0812019 電機

HW4

Part I. Architecture and Algorithm Choice

Network topology	4x4 Torus mesh
Routing Algorithm	X-Y
Switching	Wormhole switching
Buffer Size	3
Virtual Channel	No/1 Channel per input port
Flow Control	ACK-REQ protocol

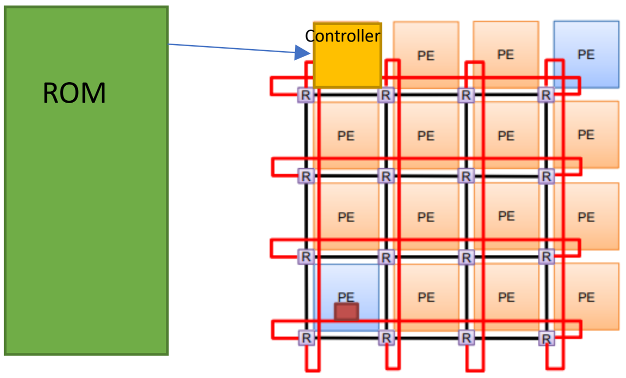


Fig.1 4x4 Torus mesh

To simplify the implementation, I choose X-Y routing algorithm. This algorithm guarantee there will no dead lock or living lock during the system running. Regarding switching method, I use wormhole switching, which only allow the flits from the same packet to fill in a channel. Below shows the design of the flit. The first flit in the packet contains the BoP (the 33th bit), source id[31:28], destination id[27:24], Is_parameter[23], Is_bias[22], and OP_id[21:18]. The body flits begin with 2 zero and follow with data value. Tail flit consists of EoP[32] and the last data value.

Header	BoP [33]	EoP [32]	Src [31:28]	Dst [27:24]	Is_parameter [23]	Is_bias [22]	OP id [21:18]	0 [17:0]
Body	BoP [33]	EoP [32]	Data [31:0]					
Tail	BoP [33]	EoP [32]	Data [31:0]					

Fig2. Flit format

A. Controller

Controller can access ROM to get the parameters and data, and send them to the assigned place. At first, Controller will send weights and bias for each conv and linear layer. To weights, the header of “Is_parameter” signal will be high to indicate the packet is for parameter, while “Is_bias” signal will be low. To bias, instead, “Is_bias” will be high.

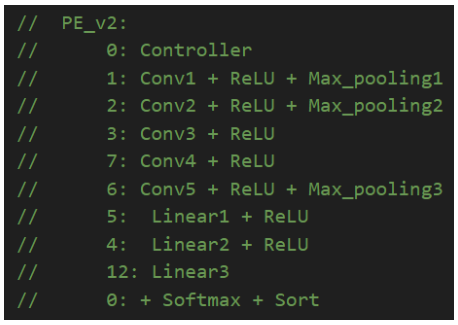
All the weights and bias will be transmitted to the core, to be more precious, they are stored in the PEs.

After finishing sending out all the parameters, Controller will send the image to the first PE (PE_1) to compute Conv1+MaxPooling1. Then, the following data propagating will be ruled by each PE because they will know their next destination for their output feature map.

Finally, Controller will receive the output feature map from linear3 (fc8). Controller will perform the softmax function, calculate the probability of each class, and print the results on screen.

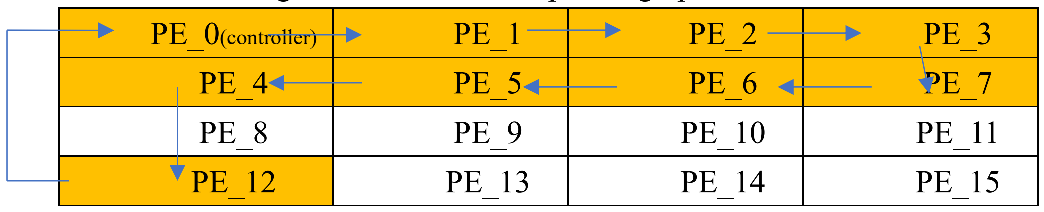
B. PE design

PEs are included in the cores. While initializing the cores object, PE will also be initialized and got to know what layer and functions shall be executed in this PE. Also, the attributes of the layer, such as stride and kernel size, will be set then. The below lists the PE id and the corresponding operations. This figure also shows how the Vgg-16 be partitioned and how the data propagate during inference. To be noticed, PE_0 is replaced by controller.



```
//  PE_v2:
//      0: Controller
//      1: Conv1 + ReLU + Max_pooling1
//      2: Conv2 + ReLU + Max_pooling2
//      3: Conv3 + ReLU
//      7: Conv4 + ReLU
//      6: Conv5 + ReLU + Max_pooling3
//      5:  Linear1 + ReLU
//      4:  Linear2 + ReLU
//      12: Linear3
//      0: + Softmax + Sort
```

Fig3. PE and the corresponding operations



PE_0(controller)	PE_1	PE_2	PE_3
PE_4	PE_5	PE_6	PE_7
PE_8	PE_9	PE_10	PE_11
PE_12	PE_13	PE_14	PE_15

Fig4. PE id and their place
(Orange block means some ops will be executed there.)

While initializing PEs, PE will also know what next Op will be done in the next destination PE. Then, when sending the output feature map, PE will config the OP_id in the header of the packet to indicate what op will be executed in the next PE. The below shows the Op_id and the corresponding operations.

Op_id (in decimal) (use [21:18] in header)	Op type
1	Conv+ReLU+MaxPooling
2	Conv+ReLU
3	Linear+ReLU
4	Linear

C. Core design:

It contains a PE and NI in each core. First, the core will always check if there is packet needed to send to somewhere. After fetching the packet, I split the source id, destination id, data and op_id. Then, I reformat them into flit format, and store in a "std::queue<sc_lv <34> >". With ACK-REQ (ack_tx, req_tx) handshaking, the flits will send to the router.

Conversely, when "req_rx" is on, which means the router want to send the flit to this core, the core will let "ack_rx", and prepare to format the packet.
After collecting all flit received from the router, the core will execute "check_packet()" method to send the packet to PE.

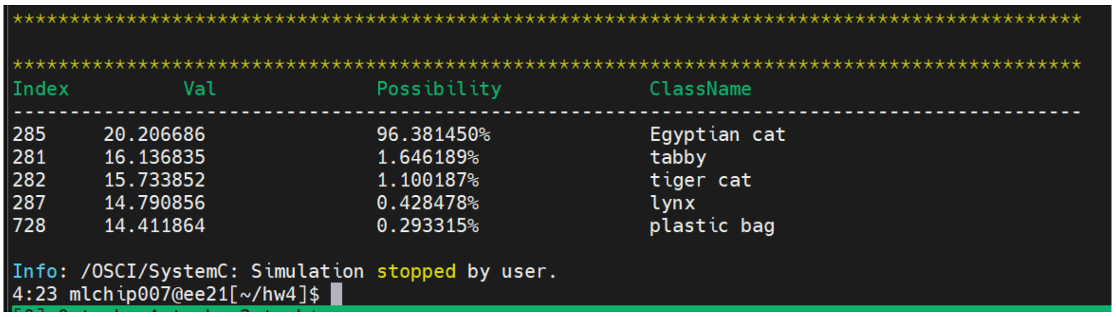
D. Router design:
It contains 5 queue (fifo) for 5 input ports. In the router, I separate it into 2 parts: input port control and output port control.

In the former part, a routing unit is implemented to determine the destination of the input flit. After getting the destination, it will lock the destination output port and limit the user (here means the packet). Then, it will start transmission and get all the receiving flit into the input buffer (queue). Here I set the queue size as 3. It means when queue store 3 flit, it will stop the transmission temporally until the queue has empty slots.

In the output port control, it will check which output port is locked and try to fetch the flit from its corresponding input buffer. When the tail flit is sent, the output port control will free the access to the locked output port.

About the routing unit, which implemented X-Y routing algorithm, it will calculate the next step in each router using its own router id. Therefore, when router get a source id from a header flit, it can calculate the next step according current router id.

Part II. Simulation results

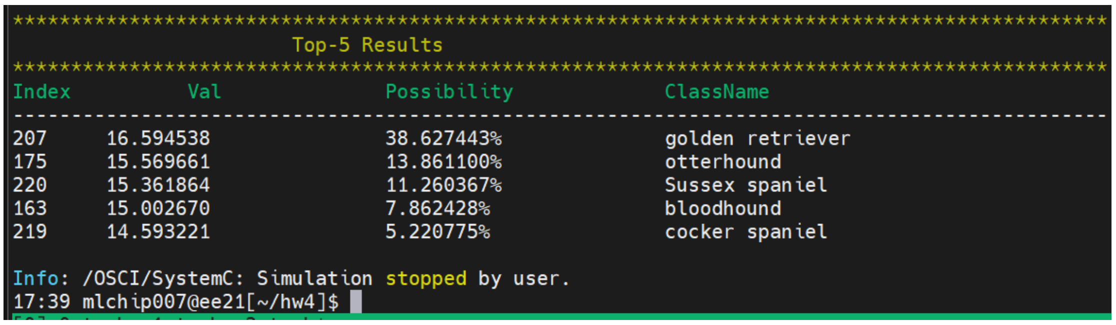


```
*********************************************************************************************

*********************************************************************************************
Index          Val                Possibility             ClassName
---------------------------------------------------------------------------------------------
285     20.206686                96.381450%              Egyptian cat
281     16.136835                1.646189%               tabby
282     15.733852                1.100187%               tiger cat
287     14.790856                0.428478%               lynx
728     14.411864                0.293315%               plastic bag

Info: /OSCI/SystemC: Simulation stopped by user.
4:23 mlchip007@ee21[~/hw4]$
```

Fig0 account: mlchip007, cat



```
***********************************************************************************************
                        Top-5 Results
***********************************************************************************************
Index          Val                Possibility              ClassName
-----------------------------------------------------------------------------------------------
207     16.594538                38.627443%               golden retriever
175     15.569661                13.861100%               otterhound
220     15.361864                11.260367%               Sussex spaniel
163     15.002670                7.862428%                bloodhound
219     14.593221                5.220775%                cocker spaniel

Info: /OSCI/SystemC: Simulation stopped by user.
17:39 mlchip007@ee21[~/hw4]$
```

Fig0 account: mlchip007, cat

Part III. Challenges and Observations

A. Test function by unit

Because loading weights , bias and image takes a lot of time, I try to split the whole design into two parts while implement it. One is loading all weights, bias and image. The other is inference the image. Because weights and bias will be stored in the corresponding PE, I write another "pe_load.h" , which also declare the PE class and point out how PE work, to trace the process and transmission of weights and bias.

It will directly read the weights/bias as golden_weights and golden_bias. After PE got its weights and bias from ROM, it will check them with golden_weights and golden_bias. If there exists error, it will print a fatal message in the screen.

data folder	Include weight, bias, and input image matrix
run	Executable files for SystemC program
controller.h	Implement controller
core.h	Implement core and Ni
pe.h	Implement the pe
pe_load.h	Debug use, for load parameters
router.h	Implement the router
Makefile	Makefile script for compile systemC program.
main.cpp	Declare the main function, create the module instances, mapping the signals. It includes all operation units, pattern module, clockreset modules, and monitor module.
ROM.h/ROM.o	ROM stores image and parameters
clockreset.h	Declare the clock module and reset modules.
clockreset.cpp	Implement the clock module and reset modules.