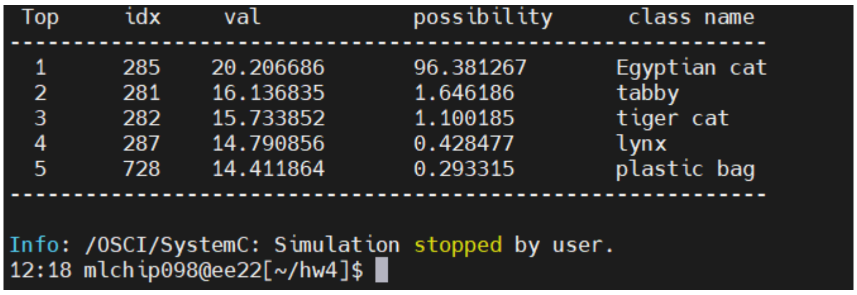
MLCHIP HW4

Integrate PEs into a NoC

Simulation Result

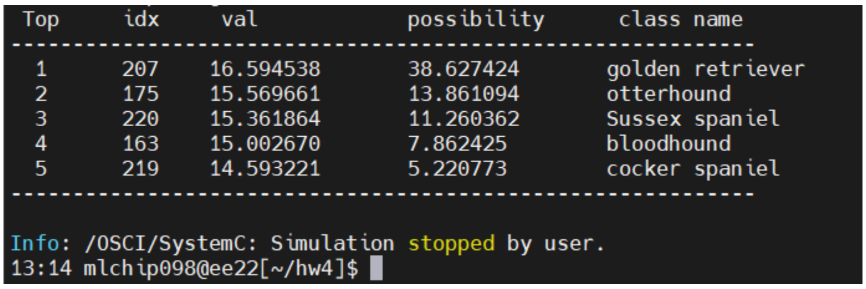
Cat:



```
 Top      idx      val             possibility       class name
-------------------------------------------------------------
  1       285     20.206686        96.381267        Egyptian cat
  2       281     16.136835        1.646186         tabby
  3       282     15.733852        1.100185         tiger cat
  4       287     14.790856        0.428477         lynx
  5       728     14.411864        0.293315         plastic bag
-------------------------------------------------------------

Info: /OSCI/SystemC: Simulation stopped by user.
12:18 mlchip098@ee22[~/hw4]$
```

Dog:

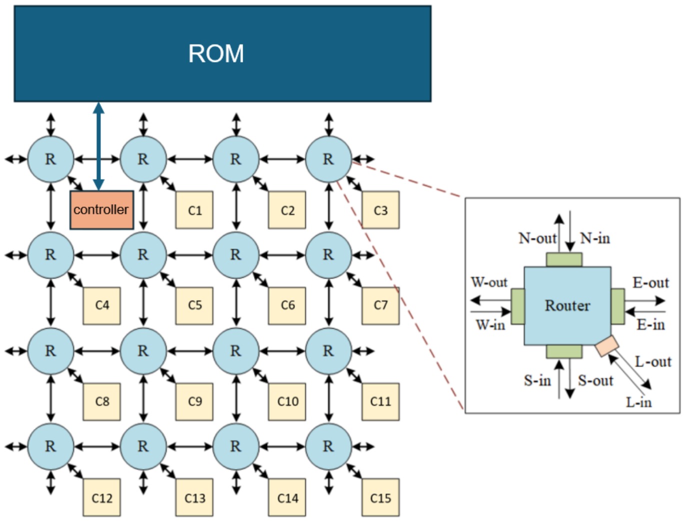


```
 Top      idx      val             possibility       class name
-------------------------------------------------------------
  1       207     16.594538        38.627424        golden retriever
  2       175     15.569661        13.861094        otterhound
  3       220     15.361864        11.260362        Sussex spaniel
  4       163     15.002670        7.862425         bloodhound
  5       219     14.593221        5.220773         cocker spaniel
-------------------------------------------------------------

Info: /OSCI/SystemC: Simulation stopped by user.
13:14 mlchip098@ee22[~/hw4]$
```

Implementation

NoC Router



Architecture	4x4 Torus NoC architecture

Routing algorithm	XY routing (deadlock free)
Switching	Wormhole switching
Buffer size	one flit (34 bits), depth=1
Virtual channel	No implementation
Flow control	ack/nack protocol (buffered flow control)

Based on the architecture used in HW3, I replace the core0 with the controller. Since I only separate the layers into 11 modules, some cores are not used and reducing the number of cores does not affect the overall performance.

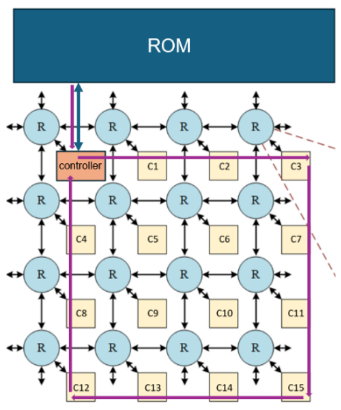
Flit Format

	33	32	31~28	27~24	23	22	21	20~0
header flit	1	0	source_id	dest_id	input	weight	bias	0
body flit	0	0	data value					
tail flit	0	1	data value					

When controller receives data from ROM, it starts organizing the packet. For the header flit, it contains the source_id, dest_id and use 3 bits to indicate whether the data type is input, weight or bias. This enables the cores to distribute them into different space and let PE calculate in a proper way. The controller is placed at the original core0 place, so the source_id is always 0 if the packet is sent by the controller. The dest_id is calculated by the controller or cores to send packet to the correct destination.

AlexNet

PE id	AlexNet Layer (details in HW1 report)
1	CONV_RELU_1
2	MAX_POOLING_1
3	CONV_RELU_2
7	MAX_POOLING_2
11	CONV_RELU_3
15	CONV_RELU_4
14	CONV_RELU_5
13	MAX_POOLING_3
12	LINEAR_RELU_1
8	LINEAR_RELU_2
4	LINEAR_3



The data flow circularly and the 4 cores at the center is not used in this work. With 11 layers, the data is just enough to walk around and back to the controller. This allows fewer data movement, saving time and energy.

File Description

File list	Description
../data/	Contain 2 input images (size: 224*224*3), weights, bias and classes.
Makefile	Automate the process of compiling and linking source code files. Use -O3 to speedup the simulation. It takes about 15 minutes to finish processing one image.
main.cpp	Connect all signals in different modules. Define sc_start. Can create waveform using sc_trace.
clockreset.h clockreset.cpp	Define clk and rst signals.
ROM.h ROM.cpp	Store data. The controller takes images, weights and bias from it. Change IMAGE_FILE_NAME if you want to process another image.
controller.h	Get data from ROM, packet them and send to appropriate location. After the AlexNet layers are finished, the result will be sent back to it. The controller does the SoftMax and print the result. The simulation will stop by sc_stop when the result of one image is printed.
router.h	Determine the direction of the flit. There may be many approaches, and my implementation is inherited from HW3.
core.h	Receive data and packet them. Send the packetized data to its PE and output the results when the PE finishes its calculation.
pe.h pe.cpp	Different AlexNet layers are mapped to different PEs (listed above). Initially, the dest_id is determined. When the core sends all necessary data to it, the calculation of the layer can begin.

Observation

- Controller design

Looking at the operation of ROM, I found that it transfers data continuously. At the beginning, I tried to send the received data out immediately, but it failed because the core cannot receive data every cycle in the handshake communication flow control. Some data are lost. Therefore, the data must be temporarily stored in the controller and only send out when the router is available.

- Segmentation fault (core dump)

  This happens when the program tries to access the memory space that is out of range. It may occur easily when we use a counter or pointer to point at a space in a vector. Thus, we should be careful when designing vector with dynamically changing size.