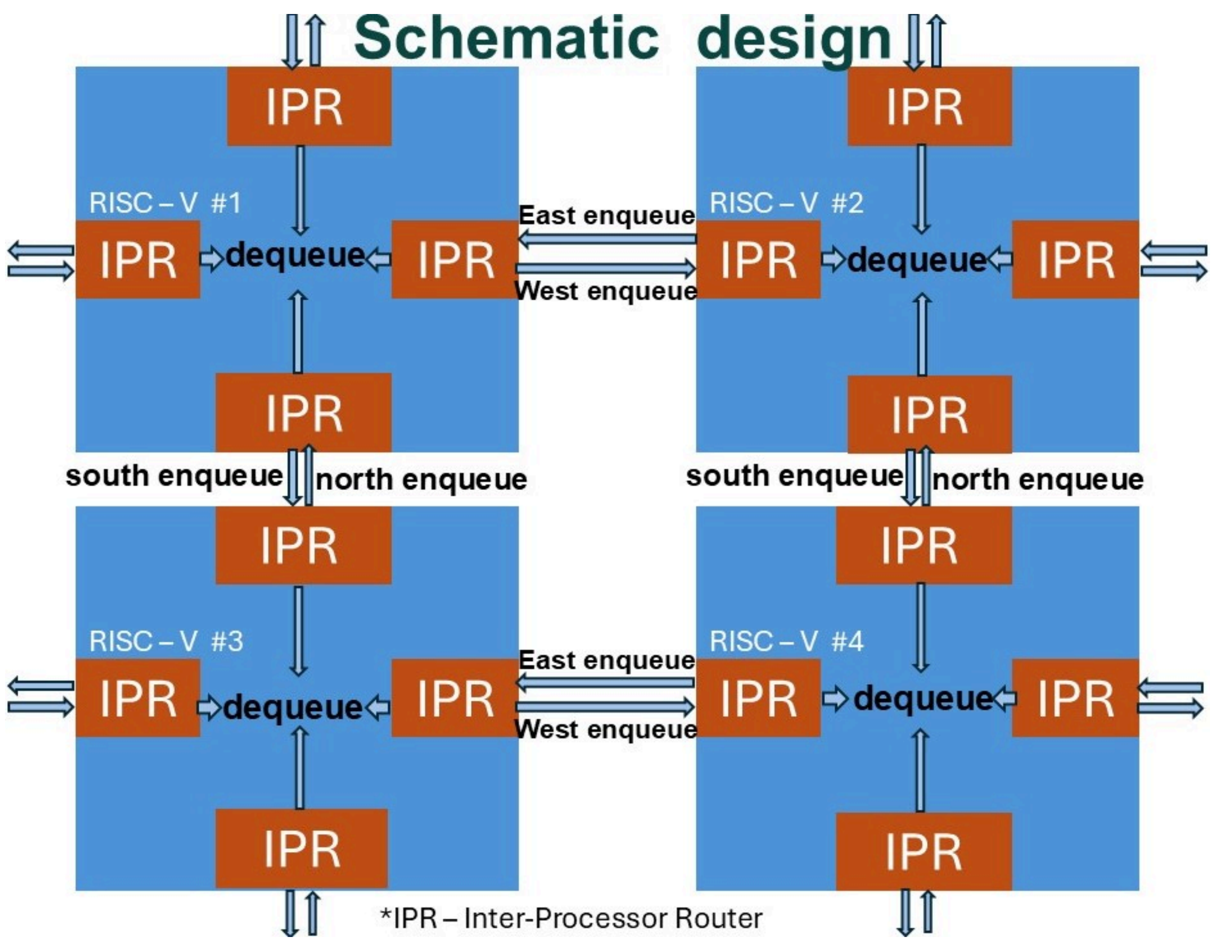
Micro Architecture Specification



High Level Architecture:

Every processor will have four IPR's (Inter Processor's Router, which is basically FIFO with some adaptations) that will have the ability to get data from the East, West, North and South directions to maintain modularity and scalability.

The processor dequeues data from its own IPRs, which reside in the core-level region, and enqueue data by passing through the cluster level, routing it back to its neighbors' IPRs.

Keep in mind that when a processor requests a read, it reads the data from its own IPR located in its core region, allowing it to access the data immediately. In contrast, when a processor requests a write, it writes the data to another processor's IPR located in that neighbor's core region. Since writing isn't an urgent task compared to reading, it's ok if the writing will take a few more clock cycles due to passing through the cluster region.

The total memory address range is 0x00000000 to 0x1C091FFF.

When the LSU unit accesses an address in the range 0x19000000 to 0x1A000000, it activates the IPR peripherals.

The two least significant bits (LSBs) are used to select which IPR to read from or write to (a total of four IPRs, one for each direction).

For example: if the core sends read req to the address 0x19000001, the request will be mapped to the IPR which sits in the west of the core since the lsb is 01.(north =00, west=01, south=10, east=11). and the core reads from the ipr's inside his core region.

If the core sends a write request, then the request will be mapped to the core which seats on the west side relative to the requested core, and the data will be written to the ipr which seats on the east side of the core that accepts the request.

IPR:

Every IPR will consist of a FIFO

the FIFO has a watchdog logic that flash the fifo in case that no read request has been performed for a certain number of clk's

FIFO:

Every FIFO will have the following interface:

inputs:

1. write clock
2. read clock
3. write reset
4. read reset
5. data in *// write data [31:0]*
6. write enable *//enqueue*
7. read enable *//dequeue*

outputs:

1. data out *// read data [31:0]*
2. FIFO full
3. FIFO empty

The FIFO will consist of the following components

- dual port SRAM memory

- write logic
- read logic
- 2 synchronizers
- watchdog

Dual Port Memory:

inputs:

1. write clock
2. write address
3. enqueue
4. write data
5. read address

outputs:

- read data

The memory will be implemented as SRAM, the size of each FIFO memory will be determined later.

Write Controller:

input:

1. write reset
2. write clock
3. write enable *//enqueue*
4. read address in gray after synchronization

output:

1. write address in binary *// for the SRAM*
2. write address in gray *// for the w2r synchronizer*
3. FIFO full

The write controller has a few roles:

he will possess a counter for the write address which will increment every time write enable has inserted the controller. the address will be delivered to the SRAM

he will convert the binary address into gray code and deliver the converted address into the clock synchronizer from write to read to reduce errors he will get the synchronized gray read address and will compare to the write gray address to determine if the FIFO is full or not

Read Controller:

input:

1. read reset
2. read clock
3. read enable *//dequeue*
4. write address in gray after synchronization

output:

1. read address in binary *// for the SRAM*
2. read address in gray *// for the r2w synchronizer*
3. FIFO empty

The write controller has a few roles:

1. he will possess a counter for the read address which will increment every time read enable has inserted the controller. the address will be delivered to the SRAM

2. he will convert the binary address into gray code and deliver the converted address into the clock synchronizer from read to write to reduce errors

3. he will get the synchronized gray write address and will compare to the read gray address to determine if the FIFO is empty or not

Read 2 Write Synchronizer:

input:

1. write clock
2. gray read address

output:

- gray read address synchronized to write clock

consist of 2 FF, the Din to the first FF will be the gray read address

the clock for both FF will be the write clock

Write 2 Read Synchronizer:

input:

1. read clock

2. gray write address

output:

- gray write address synchronized to read clock

consist of 2 FF, the Din to the first FF will be the gray write address

the clock for both FF will be the read clock

Watchdog:

input:

- read enable
- read clock

output:

- reset

The watchdog will count how many clock cycles there has not been an entry to the FIFO for reading. After X clock cycles, the watchdog will reset the read and write addresses. Every time there is a reading from the FIFO, the watchdog will reset back to 0 and he will start counting to X again.

Message passing packet definition:

we will define a packet of information where the metadata of the packet will appear at the header and the data will follow.

The bits count and the information encoded in the metadata are subjected to change along the development of the grid.

The metadata will include:

- Destination core - 2 bits *//for debugging purposes*
- Source core - 2 bits *//for debugging purposes*
- data type - x bits *// integer, floating point, weights, activations, accumulators*
- data format - 3 bits *// 2,4,8,16,32 bits*
- packet size - 8 bits *// how many words are in the packet*
- reserved - 8 bits *// for future use*

We also want to add a parity check for each word for correctness and debugging, but we debate whether to add the parity bit at the beginning of each word or at the header. if we add it at the beginning of each word we will consume 1 bit out of the 32 bit of a word, so we may add an extra bit to the

SRAM, but we are not sure if it is possible. to add it at the header may cause the header to increase in size for a long packet.