## Wbload\_controller:

#### General overview

The “wbload\_ctrl” module is used to controll the data transfer between DDR and WB set which is near the CPE according to the control signals from convolution controller (dla\_ctrl\_conv) and FC controller (dla\_ctrl\_fc). The data tranfer mode is set to burst mode including 64 and 256. And every singal data read from DDR to WB set will be allocated to 16 independent WBs (with depth of 16) which are corresponded to 16 KPEs with the same address (for example, from address 0 to 15). And if the burst mode is set to 64, the data send to WB set will be filled in 4 clock cycles from lowest bit to highest bit. Finally do to burst length is limited to 256, so if the burst number is lager than 256, it will be divided in to several parts to finish the burst.

#### FSM



The FSM of CONV operation shows you how the state machine works, you can see that once the FSM is started, then the state will change to DDR\_A and calculate the busrt length according to the busrt mode and operation (convolution, fc, pooling). In DDR\_A, it will start the DDR2WB transfer change to next state according to the rest burst length. The state will stay in DDR\_B untill all of the busrt is finished and change to READY to wait for another wbload command.

#### Mechanism

for (cnt1=0; cnt1<#active\_cpe; cnt1++){

for (cnt2=0; cnt2<acitve\_wb\_depth; cnt2++){

for (cnt3=0; cnt3<data\_num; cnt3++){

read data from ddr to corresponded WB

}

}

}

**#active\_cpe** : represent how many CPEs are active now

**active\_wb\_depth**: present current slice length in CONV, and is constant 1 in FC

**data\_num** : use in different burst mode. In 256 mode, this is constant 1; in 64 mode, this is constant 4;

#### signal definition:

|  |  |
| --- | --- |
| stgr\_wbload\_ddr\_addr | The start wbload ddr address |
| stgr\_wbload\_wb\_cnt\_default | The default WB depth (slice length in CONV, 1 in FC) |
| stgr\_wbload\_kpe\_act\_num | cpe\_cnt \* kpe\_cnt |
| stgr\_wbload\_kpe\_cnt | 1 in burst mode 256, 4 in burst mode 64, represent the clock cycles used to fill with 256 data width |
| stgr\_wbload\_cpe\_cnt | The number of active CPEs |
| actual\_wb\_cnt | The following WB depth according to the rest slice length |
| burst\_num | The total number of burst |

#### Work method

The wbload module has different work method in different operation.

**CONV:**

First, calculate the total number of burst and the DDR burst start address. And then start the data transfer in DDR\_A and DDR\_B. every new transfer must wait for previous transfer finish. When the total transfer finished, it will not start the next burst untill a new wbload request come (Involved in 4 state: IDLE, DDR\_A, DDR\_B, HZZM\_READY).

**FC:**

The same operation like CONV, but the wb\_cnt is always set to 1. And have a new state—FC\_WAIT.

**POOLING:**

Do nothing

## Wbload\_mng

Merge the wbload ctrl signal from dla\_ctrl\_conv and dla\_ctrl\_fc module. It is easy, so there is no redundant introduction

## Convolution Controller

#### General overview

This module controls the behavior in the convolution operation including ifmap read from LB, weight read from Wb and the output sequence. This module will first divide the whole kernel into several slices due to the limited WB depth. In one slice, it will generate the whole ofmap partial sum using the whole ifmap (of course, limited by LB depth). When finished, it will start the rest convolution using the weights in another slice. This module will generate the ofmap one-by-one in the sequence of left to right and top to bottom. Every change of the ofmap element, we must consider whether the LPE has finished it work or not. If not, must wait for it.

We call the different parts of the kernel “KS kernel slice” and when the operation in one slice, we call it “KW kernel window”

#### Mechanism

for (cnt1=0; cnt1<kslice\_cnt; cnt1++){

for (cnt2=0; cnt2<ofmap\_height; cnt2++){

for (cnt3=0; cnt3<ofmap\_width; cnt3++){

for (cnt4=0; cnt4<kslice\_length; cnt4++){

read ifmap and weight

}

}

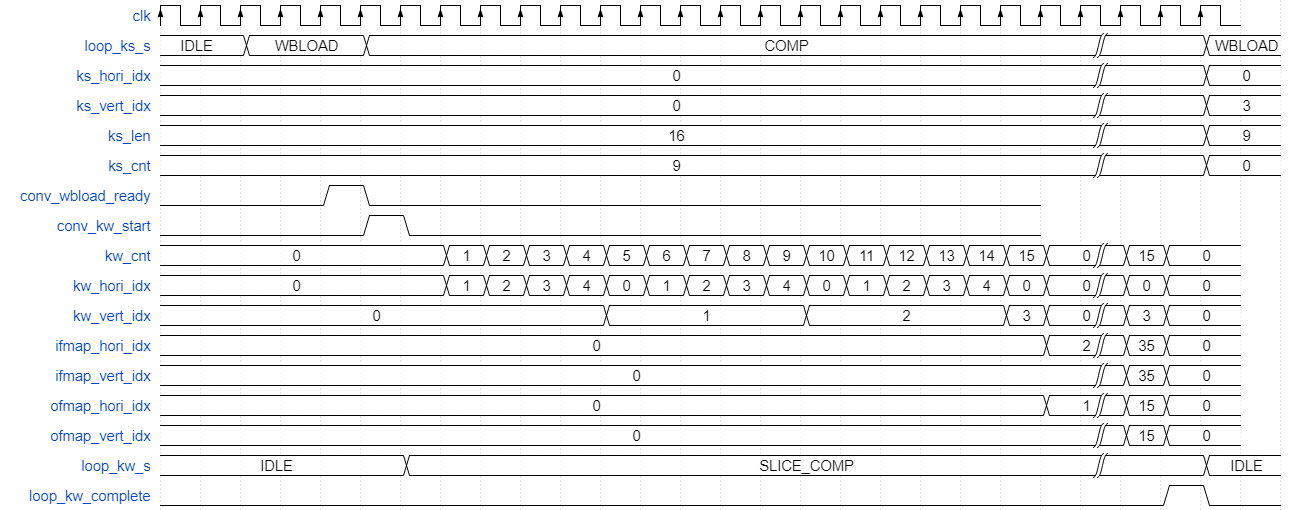
}

}

#### Signal definition

|  |  |
| --- | --- |
| Stgr\_conv\_k\_hori\_delta | The distance of the new kslice weight start address in hori between original address, start with 1 (1 index) |
| Stgr\_conv\_k\_vert\_delta |  |
| Stgr\_conv\_wait\_lpe | The total clock cycles used in the LPE you need to wait for |
| Stgr\_conv\_slice\_max | the default max slice length, generally 16 |
| Ks\_len | The current kernel slice length |
| Ks\_size\_cnt | The rest kernel length |
| Ks\_hori/vert\_idx | Current kernel slice start index |
| Kw\_hori/vert\_idx | The index used in one slice to present which weight is read in current slice |
| lb\_hori/vert\_nopad | The address of LB calculated according to current index in the KW and current ofmap index and of course stride (these signals are considered the padding element) |
| Bif\_lb\_conv\_hori/vert | The ifmap stored in LB without padding (means effective address in LB) |

Here give a waveform of the first slice of a 5x5 kernel and the ifmap size is 36x36 with stride 2.



## FC controller

#### General overview

This module controls the behavior of FC operation, including ifmap in LB read and weight load from DDR to WB. According to the FC mechanism in KPEs, we just use the first address of WB. Meanwhile, in order to fully use the sparsity of ifmap, once the ifmap is read from LB, will check whether it is zero or not to reduce the power consumption of reading 256 weights. And for better reducing power consumption, the weight load won’t start untill the first non-zero ifmap comes.

#### Mechanism

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

if (ifmap is zero) {

continue;

}

else {

request for wbload;

while (!wbload\_ready);

ifmap read;

weight read;

}

}

#### FSM

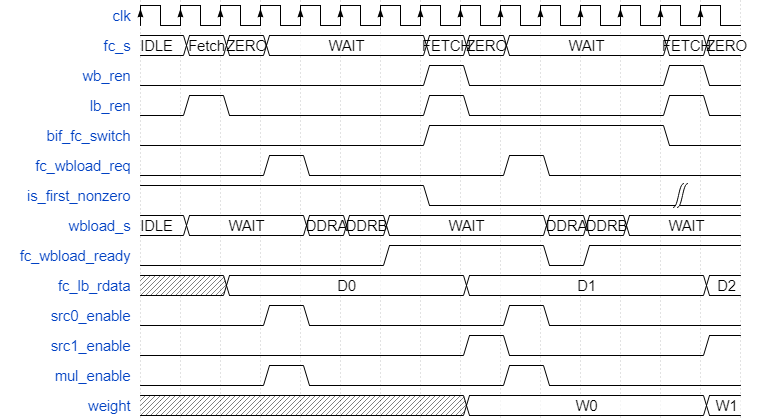


This is the FSM flow of FC operation. First, when start the FSM, it will fetch one ifmap from LB and check whether it is zero or not. If is, then continue to fetch another ifmap; if not, ask for loading weight from DDR. Once the weight is loaded, it will fetch the next data.

#### Signal definition

|  |  |
| --- | --- |
| Fc\_wbload\_req | Used to ask for the weight load |
| Fc\_wbload\_ready | Tell that the weight load from DDR is done, and the weight load controller is ready for another weight load |
| Is\_first\_nonzero | Use to indicate that the first non-zero ifmap, 1 represent ifmap is still zero. And will set to low once the first nonzero ifmap come during the whole FC operation |
|  |  |
|  |  |

Here give a short piece of FC waveform to show you the behavoir of FC operation



In this waveform, assume that the first ifmap read from LB is nonzero. And the “fc\_s, wb\_ren, lb\_ren, bif\_fc\_switch, fc\_wbload\_req, is\_first\_nonzero” are the signals of FC controller. And the “wbload\_s, fc\_wbload\_ready” are the signals of Wbload controller. The rest are the signals of scheduler of computing part.