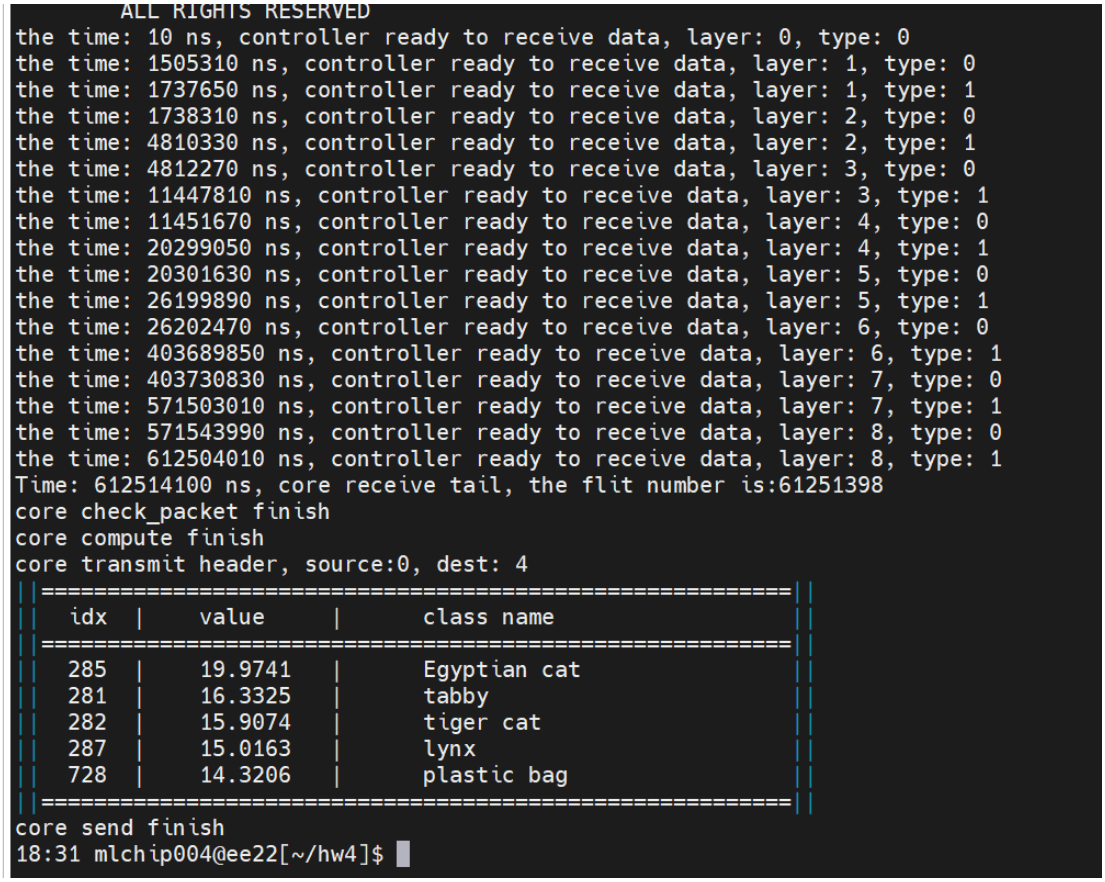
312510190 張家瑋

HW4 Report

Simulation results demonstrate the predicted output for the provided input data:



How do you design the router and NI? What routing algorithm do you use? What is the depth of the buffer? Do you use virtual channels?

I use the same communication protocol in both router and network interface, which is data_valid, data_ready signal in AXI-4:

```
    // to router0
    sc_out < sc_lv<34> > flit_tx;
    sc_out < bool > valid_tx;
    sc_in  < bool > ready_tx;

    // from router0
    sc_in  < sc_lv<34> > flit_rx;
    sc_in  < bool > valid_rx;
    sc_out < bool > ready_rx;
```

And in the controller, I make a decision that once the controller receive header from ROM, it can be transmit to the next hop at second cycle:

```
void receive_from_ROM_transmit_to_router(){
    //layer_id.write(cnt_layer);
    //layer_id_type.write(cnt_type);
    layer_id_valid.write(0);

    if(data_valid.read()){
        //cout<<"the time: "<<sc_time_stamp()<< "controller receive data"<<data.read()<<endl
        sc_lv<34> flit_out;
        sc_dt::scfx_ieee_float tx_temp(data.read());

        sc_lv<32> flit_out_sc_lv = convert_to_sc_lv(tx_temp);
        flit_out.range(31, 0) = flit_out_sc_lv;
        flit_out.range(33, 32) = "00";
        flit_tx.write(flit_out);
        //cout<<"tx_cnt"<<tx_cnt<<", "<<"flit_out:"<<flit_out<<endl;
        valid_tx.write(true);
    }
    else{
        //cout<<"the time: "<<sc_time_stamp()<< " test"<<endl;
        ready_to_receive ^= 1;
        valid_tx.write(false);
    }
```

In the router, I use XY-routing algorithm in this homework, each router has a input buffer size of 8, and I found that virtual channel is no need in this case:

```
void xy_routing(){

    //1: East output channel
    // xy routing
    if (out_busy[1] == 0) {
        bool found_flit = false;
        // Iterate over all input buffers
        for (int i = 0; i < 5; i++) {
            // output selection
            if (buffer_empty[i] == 0) {
                sc_lv<34> flit = flit_buffer[i][r_addr_buf[i].range(2,0)];
                int xy_route_direct = int((sc_uint<4>)(sc_lv<4>)flit.range(27,24)) % 4;
                bool is_head_flit = flit[33] == 1;
                if (xy_route_direct > Router_ID%4 && is_head_flit) {
                    channel_select[1] = i;
                    out_valid_tmp[1] = 1;
                    found_flit = true;
                    break;// found the flit
                }
            }
```

implementation approach:

at core.h, I divide module into two parts: handle_receive and handle transmit function. And if input from router is finished, neuron computing task start

```
52      // Main process
53      void run() {
54          if (rst.read() == 1) {
55              initialize();
56          } else {
57              handle_receive();
58              if(input_finish){
59                  conv1();
60                  input_finish = false;
61              }
62              handle_transmit();
63          }
64      }
```

For transmission protocol, I use valid and ready flag in AXI-4 protocol to be decide whether the signal can be deliver or not

```
12      sc_in<sc_lv<34>> flit_rx;  // Input channel
13      sc_in<bool> valid_rx;      // Input channel valid signal
14      sc_out<bool> ready_rx;     // Input channel ready signal
15
16      sc_out<sc_lv<34>> flit_tx; // Output channel
17      sc_out<bool> valid_tx;     // Output channel valid signal
18      sc_in<bool> ready_tx;      // Output channel ready signal
19
```

At controller.h, the controller manages different states using counters and flags (cnt_layer, cnt_type, ready_to_receive, tx_cnt) to track the current layer, data type, and readiness to receive or transmit data.

```
88      if(ready_to_receive){
89          cout<<"the time: "<<sc_time_stamp()<< ", controller ready to receive data, layer: "<<cnt
90          layer_id.write(cnt_layer);
91          layer_id_type.write(cnt_type);
92          layer_id_valid.write(1);
93          if(cnt_layer == 0){
94              cnt_layer++;
95          }
96          else if(cnt_type==0){
97              cnt_type=1;
98          }
99          else{
100             cnt_type=0;
101             cnt_layer++;
102             if(cnt_layer == 9){
103                 layer_id_valid.write(0);
104                 ready_to_receive = 0;
105             }
106         }
107     }
```

The receive_from_ROM_transmit_to_router() method handles both receiving data from the ROM and transmitting it to the router, ensuring smooth data flow. The receive_from_router0() method handles the reception of flits from the router and processes them accordingly:

```
53    void receive_from_ROM_transmit_to_router(){
54        //layer_id.write(cnt_layer);
55        //layer_id_type.write(cnt_type);
56        layer_id_valid.write(0);
57
58        if(data_valid.read()){
59            //cout<<"the time: "<<sc_time_stamp()<< "controller receive data"<<data.read()<<endl;
60            sc_lv<34> flit_out;
61            sc_dt::scfx_ieee_float tx_temp(data.read());
62
63            sc_lv<32> flit_out_sc_lv = convert_to_sc_lv(tx_temp);
64            flit_out.range(31, 0) = flit_out_sc_lv;
65            flit_out.range(33, 32) = "00";
66            flit_tx.write(flit_out);
67            //cout<<"tx_cnt"<<tx_cnt<<", "<<"flit_out:"<<flit_out<<endl;
68            valid_tx.write(true);
```

For Routing and Channel Selection, channel_select[5]: Keeps track of the selected channel for each output. out_busy[5], out_valid_tmp[5]: Status flags to manage output channels.

```
112       // Output selection for the local output channel
113       if (out_busy[0] == 0) {
114           bool found_flit = false;
115           // Iterate over all input buffers
116           for (int i = 0; i < 5; i++) {
117               // output selection
118               if (buffer_empty[i] == 0) {
119                   sc_lv<34> flit = flit_buffer[i][r_addr_buf[i].range(2,0)];
120                   int flit_router_id = int((sc_uint<4>)(sc_lv<4>)flit.range(27,24));
121                   bool is_head_flit = flit[33] == 1;
122
123                   if (flit_router_id == Router_ID && is_head_flit) {
124                       channel_select[0] = i;
125                       out_valid_tmp[0] = 1;
126                       found_flit = true;
127                       break;// found the flit
128                   }
129               }
130           }
131           // If no valid flit found, set the output to invalid
132           if (!found_flit) {
133               channel_select[0] = 5;
134               out_valid_tmp[0] = 0;
135           }
136       } else {
137           // Output channel is busy, retain current buffer selection
138           channel_select[0] = channel_select[0];
139           out_valid_tmp[0] = 1;
140       }
```

challenges faced:

challenge faced in this task is the communication between controller and core. Since the input buffer size is limited, it's may occur state of buffer full, so dealing with the packet transmitting is a problem. And controller receive from ROM need a two cycle gap, so I use XOR operation to make the data receive successfully.

```
59          if(data_valid.read()){
60              //cout<<"the time: "<<sc_time_stamp()<< "controller receive data"<<data.read()<<endl;
61              sc_lv<34> flit_out;
62              sc_dt::scfx_ieee_float tx_temp(data.read());
63
64              sc_lv<32> flit_out_sc_lv = convert_to_sc_lv(tx_temp);
65              flit_out.range(31, 0) = flit_out_sc_lv;
66              flit_out.range(33, 32) = "00";
67              flit_tx.write(flit_out);
68              tx_cnt++;
69              //cout<<"tx_cnt"<<tx_cnt<<", "<<"flit_out:"<<flit_out<<endl;
70              valid_tx.write(true);
71          }
72          else{
73              //cout<<"the time: "<<sc_time_stamp()<< " test"<<endl;
74              ready_to_receive ^= 1;
75              valid_tx.write(false);
76          }
```

other observations or insights gained:

implement a Controller module that manages the communication between a ROM and a router in an NoC system, ensuring proper data flow and processing. The goal of this homework include data conversion, get neurons into PEs, state management, data reception and transmission, and result processing. The result shows efficient and reliable data transfer and processing within the NoC system.