HW3

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

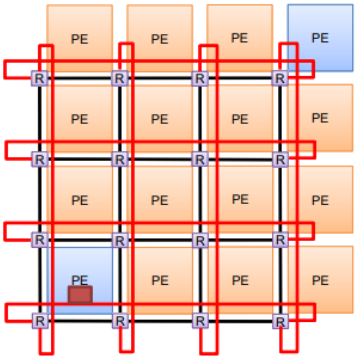

Fig.1 4x4 Torus mesh

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], and destination id[27:24]. The body flits begin with 2 zero and follow with data value. Tail flit consists of EoP[32] and the last data value.

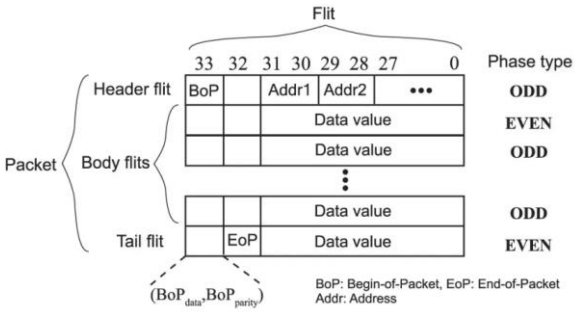

Fig2. Flit format

A. Core design:

It contains a PE and NI in each core. First, the core need to get the packet from PE. After fetching the packet, I split the source id, destination id, and data. 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.

B. 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

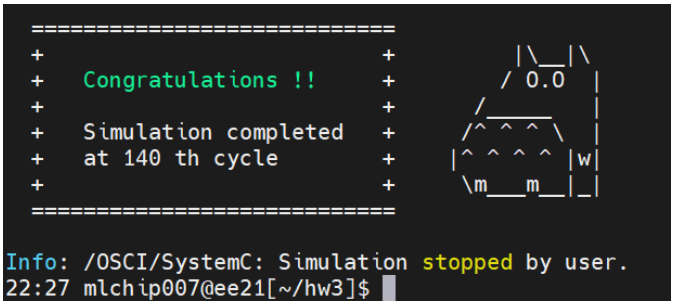

Fig0 account: mlchip007, 140 cycle

Part III. Challenges and Observations

A. Usage of std::queue< sc_lv <34> >

In the core design, I use “std::queue< sc_lv <34> >” to split and format the packet. This data type is actually similar to sc_fifo. I choose it here because this homework doesn’t limit the packet data size. Therefore, I can’t design some a fixed FIFO buffer size here according to some limitation.

B. Buffer size

In the Router design, I try to make the buffer size bigger from 2 to 100. However, the simulation of Demo pattern always cost 140 cycles to finish. This points out buffer size is not dominating problem here. In my design, I make all the transmission be continuous as possible. Thus, most part of transmission of a packet sends continuously unless at the buffer size is exploded.

To make the system run faster, I think it is required to build virtual channel, and even adopt some adapting routing algorithms.