OASIS NoC Architecture Design in Verilog HDL

Technical Report: TR-062010-OASIS

Written by

Kenichi Mori

ASL-Ben Abdallah Group
Graduate School of Computer Science and Engineering
The University of Aizu

Outline

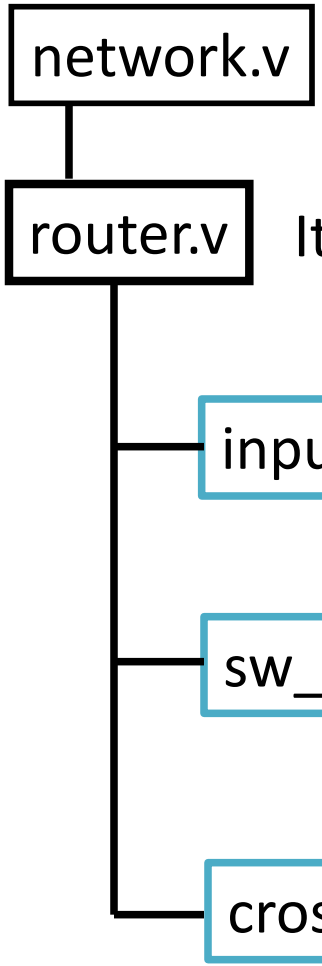
- Network-on-Chip OASIS NoC Overview
- Hardware Design Detail
  - **Network** design
  - **A router** design
  - Input port design
    - Buffering, and **routing mechanism**
  - Arbiter design
    - **Scheduling**, and **Stall/Go flow control** mechanism
  - Crossbar design
    - Transmission mechanism
- Design tools and Results
- Conclusion

Network-on-Chip configurations

To configure Network-on-Chip, some parameters can be selected

- *Topology*
  - Direct(torus, mesh), Indirect(Fat tree, butterfly), and Irregular.
- *Routing Algorithms*
  - Deterministic (Destination-tag, XY-routing), Oblivious (minimal oblivious), and Adaptive (minimal, non-minimal)
- *Flow control mechanisms*
  - Credit-based, ON/OFF, ACK/NACK, Handshaking.
- *Forwarding methodology*
  - Wormhole-Switching, Store and Forwarding, and Virtual-Cut-Through
- *Packet and buffer size*
  - It can be selected appropriate size with trade-off between Latency and area utilization.

Design Level



It decides connection between routers, and also topology

It connects under level modules

It decides direction of next port by using XY-routing
It issues stop signal depends on buffer condition

It works as a scheduler and a flow control.
-scheduler is needed for output occupation
-flow control is needed for avoiding packet send miss.

It transmits flits appropriate output port.

OASIS parameters selection

OASIS parameters	Value
Network size	4x4-mesh
Routing algorithm	deterministic XY-routing
Flow control mechanism	Stall/Go
Forwarding method	Wormhole Switching
Flit size	N+12 bit (Header: 12bit, Payload: N bit)
Buffer Depth	4

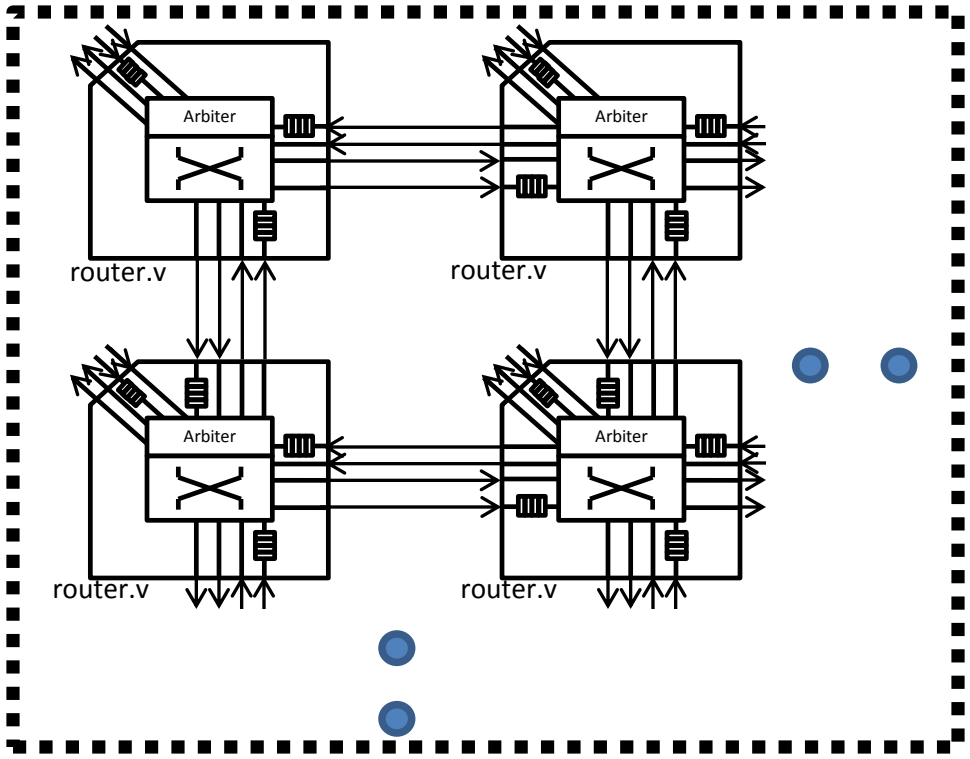
Network (1/2)

```
module network(clk, reset,
               data_in, data_out,
               stop_in, stop_out);

  //network size
  parameter X_WIDTH = 4;
  parameter Y_WIDTH = 4;

  //router parameters
  parameter NOUT          = 5;
  parameter FIFO_DEPTH    = 4;
  parameter FIFO_LOG2D    = 2;
  parameter FIFO_FULL_LVL = 2;


  input                                clk, reset;

  input [X_WIDTH*Y_WIDTH*`WIDTH-1:0]   data_in;
  input [X_WIDTH*Y_WIDTH-1:0]          stop_in;

  output [X_WIDTH*Y_WIDTH*`WIDTH-1:0]  data_out;
  output [X_WIDTH*Y_WIDTH-1:0]         stop_out;
```

4x4 network size
Total data input size is declared.

Control wire and ***data wire*** are separated.

network.v

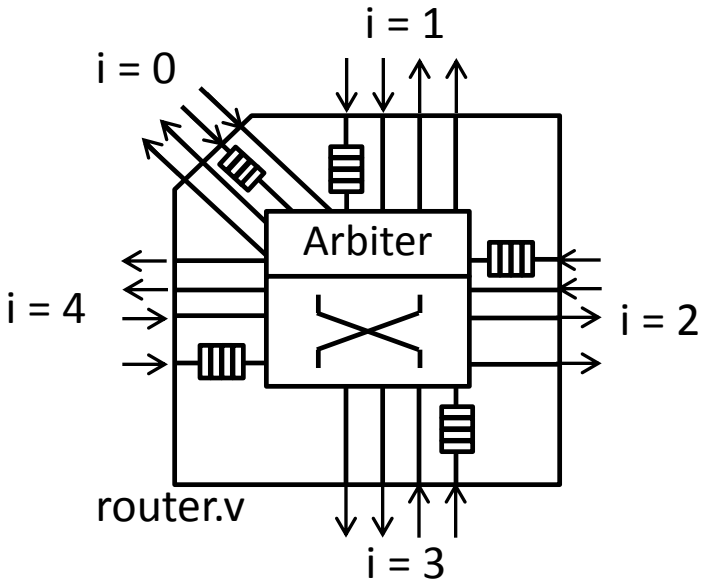


Network (2/2)

```
//north edge of router
if(i==1) begin
   if(y_pos==Y_WIDTH-1) begin
      assign net_data_in [x_pos][y_pos]['WIDTH*(i+1)-1:'WIDTH*i] = 0;
      assign net_stop_in [x_pos][y_pos][i] = 1'b1;
   end else begin
      assign net_data_in [x_pos][y_pos]['WIDTH*(i+1)-1:'WIDTH*i]
                = net_data_out[x_pos][y_pos+1]['WIDTH*(3+1)-1:'WIDTH*3];
      assign net_stop_in [x_pos][y_pos][i] = net_stop_out[x_pos][y_pos+1][3];
   end
end
```

Network connection is set here.

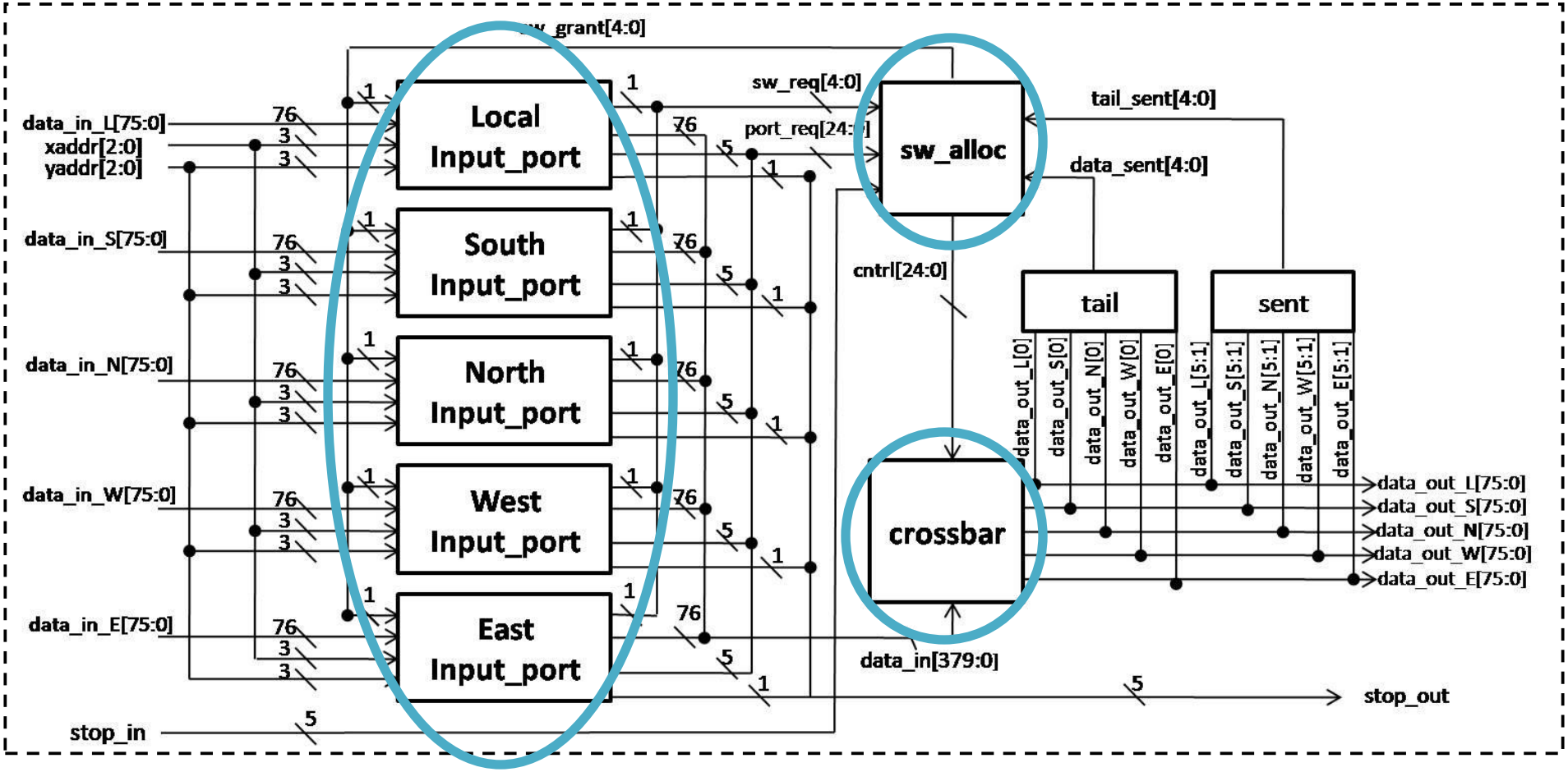
- Each routers connects together.
- The pattern of router's connection is 5.
  - -Local, North, East, South, and West



Outline

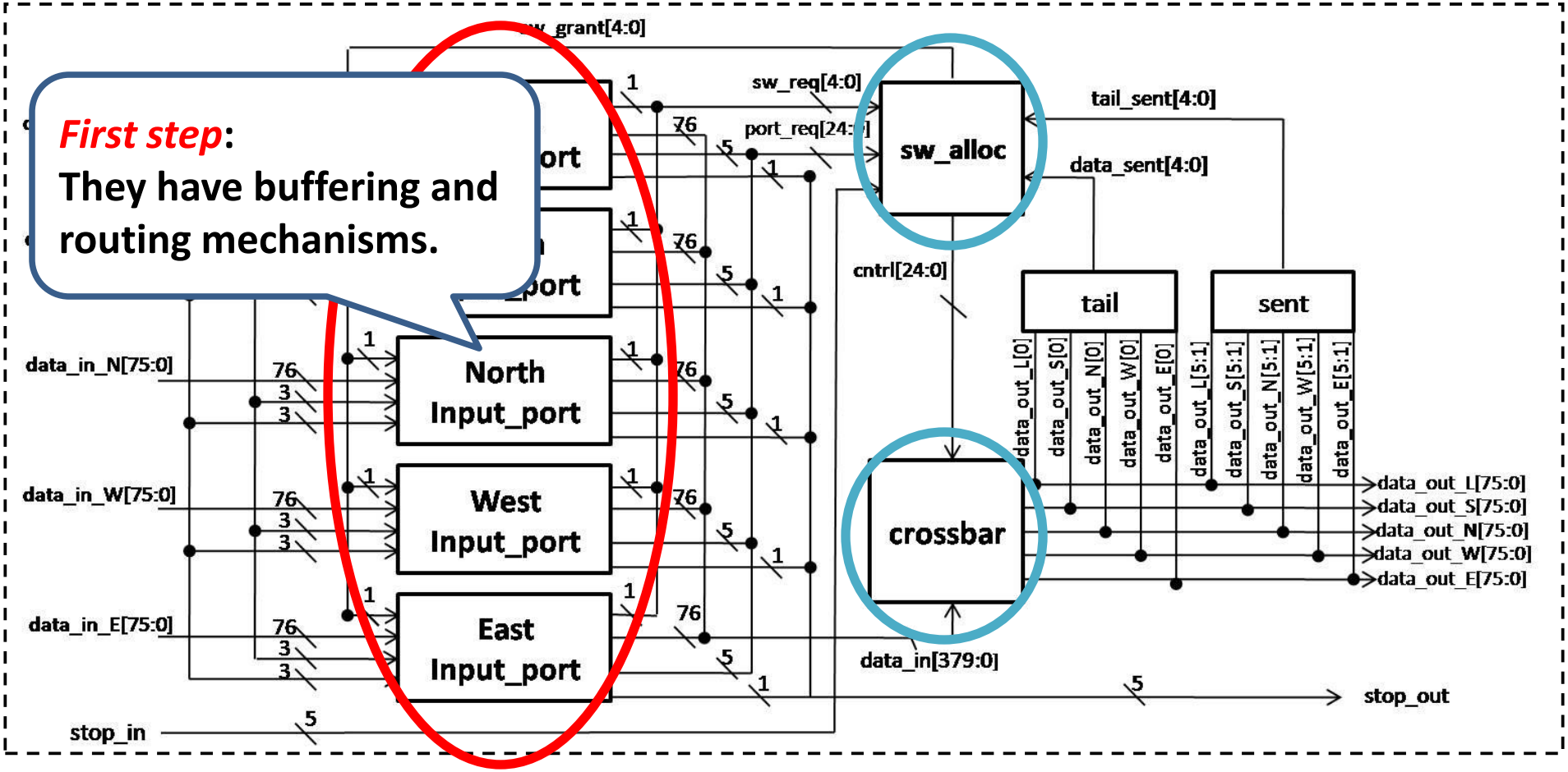
- Network-on-Chip OASIS NoC Overview
- Hardware Design Detail
  - **Network** design
  - **A router** design
  - Input port design
    - Buffering, and **routing mechanism**
  - Arbiter design
    - **Scheduling**, and **Stall/Go flow control** mechanism
  - Crossbar design
    - Transmission mechanism
- Design tools and Results
- Conclusion

One router design



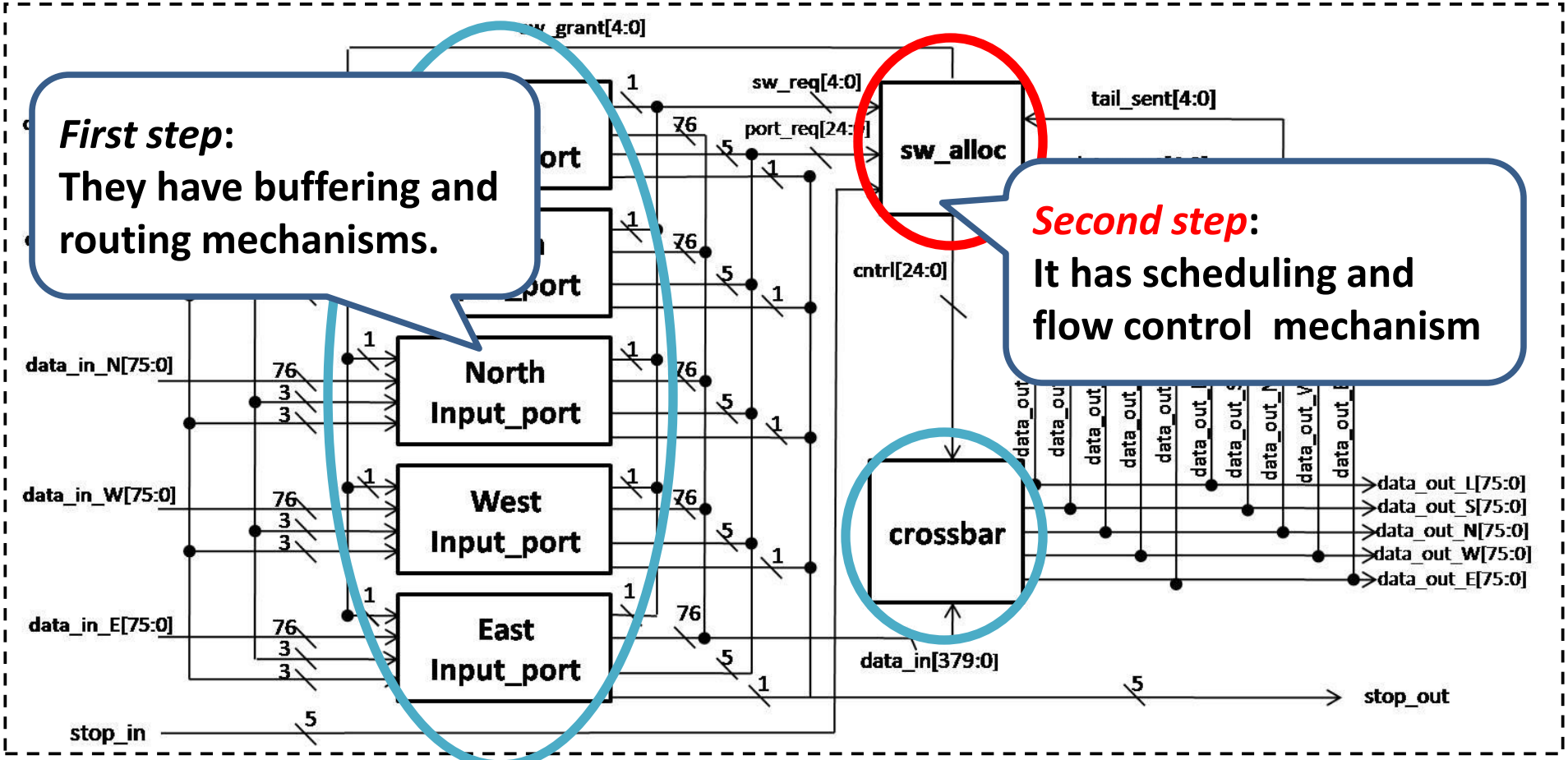
One router has important **three** elements.

One router design



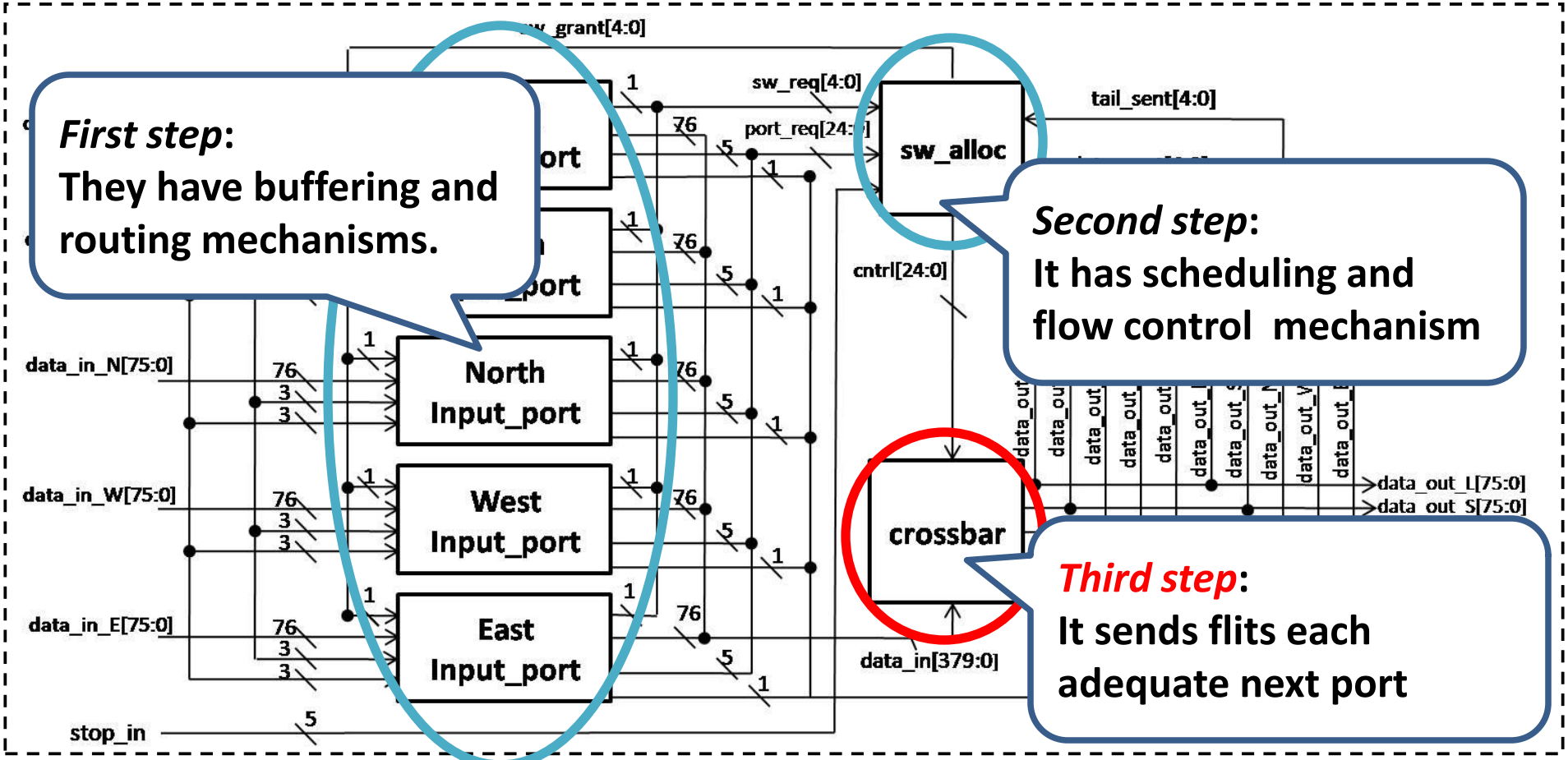
One router has important **three** elements.

One router design



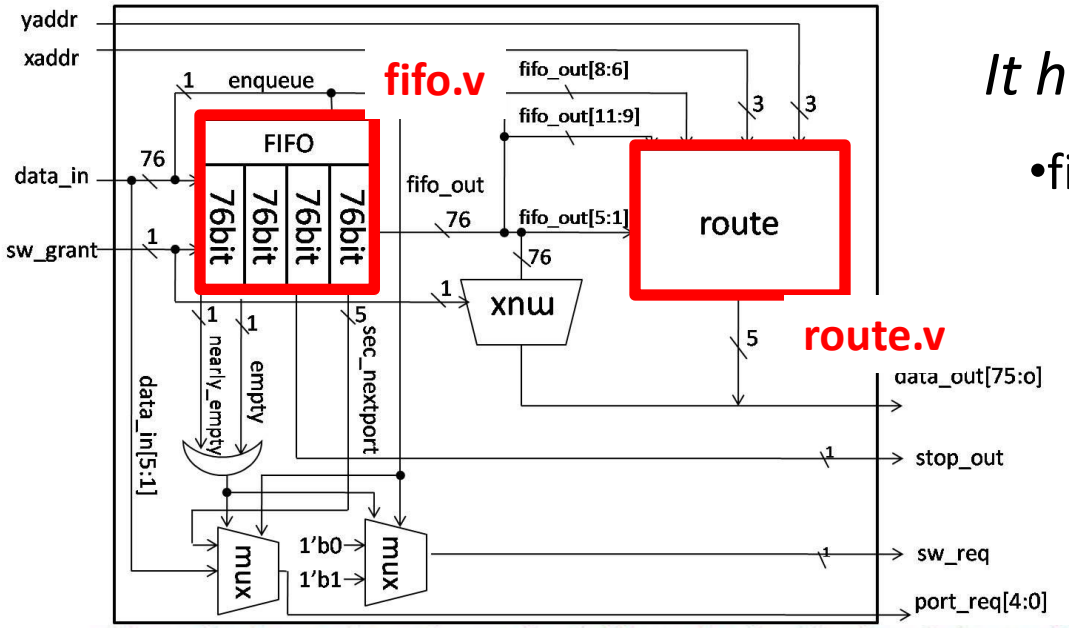
One router has important **three** elements.

One router design



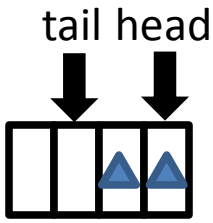
One router has important **three** elements.

Input port design(fifo)



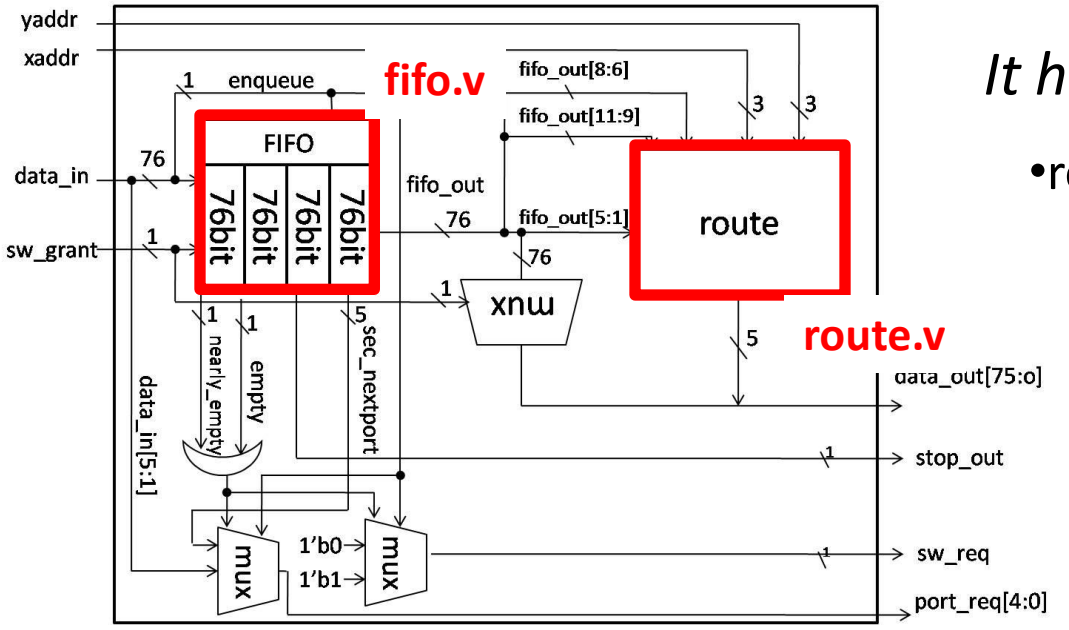
*It has **fifo** and routing modules*

- fifo.v
  - -It has pointers for queue systems
  - -It makes **stop signal** for flow control




```
// evaluate empty and nearly_full control signals. this could be done combinationally?
if ((((head_ptr + 1'b1)==tail_ptr) & dequeue & !enqueue) | (empty & !enqueue)) empty <= 1'b1;
if (empty & enqueue) empty <= 1'b0;

//evaluate stop signal - obtained sequentially to ensure it arrives at upstream
//router, early in it's clock cycle
if (((tail_ptr + FULL_LVL[LOG2D-1:0] + 1'b1)==head_ptr) && enqueue && !dequeue)
  stop_out <= 1'b1;
if (((tail_ptr + FULL_LVL[LOG2D-1:0])==(head_ptr+1'b1)) && !enqueue && dequeue)
  stop_out <= 1'b1;
if ((tail_ptr + FULL_LVL[LOG2D-1:0])==head_ptr)
  if ((enqueue && !dequeue) || (!enqueue && dequeue))
    stop_out <= 1'b0;
```

Input port design(routing)



*It has fifo and **routing** modules*

- route.v

  -It decides transaction direction to use the current address and destination address.

```
//assign next addresses
if (nextport == `EAST) next_xaddr = xaddr + 1'b1;
  else if (nextport == `WEST) next_xaddr = xaddr - 1'b1;
    else next_xaddr = xaddr;

if (nextport == `NORTH) next_yaddr = yaddr + 1'b1;
  else if (nextport == `SOUTH) next_yaddr = yaddr - 1'b1;
    else next_yaddr = yaddr;
```

First, next address is found

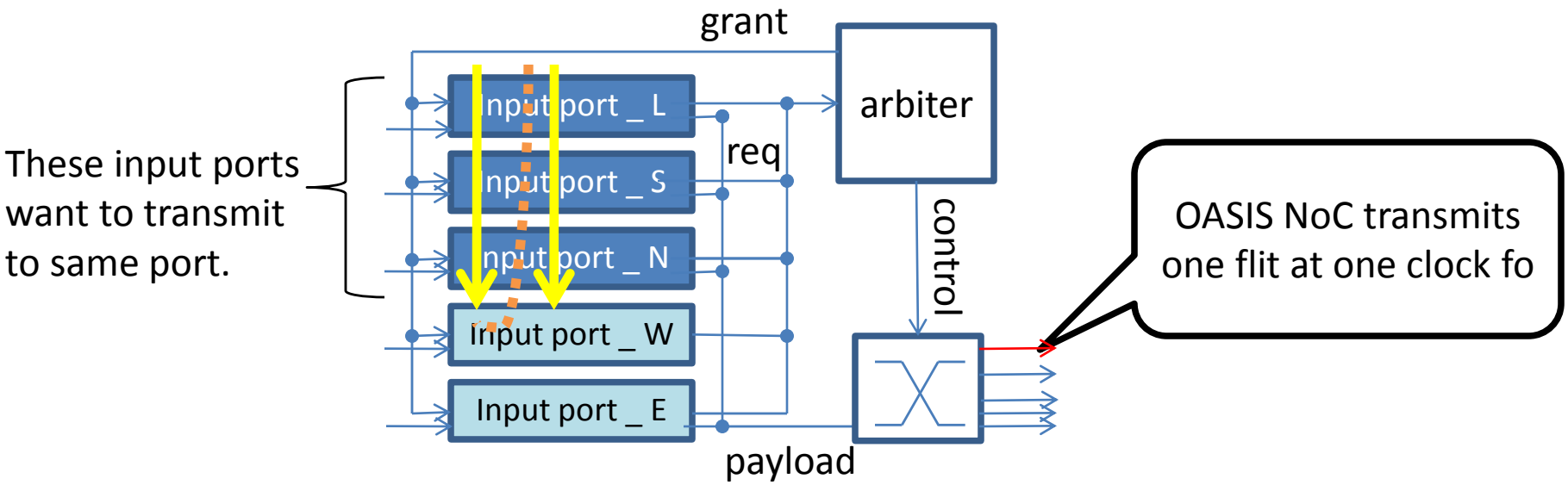
```
//evaluate next port
if (next_xaddr == xdest) begin
   if (next_yaddr == ydest) route = `SELF;
     else if(next_yaddr < ydest) route = `NORTH;
       else route = `SOUTH;
end else begin
   if (next_xaddr < xdest) route = `EAST;
     else route = `WEST;
end
```

Next port is decided by using next address

Outline

- Network-on-Chip OASIS NoC Overview
- Hardware Design Detail
  - **Network** design
  - **A router** design
  - Input port design
    - Buffering, and **routing mechanism**
  - Arbiter design
    - **Scheduling**, and **Stall/Go flow control** mechanism
  - Crossbar design
    - Transmission mechanism
- Design tools and Results
- Conclusion

Why scheduling is needed?

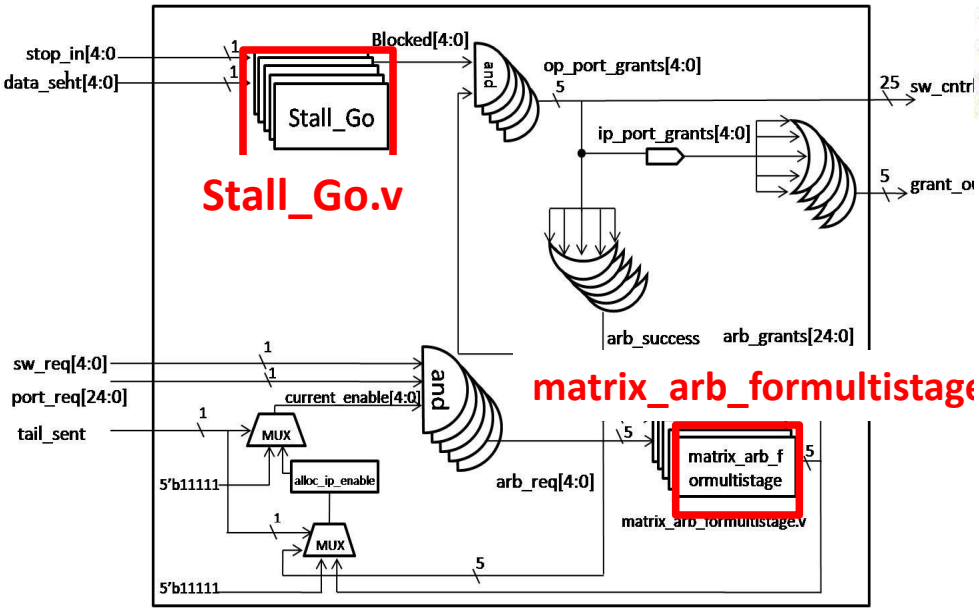


Output bandwidth is limited for one flit data size.

Scheduling is needed

OASIS NoC supports **Round-Robin** scheduling.

Arbiter design(scheduling)

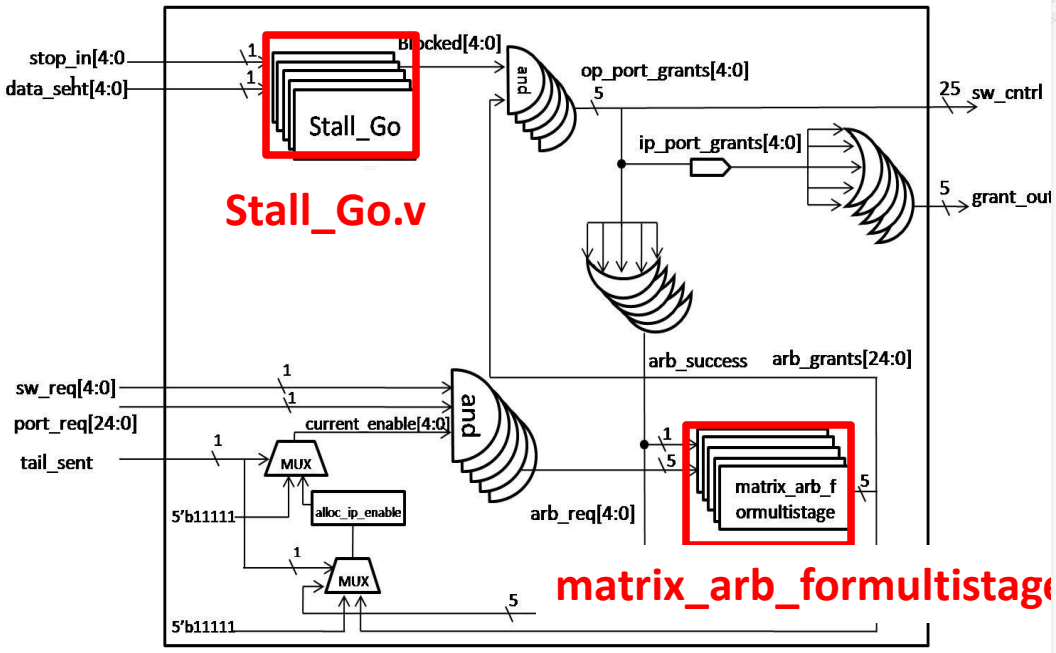



```
//
// generate grants
//
generate
for (i=0; i<SIZE; i=i+1) begin:ol1
   // generate grant i
   for (j=0; j<SIZE; j=j+1) begin:il1
  if (j==i)
    // request i wins if requesting and....
    assign pri[i][j]=request[i];
  else
    // ....no other request with higher priority
    if (j>i)
      // j beats i
      assign pri[i][j]=!(request[j]&&state[j*SIZE+i]);
    else
      // !(i beats j)
      assign pri[i][j]=!(request[j]&&!state[i*SIZE+j]);
   end

   assign grant[i]=&pri[i];
end
```

-Right code indicates comparison of priority between current transmitting input port and other routers which send request to arbiter.

```
// update state
//
generate
for (i=0; i<SIZE; i=i+1) begin:ol2
   for (j=0; j<SIZE; j=j+1) begin:il2
  assign new_state[j*SIZE+i]=(success&&((state[j*SIZE+i]&&!grant[j])
                             ||(grant[i])))||(!success&&state[j*SIZE+i]);
   end
end
endgenerate
```

Arbiter design(flow control)



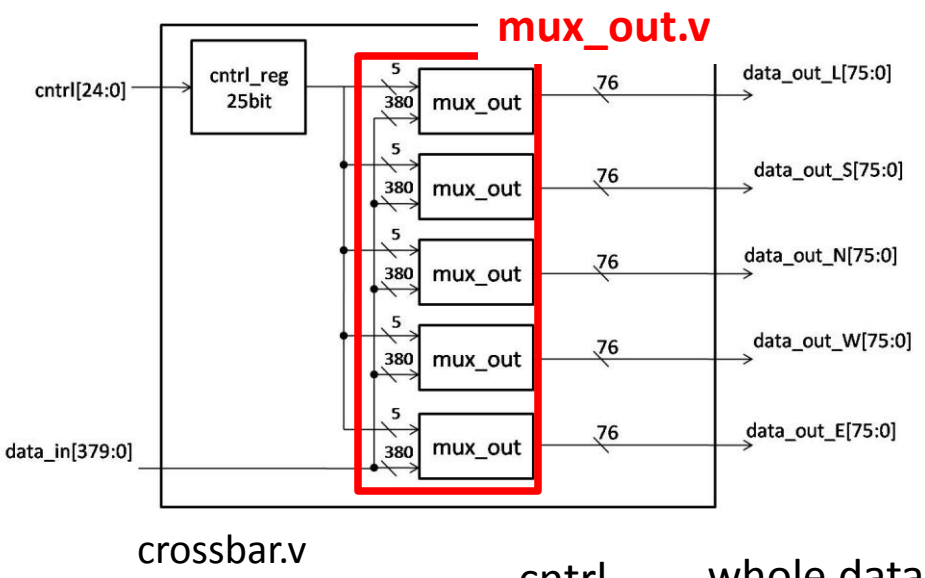

```
module stall_go(clk, reset, data_sent,
                stop_in, blocked);

  input clk, reset;
  input data_sent, stop_in;

  output blocked;

  reg [1:0] state;

  always @(posedge clk) begin
     if (!reset) begin

        if ((state=='GO) && stop_in && data_sent)
          state <= 'SENT1;

        if (state=='SENT1) begin
           if (stop_in && !data_sent)
             state <= 'GO;
           if (!stop_in && data_sent)
             state <= 'STOP;
        end

        if ((state=='STOP) && stop_in)
          state <= 'GO;

     end else
       state <= 'GO;
  end
```

stall_go.v treats Stall_go flow control. It has state machine to decide when it issues stall signal.

Outline

- Network-on-Chip OASIS NoC Overview
- Hardware Design Detail
  - **Network** design
  - **A router** design
  - Input port design
    - Buffering, and **routing mechanism**
  - Arbiter design
    - **Scheduling**, and **Stall/Go flow control** mechanism
  - Crossbar design
    - Transmission mechanism
- Design tools and Results
- Conclusion

Crossbar design



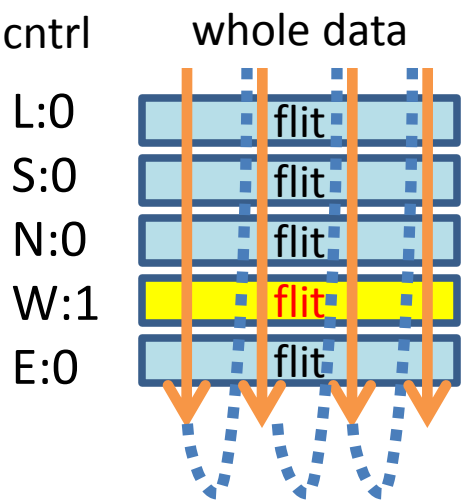
crossbar.v

```
generate

   //loop over each bit of data
   for (i=0;i<WIDTH;i=i+1) begin:bit_loop
      assign data_out[i] = mux(cntrl, data_bits[i]);

      //loop over each input channel
      for (j=0;j<n_in;j=j+1) begin:input_loop
         assign data_bits[i][j] = data_in[WIDTH*j+i];
      end

   end

endgenerate
```

```
function mux;

   input [n_in-1:0] cntrl;
   input [n_in-1:0] data_in;

   integer i;

   begin
      mux = 0;

      for (i=0; i<n_in; i=i+1) begin
         if(cntrl[i] == 1'b1) mux = data_in[i];

      end
   end
endfunction // mux
```

It transmits flits to neighbor routers.

-cntrl signal indicates which direction is destination.



Design Tools and Results

Design environments

- Verilog HDL is used.
- Quartus II ver. 9.0
- Target device
  - Family : Stratix III EP3SL150F1152C2
- Flit's payload is 8bit.

OASIS NoC hardware results

- Area (ALUTs):5,485(5%)
- Power (mW):649.17
- Speed (MHz):185.87

Module	Line
network.v	133
router.v	72
inputport.v	113
fifo.v	100
route.v	64
sw_alloc.v	109
matrix_arb.v	111
stall_go.v	56
crossbar.v	44
mux_out.v	54
Total	856