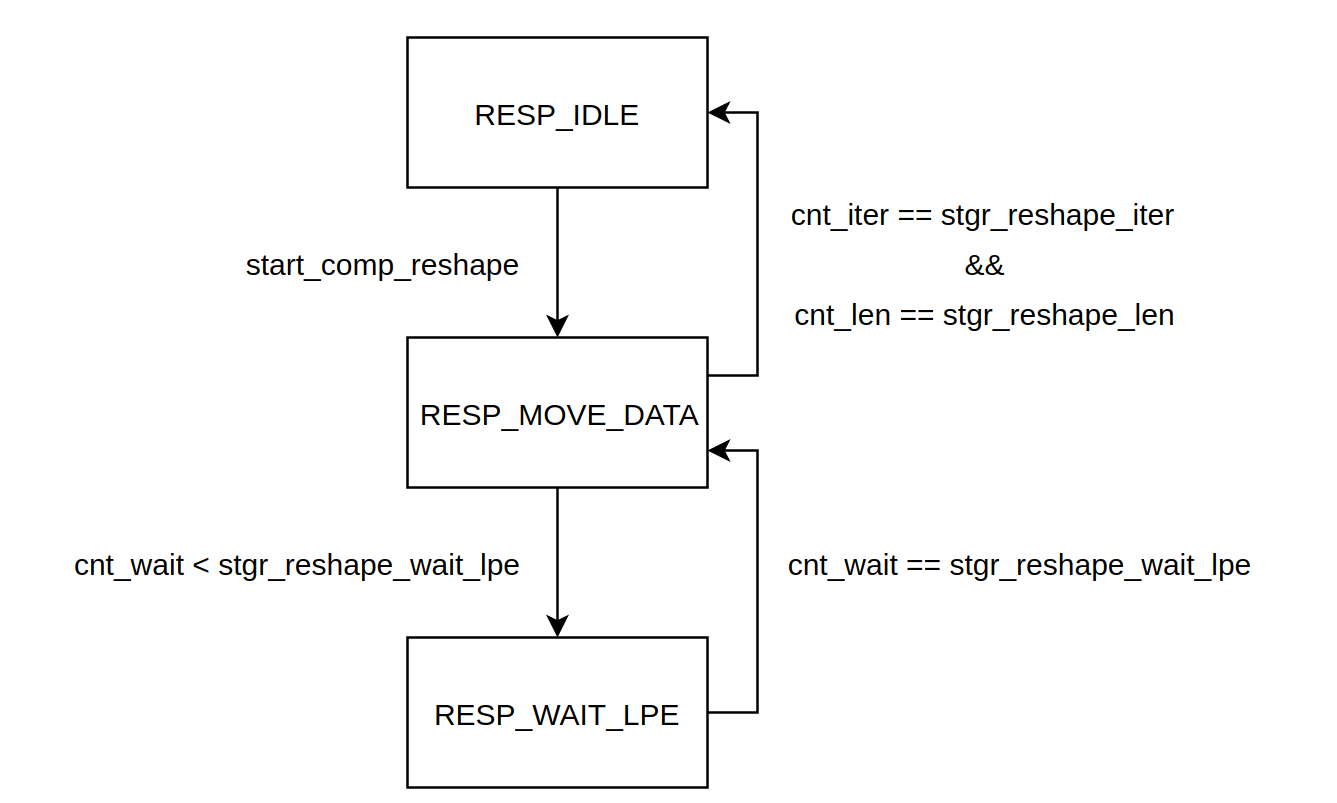
# Controller - Reshape

The Reshape Controller is a finite state machine (reshape\_s) which consists of three states: RESP\_IDLE, RESP\_MOVE\_DATA, and RESP\_WAIT\_LPE. The state conversion relationship of reshape\_s is shown in the following figure.



When the input reset signal asserted, the reshape\_s is forced to initialized to RESP\_IDLE state, and all the counters are also reset to zero. The reshape\_s doesn’t leave the RESP\_IDLE state until it receives the start\_comp\_reshape, which is driven by the register interface.

When in the RESP\_MOVE\_DATA state, with the ctrl\_reshape\_lb\_ren signal asserted, the reshape controller moves the data one by one from Local Buffers to their corresponding CPEs.

There are two counters, cnt\_iter and cnt\_len, making suring the number of data transferred is correct. The following pseudo code shows how they works.

int address = stgr\_lb\_addr;

for (cnt\_iter = 0; cnt\_iter < stgr\_reshape\_iter; cnt\_iter ++) {

for (cnt\_len = 0; cnt\_len < stgr\_reshape\_len; cnt\_len ++) {

address += 1; // Move a data from LB to CPE

}

}

In each cycle, only one data can be pulled from the LPE FIFO to the Global Buffer, while 16 data from 16 KCEs in the same row are pushed into the LPE FIFO. Therefore, the reshape controller needs to wait the LPE FIFO is nearly empty, so another counter cnt\_wait is introduced. It keeps counting when the Reshape Controller is not in the RESP\_IDLE state.

# Controller - GB2LB

In the TanJi-3 accelerator, the 2-level scratchpad, including Global Buffers and Local Buffers, is explicitly controlled by the software code. That is to say, users must move the required data from the GBs to the LBs manually with the MOV\_GB2LB instruction.

The GB2LB Controller is used to perform data transfer from GBs to LBs. Note that the transfer between these two buffers is unidirectional, and the data in the LBs is just a copy of part of the data in their corresponding GBs.

The internal mechanism of the GB2LB controller is similar with the Reshape Controller. Because it does not need to wait the LPE FIFOs drain when processing the MOV\_GB2LB instruction, there is only two states, GB2LB\_IDLE and GB2LB\_RUN, in this controller.

int gb\_address = stgr\_gb\_addr;

int lb\_address = stgr\_lb\_addr;

for (cnt\_iter = 0; cnt\_iter <= stgr\_gb2lb\_iter; cnt\_iter ++) {

for (cnt\_len = 0; cnt\_len <= length; cnt\_len ++) {

gb\_address += 1;

lb\_address += 1;

}

gb\_address += skip + 1;

lb\_address += skip + 1;

}

# Controller - DDR2GB & Master Interface

Actually, there is no dedicated module to control the data transfer between the off-chip DDR memory and the Global Buffers. In the MOV\_DDR2GB mode, the control signals from the register interface are muxed to the hzzmif (master interface) directly.

In this section, we will focus on the internal logic of the master interface. Here is the port definitions of this module.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Width** | **Description** |
| start\_hzzm | input | 1 | The start signal of MOV\_DDR2GB. |
| complete\_hzzm | output | 1 | When MOV\_DDR2GB is done, this signal is asserted to inform the register interface that the operation is done. |
| ctrl\_dir | input | 1 | 0: miso, 1: mosi |
| ctrl\_ddr\_addr | input | 54 | The DDR start address of the burst transfer. |
| ctrl\_mo\_ren | output | 1 | The read enable signal of the Global Buffers. |
| ctrl\_mo\_data | input | T2D\_WIDTH | The data read from Global Buffers. |
| ctrl\_mo\_valid | input | 1 | This signal indicates that the ctrl\_mo\_data is valid. |
| ctrl\_mi\_valid | output | T2D\_WIDTH | The data written to the Global Buffers. |
| ctrl\_mi\_valid | output | 1 | This signal indicates that the ctrl\_mi\_data is valid. |

The state machine in hzzmif includes four states, HZZM\_IDLE, HZZM\_MOSI, HZZM\_MISO, and HZZM\_MOSI\_RESP.

HZZM\_IDLE is the initialized state, and in this state all the output ports are set to invalid. When start\_hzzm is asserted, the state machine goes to HZZM\_MOSI or HZZM\_MISO according to the input ctrl\_dir signal. At the same time, the master interface will release a write/read command to the HZZ slave.

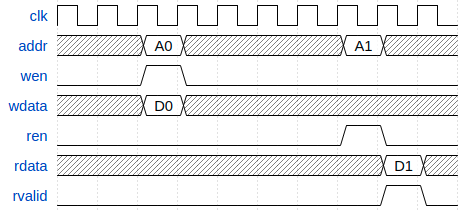
In the HZZM\_MISO state, the accelerator will monitor the FIFO empty signal. If the FIFO is not empty, the master interface fetches the data from it and store them into the Global Buffers.

In the HZZM\_MOSI state, the master interface read data from the Global Buffers and issue them through the HZZ bus. However, the slave side may not be able to process all the data in time. If the master side initiates the next data transfer too early, the data may be lost. Therefore, we introduce a state named HZZM\_MOSI\_RESP. After the master sends all the data in a burst, it goes into this state, and keep stuck there until it receives the response signal from the slave. Here the response signal from the slave means that all the data has been stored into the DDR memory.

# Register Interface

TanJi-3 features an AXI-4 interface to configure its parameter registers.

The AXI interface packs the access request into HZZ protocol to save the port number of the chip. The HZZ slave interface unpacks the HZZ bus data, and access the DLA registers according to the following protocol (Regif Protocol).



All the registers are connected to the Regif Master (i.e. HZZ slave interface) through a MUX. This part of the code can be found in dla\_regif\_map, and is easily understood. If you want to add a new register, it is important to add it into the address map. The address map of all the registers can be found in the project repo (tj3/doc/register-definition.md).