller than `RCK_CLINE_SIZE`(32B),
      - Copy the first cacheline to `MBX_PACKETSTART`.
      - Copy all remaining data to `MBX_PACKETDATA`.
        - If the remaining data is smaller than `RCK_CLINE_SIZE`,
          - Copy `RCK_CLINE_SIZE` bytes to `MBX_PACKETDATA`.
          - And repeat this process.
        - Else,
          - Just copy the remaining bytes to `MBX_PACKETDATA`.
    - Else,
      - Copy them all to `MBX_PACKETSTART`.
  - The source core ID is at the packet header.

SCC Core Rckpc Receive

- **SCC core Rckpc receive (`rckpc_rx()`)**
  - 1. Interrupt triggers the receiver's polling function.
  - 2. The polling function looks for valid packets.
    - The address of the receiving packet is calculated.
      - The current slot at the receiving core's `rxb`
    - Checks for valid packet lengths there.
  - 3. Copy data from the address.
  - 4. Invalidate the packet length so as not to process the data again.
  - 5. Tell the host we are done with the slot and move on to the next.
    - Write the current slot value to `MBX_RXDONE` in the mailbox.
    - Update the current slot value.

crb

- **From crb's point of view**
  - `MAX_PKTSIZE` : 4096
    - Maximum size of data transfers to Rckpc
  - `txPacketSlots` : 16
    - 1 slot : `MAX_PKTSIZE` = 4096
    - Same as `rxPacketSlots` in Rckpc code.
  - `txBaseAddress` : 0xF0200000
    - Local address where the shared memory is located
      - At the destination DDR3 memory controller
      - 16 MB at 0xF0000000 of each MC are mapped to 0x80000000 – 0x83000000.
    - Start address of the packet space at the MC
    - Corresponds to `rxBaseAddress` in Rcpc code.

MCPC crb Send

- **MCPC crb send (`crbnet_tx()`)**
  - 1. Extract the destination address from the IP header.
    - Get the core ID.
  - 2. Calculate the address of the buffer(`rxb`) for the destination.
  - 3. Write the packet data to the address.
    - 32 bytes at a time
  - 4. Generate an interrupt for the destination core.

MCPC crb Receive

- **MCPC crb receive**
  - The crbif daemon fetches data from the FPGA.
  - Then the filter function passes the packets that are intended for the network interface to crb.
    - crbif_main.c : `crbif_filter()`
      - 1. Decode the packet header.
        - Gets the source core ID.
      - 2. Calls `crbnet_pktHandler().`

MCPC crb Receive

- **MCPC crb receive**
  - crbif_net.c : `crbnet_pktHandler()`
    - Core ID of the source already known at this stage
    - Decode the different mailbox addresses.
      - If the address is at `MBX_PACKETSTART`,
        - Call `crbnet_tx_start()`.
          - Copy the data from the mailbox.
      - If the address is at `MBX_PACKETDATA`,
        - Call `crbnet_rx_data()`.
          - Copy the data.
      - If the address is at `MBX_RXDONE`,
        - Call `crbnet_rx_complete()`.
          - This is a verification that a send from here to a core's Rckpc has been received fine.

ISSUES WITH RCKPC

Migration & Rckpc

- **Rckpc send & crb receive**
  - After migration,
    - At send,
      - The core IP info is in the packet.
        - MODIFY : Modify IP to that of the migration src core.
      - Rckpc shall simply write this packet to the system IF.
    - At receive,
      - crb shall decode the packet header to find out the src IP.
        - Should be fine as it is.
  - To solve this problem,
    - Every core has to maintain a IP-to-Core table.

Migration & Rckpc

- **crb send & Rckpc receive**
    - After migration,
        - At send,
            - crb shall write at the original `rxb` of the migrated core.
            - Then, crb shall send an interrupt to the wrong core.
                - MODIFY : Send the interrupt to the migration dst core.
        - At receive,
            - When the core gets interrupted,
            - Rckpc shall probably read the original `rxb`.
                - Should be fine as it is.
    - To solve this problem,
        - MCPC has to maintain a IP-to-Core table.
        - At every migration, MCPC has to be told about it somehow.