**Router**

# **1. Mesh Coordinate Numbers and Channels**

We are using a 4X4 mesh structure and XY routing algorithm in the router. Hence, the X address would be from 0 to 3, and the Y address from 0 to 3.

The coordinate (0,0) is set at the left bottom corner. (3,3) is set at the right top corner.

Here is the diagram:

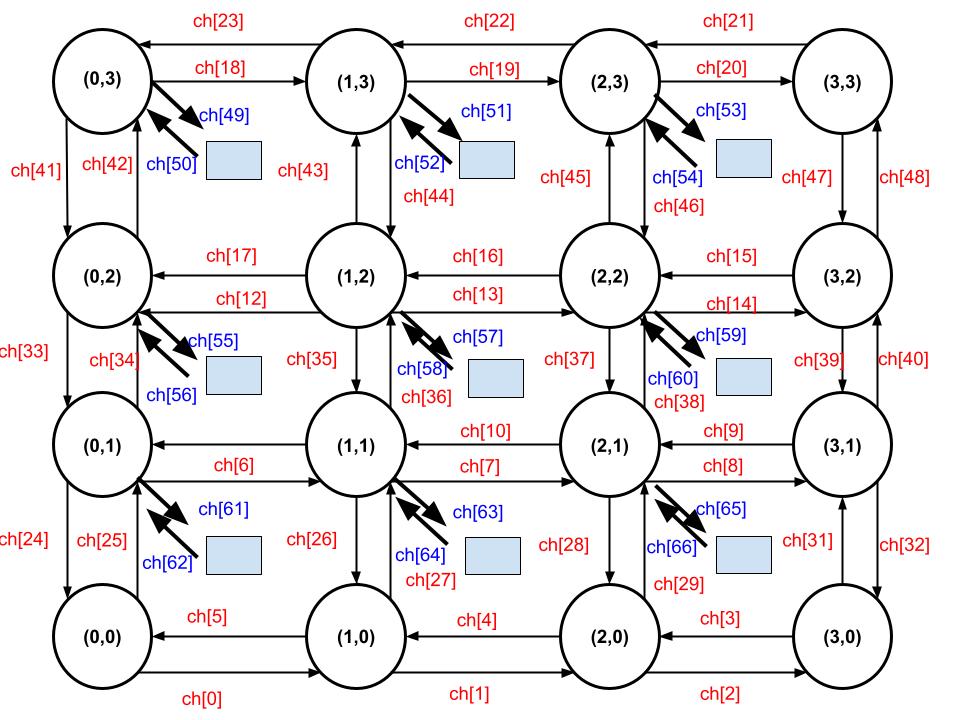


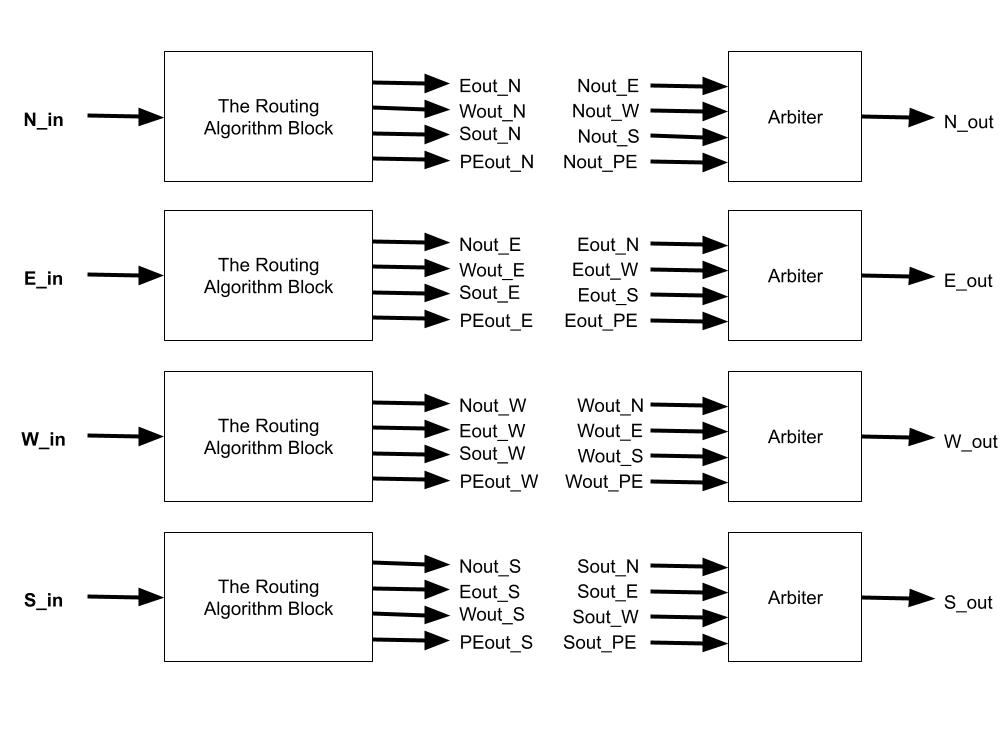
Fig. 4X4 Mesh with Channel Numbers

Based on our mesh structure, we use 4 bits as source address, showing X and Y addresses contain 2 bits respectively. Destination Address of X and Y contains 2 bits respectively.

Hence, we use 8 bits to define the packet routing address.

# **2. Router Submodules**

The most important part of the router is the routing algorithm. It can be implemented in many ways. Here are my explanation and the structure.



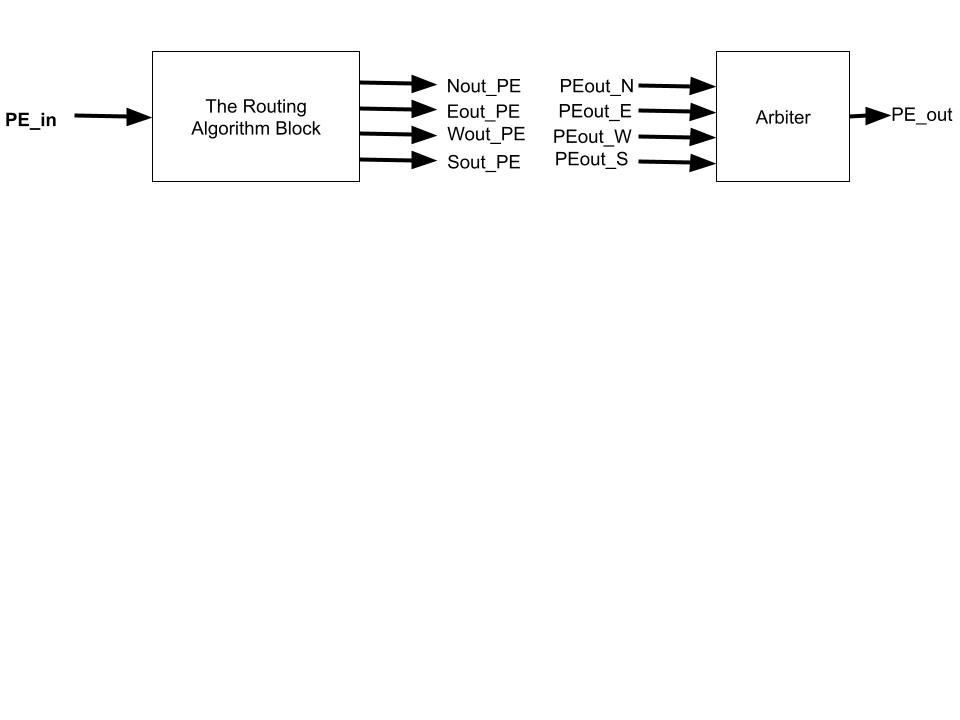


Fig. Router Architecture

First, for a router, it has 5 directions that need to be considered, PE, North, East, West, and South. Each direction has input and output. Hence, I define 10 interfaces to implement these bidirectional transmissions in each direction.

But to reduce the level of debugging, I separate each direction as a module. It means different modules handle the packet coming from North, East, West, South, and PE direction respectively. Also, I have 5 same 4-input arbiters to decide which packet goes to the output port first.

About the name of internal channels.Let’s take the North submodule for example, if this packet is received from the North interface and it will go to PE, I name this channel as PEout\_N, and so on.

Let me explain what the ideas are in these blocks and how each block works.

## **(A)North Routing Algorithm Block**

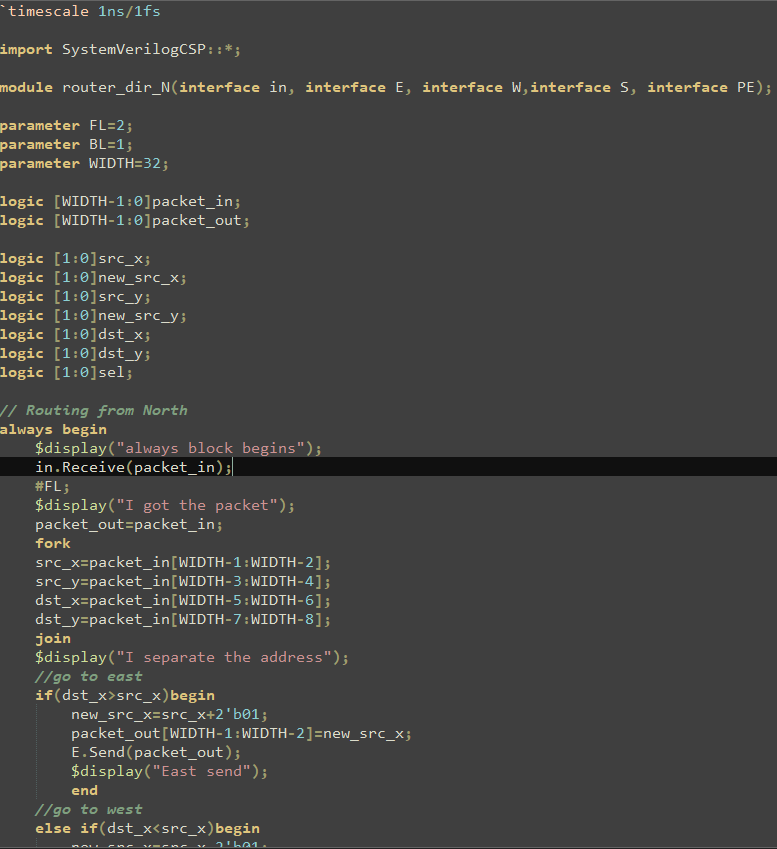


Fig. North Code 1

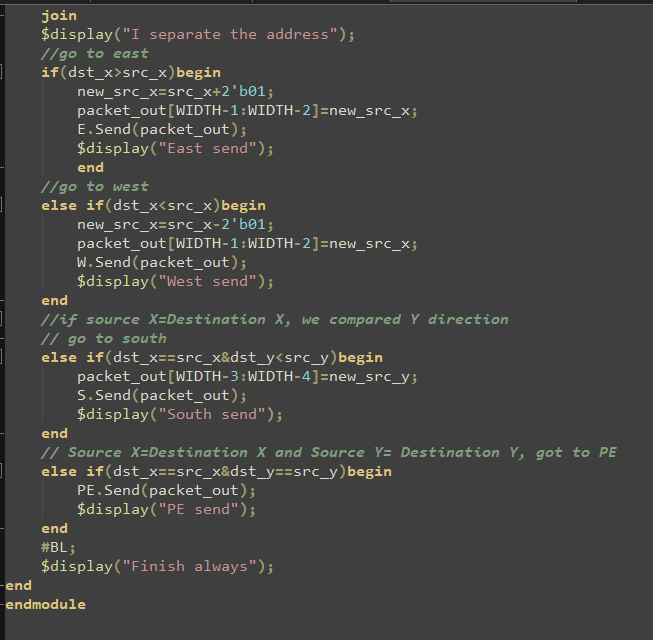


Fig. North Code 1

In this algorithm, I dispacketize the packet to get the source and destination address. After I had the address, I compared the source and destination address. First, I compared the X direction (West and East). If the value of destination X address is larger or smaller than the value of source address, which means the destination position is on the right or left of source position respectively, the packet goes to East or West.

Later, if the source and destination X addresses are the same, I compared the Y address. If the destination Y address is smaller than the source address, the packet goes to the south direction. If the source and destination Y addresses are the same, the packet goes to the function blocks, such as PE or the threshold block.

After I determine the direction where the packet should go, I change the source address.

If the packet goes to East, I add one to the source X address. If the packet goes to West, I subtract one to the source X address. If the packet goes to South, I add one to the source Y address. If the packet goes to PE, I do not change the source address. Before the packet leaves the PE block, I update the destination address in PE.

The reason why the routing algorithm block does not have the North direction is that it is useless and redundant because a packet from the North arrives at this router and this router sends the packet back to the Northern router. It means the packet does not move forward. Hence, the routing algorithm block has 4 output directions.

## **(B)East Routing Algorithm Block**

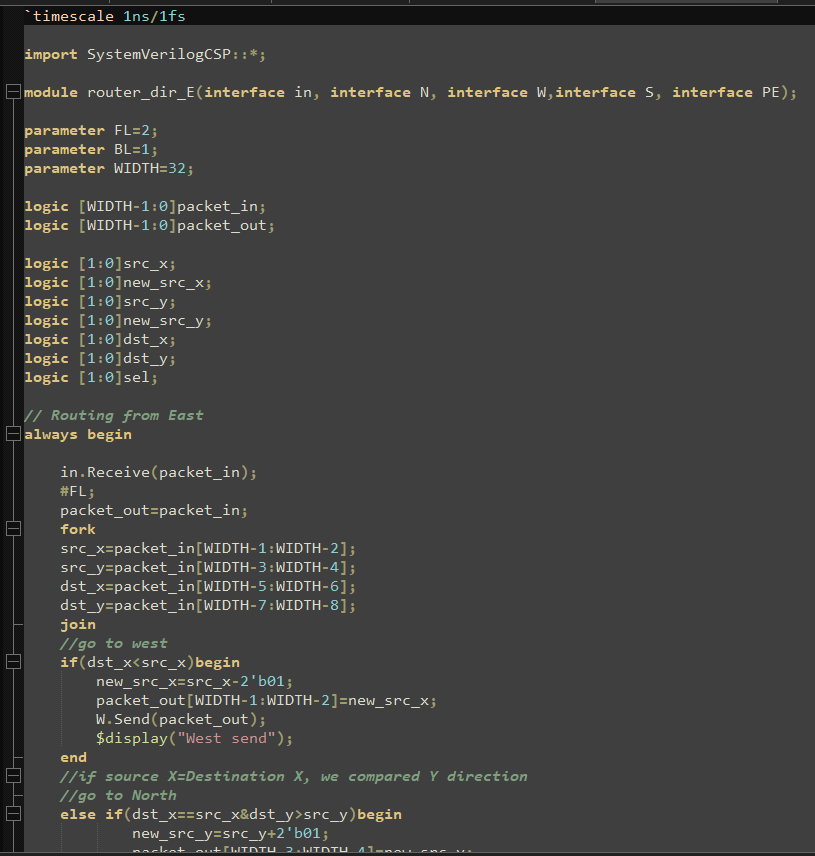


Fig. East Code 1

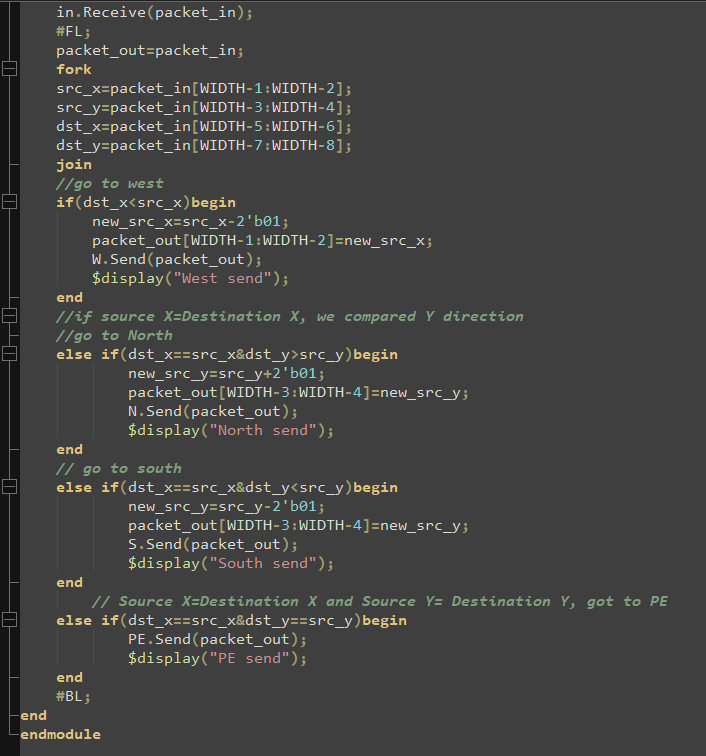


Fig. East Code 2

It is the same concept. First, the east algorithm block does not have eastern direction output. Also, I analyze the source and destination X and Y address. If the destination X address is smaller than the source X address, the packet goes to West.

Later, if the source and destination X addresses are the same, I compared the Y address. If the destination Y address is larger or smaller than the source address, the packet goes to the North or South respectively. If the source and destination Y addresses are the same, the packet goes to the function blocks, such as PE or the threshold block.

After I determine the direction where the packet should go, I change the source address.

If the packet goes West, I subtract one to the source X address. If the packet goes North, I add one to the source Y address.If the packet goes South, I subtract one to the source Y address. If the packet goes PE, I do not change the source address. Before the packet leaves the PE block, I update the destination address in PE.

## **(C)West Routing Algorithm Block**

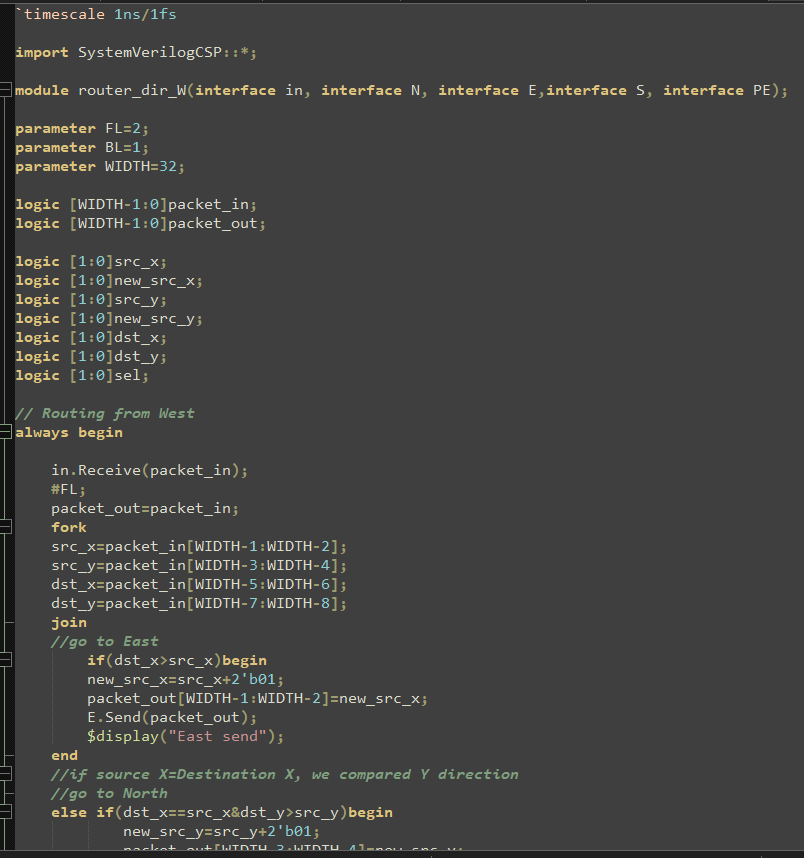


Fig. West Code 1

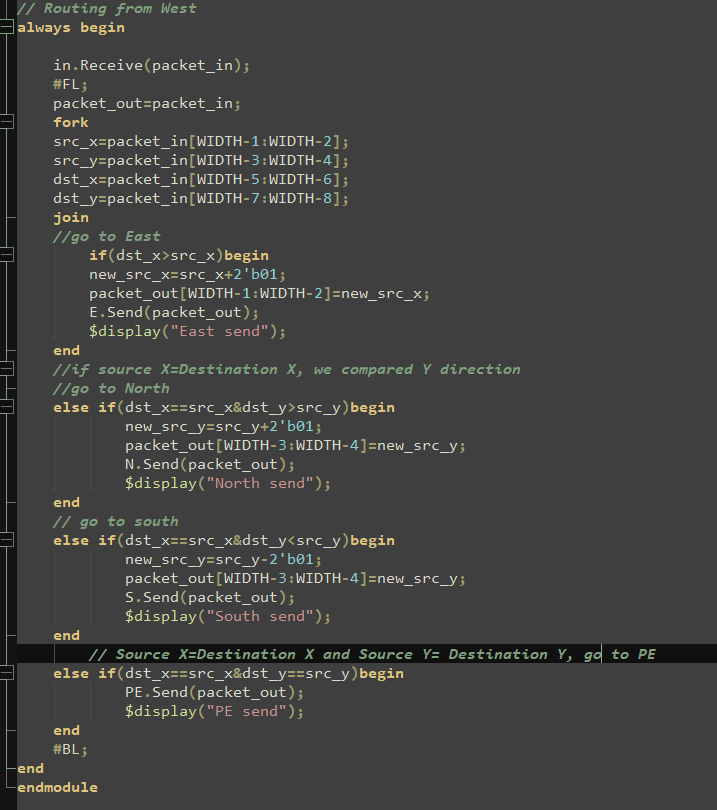


Fig. West Code 2

It is the same concept. First, the west algorithm block does not have western direction output. Also, I analyze the source and destination X and Y address. If the destination X address is larger than the source X address, the packet goes to East.

Later, if the source and destination X addresses are the same, I compared the Y addresses. If the destination Y address is larger or smaller than the source address, the packet goes to the North or South direction respectively. If the source and destination Y addresses are the same, the packet goes to the function blocks, such as PE or the threshold block.

After I determine the direction where the packet should go, I change the source address.

If the packet goes East, I add one to the source X address. If the packet goes North, I add one to the source Y address.If the packet goes South, I subtract one to the source Y address. If the packet goes PE, I do not change the source address. Before the packet leaves the PE block, I update the destination address in PE.

## **(D)South Routing Algorithm Block**

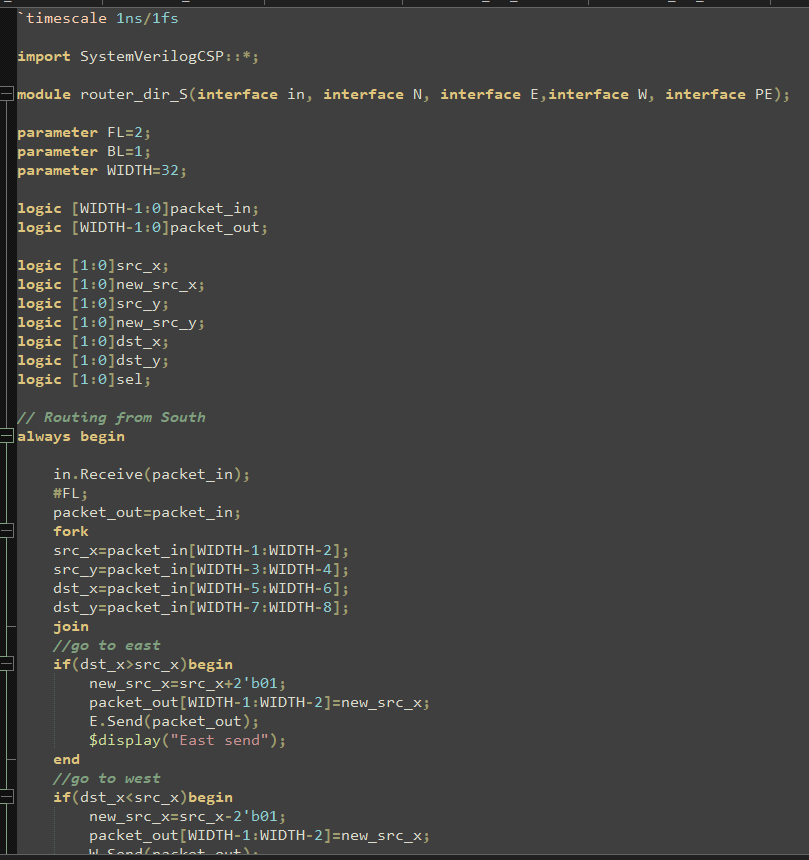


Fig. South Code 1

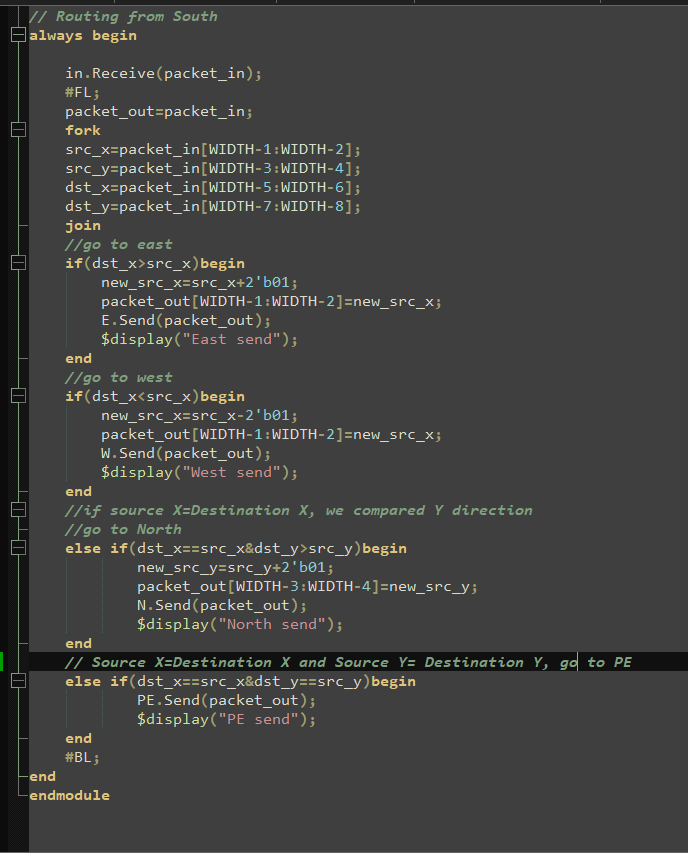


Fig. South Code 2

It is the same concept. First, the south algorithm block does not have southern direction output. Also, I analyze the source and destination X and Y address. If the destination X address is larger or smaller than the source X address, the packet goes to East and West respectively.

Later, if the source and destination X addresses are the same, I compared the Y addresses. If the destination Y address is larger than the source address, the packet goes to North direction. If the source and destination Y addresses are the same, the packet goes to the function blocks, such as PE or the threshold block.

After I determine the direction where the packet should go, I change the source address.

If the packet goes East, I add one to the source X address. If the packet goes West, I subtract one to the source X address. If the packet goes North, I add one to the source Y address.If the packet goes PE, I do not change the source address. Before the packet leaves the PE block, I update the destination address in PE.

## **(E)PE Routing Algorithm Block**

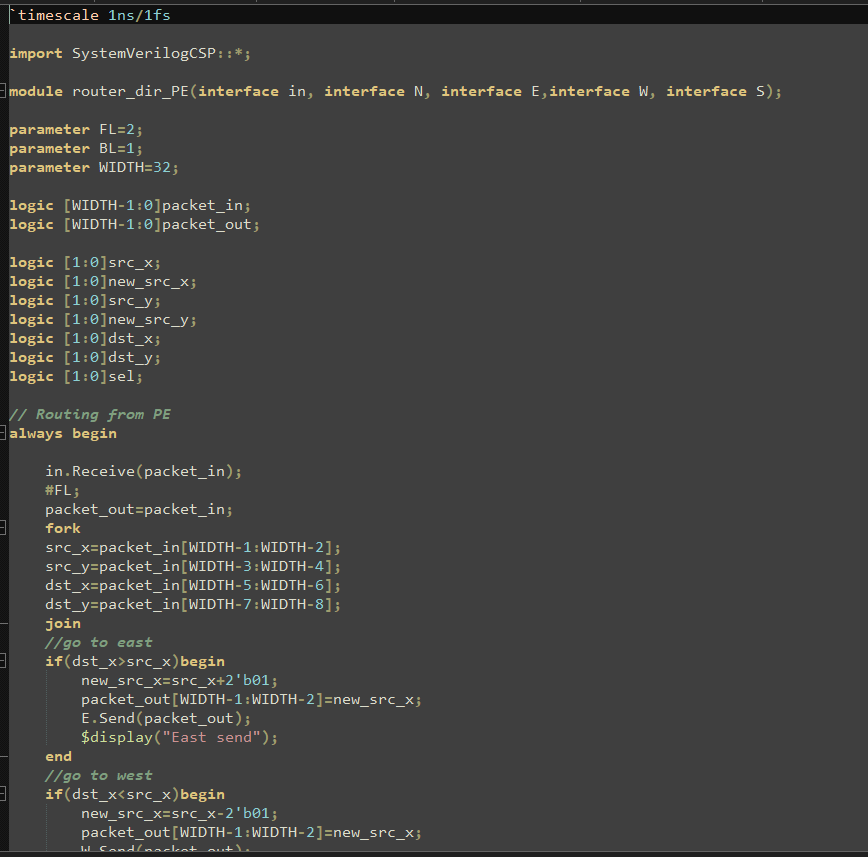


Fig. PE Code 1



Fig. PE Code 2

It is the same concept. First, the PE algorithm block does not have PE direction output. Also, I analyze the source and destination X and Y address. If the destination X address is larger or smaller than the source X address, the packet goes to East and West respectively.

Later, if the source and destination X addresses are the same, I compared the Y addresses. If the destination Y address is larger or smaller than the source address, the packet goes to the North or South direction.

After I determine the direction where the packet should go, I change the source address.

If the packet goes East, I add one to the source X address. If the packet goes West, I subtract one to the source X address. If the packet goes North, I add one to the source Y address. If the packet goes South, I subtract one to the source Y address.

# 

# **3. Arbiter**

For one output port, it will face 4 packets coming from different input directions. Hence, a question comes out: which packet should leave first? That’s why I built a 4-input arbiter.

## **(A) Two-input Arbiter**

First, I build a 2-input arbiter.

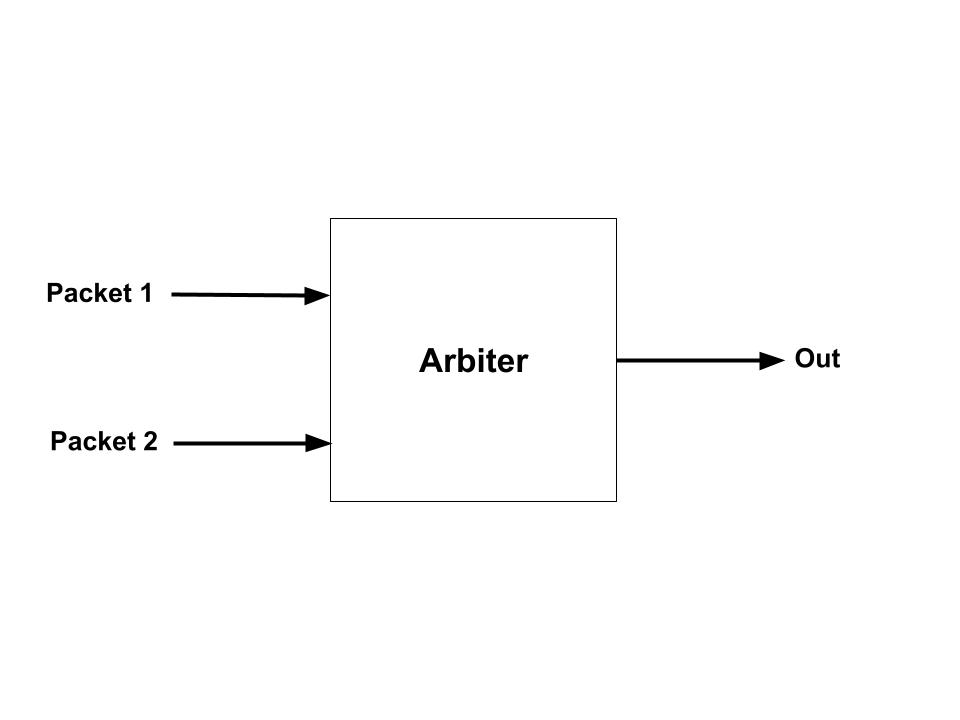


Fig. 2-input arbiter

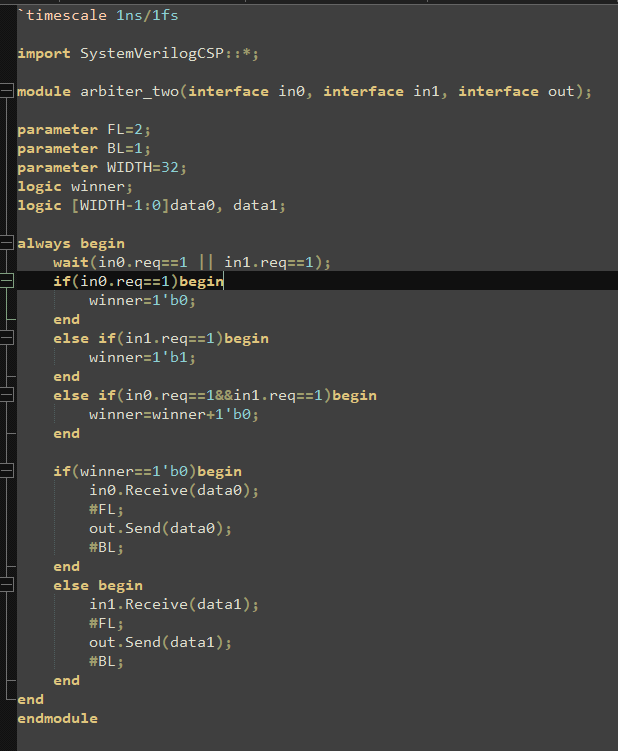


Fig. 2-input arbiter Code

I use the SVCSP variables “req” as a standard to tell which packet arrives the arbiter first or later.

First, the function waits for one of the requests. If in0.req=1, it means packet 1 arrives at the arbiter first and “winner” is 0. If in1.req=1, the packet 2 arrives at the arbiter first and “winner” is 1. If in0.req=in1.req=1, it means when two packets arrive at the arbiter at the same time, “winner” add 1.

If “winner” is 0, the arbiter receives the packet 1 and sends it to the output port. If “winner” is 1, the arbiter receives the packet 2 and sends it to the output port.

## **(B) Four-input Arbiter**

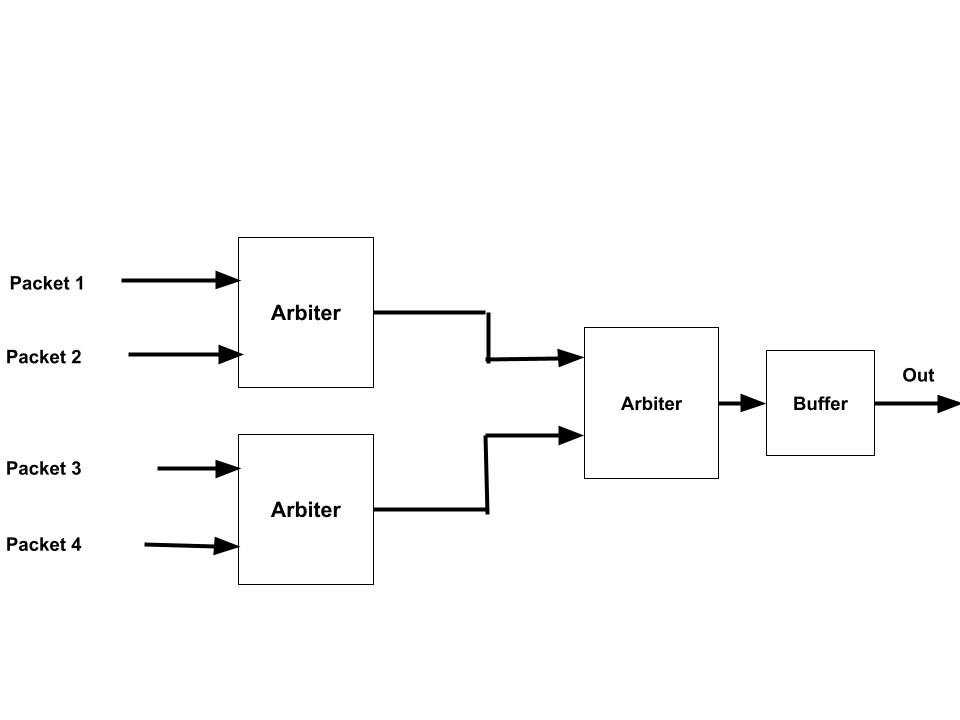


Fig. 4-input Arbiter

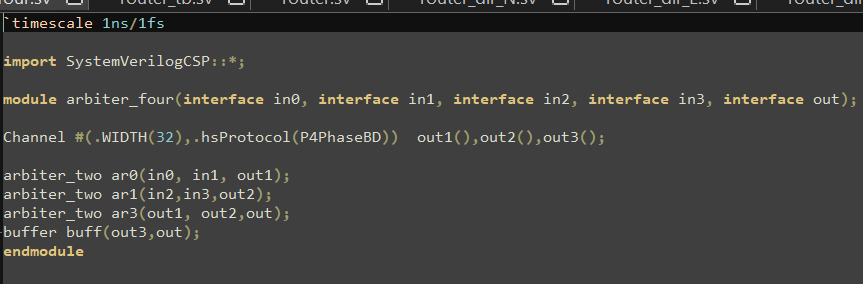


Fig. 4-input Arbiter Code

In this module, I use 3 2-input arbiters to build a 4-input arbiter. Also, I add an output full buffer to improve the throughput.

# **4. Router**

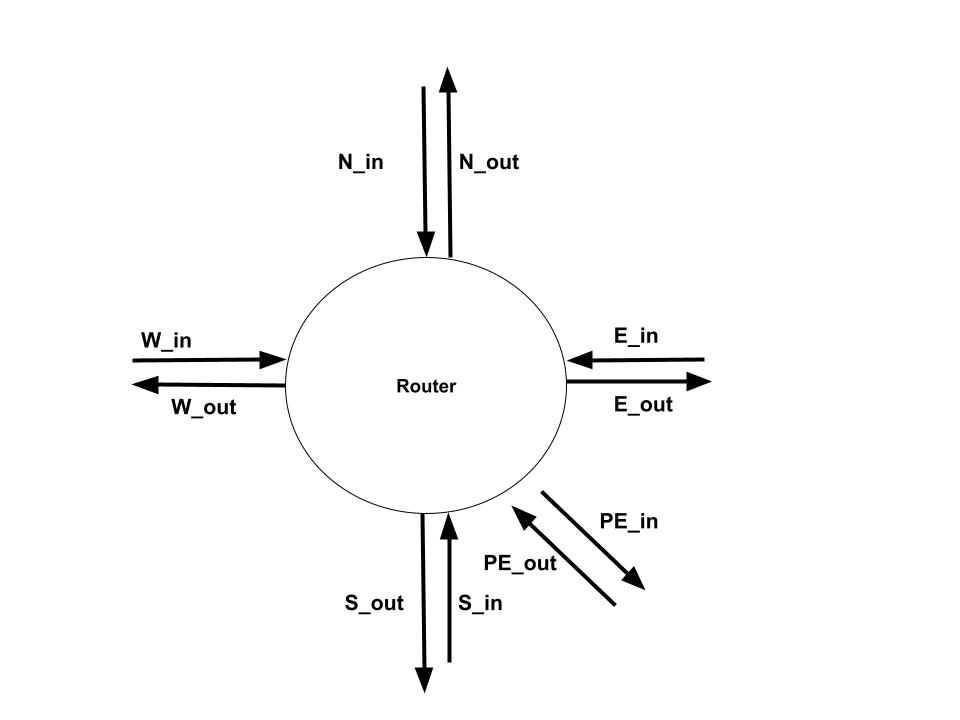


Fig. Router

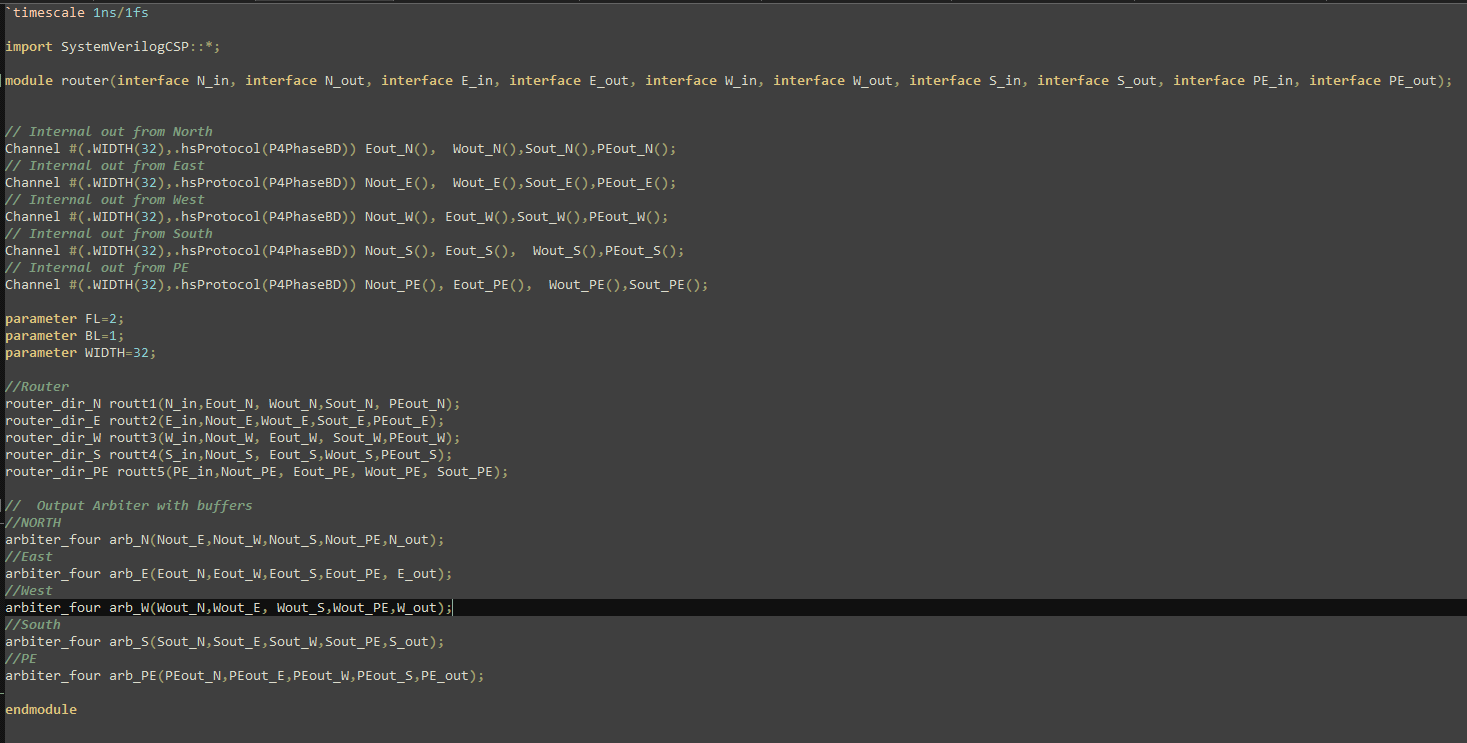


Fig. Router Code

Fig. Router Code

I connect submodules to make the router work. Let's look at the simulation.

## **(A)Simulation**

### **(1)PE Out**

In this case, I make all packets go to the PE output from different input ports.

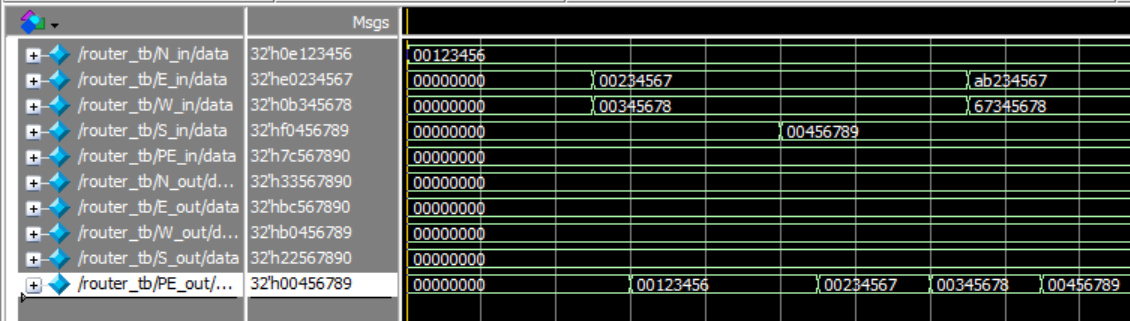


Fig. PE out Simulation

Let’s see the packet starting with 00. We can see that the packets from North, East, West, and South are sent to PE\_out successfully. Also, we can see that although 2 packets arrive at the same time, PE\_out is still correct.

### **(2)North Out**

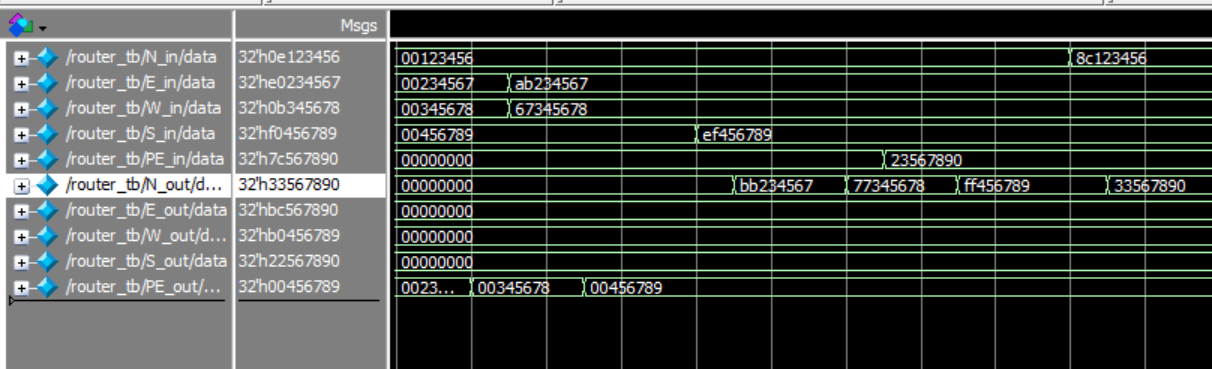


Fig. North out Simulation

Let’s look at the packets: **ab234567**, **67345678**, **ef456789**, and **23567890**. Because the packet bits are in heximal, we just need to consider the first 2 bits of the packet. They represent the source and destination address.

For address “ab” in hex, it means source address is (2,2), and destination address is (2,3) in decimal. Apparently, this packet should go to N\_out and it changes the source address successfully.

For address “67” in hex, it means source address is (1,2), and destination address is (1,3) in decimal. Apparently, this packet should go to N\_out and it changes the source address successfully.

For address “ef” in hex, it means source address is (3,2), and destination address is (3,3) in decimal. Apparently, this packet should go to N\_out and it changes the source address successfully.

For address “23” in hex, it means source address is (0,2), and destination address is (0,3) in decimal. Apparently, this packet should go to N\_out and it changes the source address successfully.

In a nutshell, all packets coming from different directions go to N\_out successfully. It shows every function for North output is correct under every submodule.

### **(3)East Out**

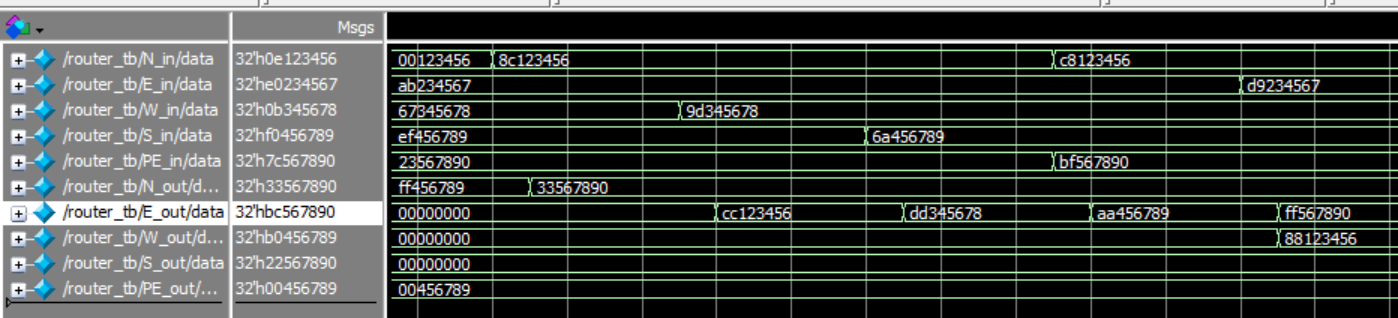


Fig. East Out Simulation

Let’s look at the packets: **8c123456**, **9d345678**, **6a456789**, and **bf567890**. Because the packet bits are in heximal, we just need to consider the first 2 bits of the packet. They represent the source and destination address.

For address “8c” in hex, it means source address is (2,0), and destination address is (3,0) in decimal. Apparently, this packet should go to E\_out and it changes the source address successfully.

For address “9d” in hex, it means source address is (2,1), and destination address is (3,1) in decimal. Apparently, this packet should go to E\_out and it changes the source address successfully.

For address “6a” in hex, it means source address is (1,2), and destination address is (2,2) in decimal. Apparently, this packet should go to E\_out and it changes the source address successfully.

For address “bf” in hex, it means source address is (2,3), and destination address is (3,3) in decimal. Apparently, this packet should go to E\_out and it changes the source address successfully.

In a nutshell, all packets coming from different directions go to E\_out successfully. It shows every function for East output is correct under every submodule.

### **(4)West Out**

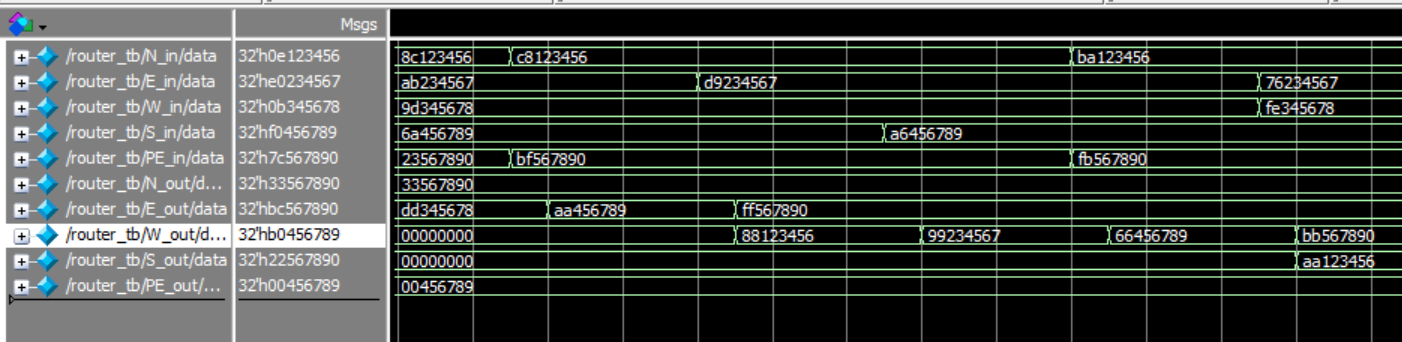


Fig. W out Simulation

Let’s look at the packets: **c8123456**, **d9234567**, **a6456789**, **fb567890**. Because the packet bits are in heximal, we just need to consider the first 2 bits of the packet. They represent the source and destination address.

For address “c8” in hex, it means source address is (3,0), and destination address is (2,0) in decimal. Apparently, this packet should go to W\_out and it changes the source address successfully.

For address “d9” in hex, it means source address is (3,1), and destination address is (2,1) in decimal. Apparently, this packet should go to W\_out and it changes the source address successfully.

For address “a6” in hex, it means source address is (2,2), and destination address is (1,2) in decimal. Apparently, this packet should go to W\_out and it changes the source address successfully.

For address “fb” in hex, it means source address is (3,3), and destination address is (3,2) in decimal. Apparently, this packet should go to W\_out and it changes the source address successfully.

In a nutshell, all packets coming from different directions go to W\_out successfully. It shows every function for West output is correct under every submodule.

### **(5)South Out**

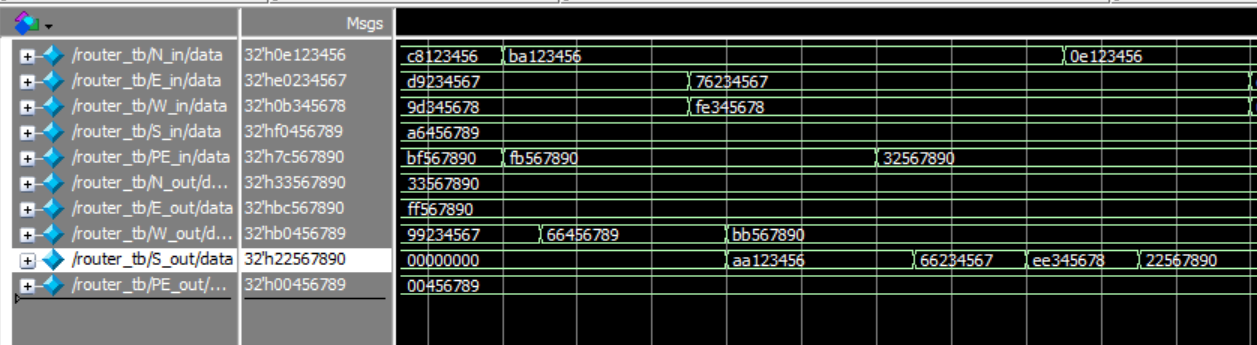


Fig. South Out Simulation

Let’s look at the packets: **ba123456**, **76234567**, **fe345678**, **32567890**. Because the packet bits are in heximal, we just need to consider the first 2 bits of the packet. They represent the source and destination address.

For address “ba” in hex, it means source address is (2,3), and destination address is (2,2) in decimal. Apparently, this packet should go to S\_out and it changes the source address successfully.

For address “76” in hex, it means source address is (1,3), and destination address is (1,2) in decimal. Apparently, this packet should go to S\_out and it changes the source address successfully.

For address “fe” in hex, it means source address is (3,3), and destination address is (3,2) in decimal. Apparently, this packet should go to S\_out and it changes the source address successfully.

For address “32” in hex, it means source address is (0,3), and destination address is (0,2) in decimal. Apparently, this packet should go to S\_out and it changes the source address successfully.

In a nutshell, all packets coming from different directions go to S\_out successfully. It shows every function for South output is correct under every submodule.

### 

### **(6)Random**

In this case, I just set random source and destination addresses to each input port.

Before we analyze the simulation, let's see the priority issue in the router.

For a router, when it needs to differentiate where the packet should go, it depends on the order of “if” or “else if” in the code. For example, this packet can go to East or North directions based on its destination address. But the algorithm “if block” for North is before the “if block” for East, the packet will go to North output first.

Based on the submodule showing in the previous pages, we can conclude the order in each submodule.

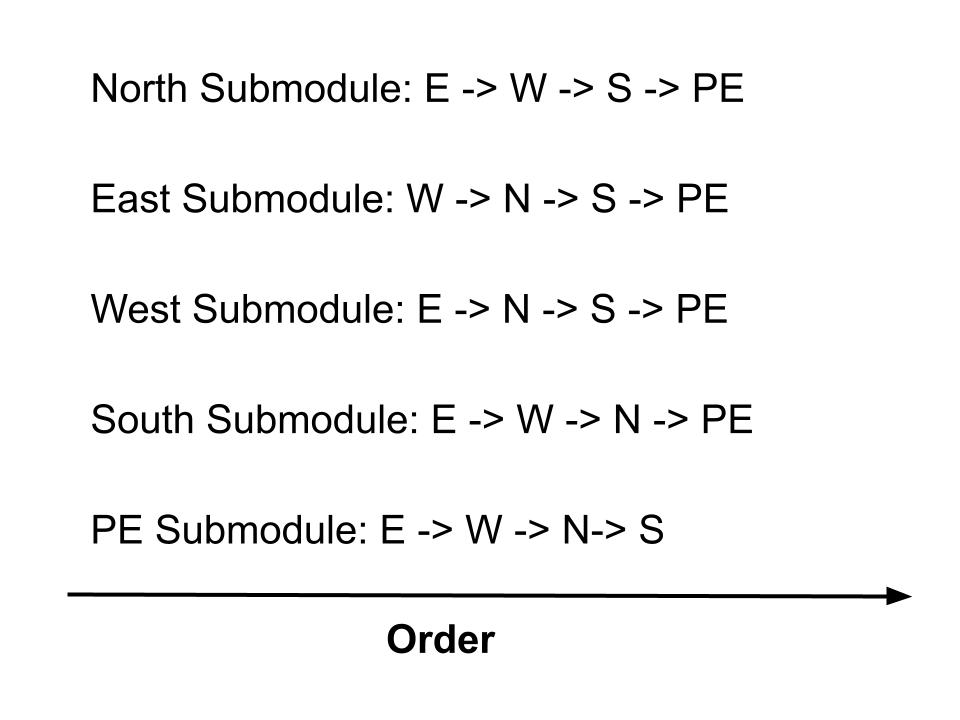


Fig. The order of Routing Algorithm

Now, let’s look at the waveform.

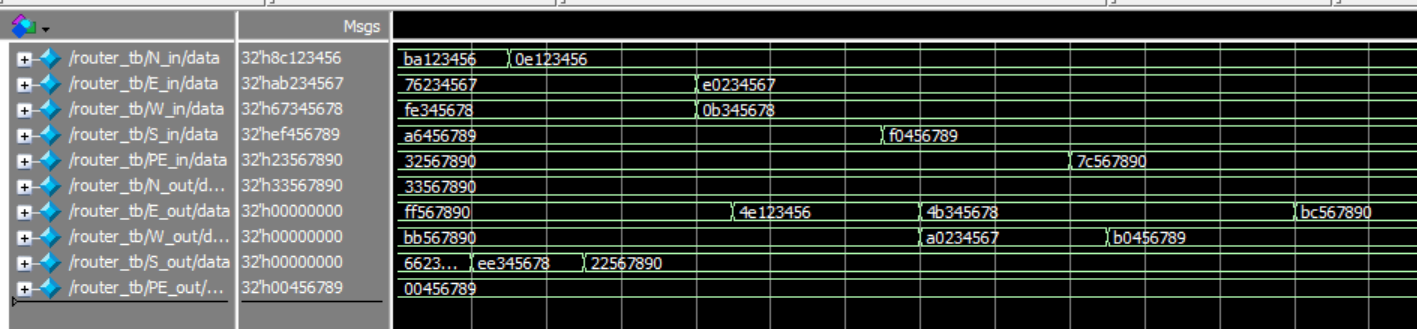


Fig. Random Case Simulation

Let’s look at the packets: **0e123456**, **e0234567**, **0b345678**, **f0456789**, **7c567890**. Because the packet bits are in heximal, we just need to consider the first 2 bits of the packet. They represent the source and destination address.

For address “0e” in hex, it means source address is (0,0) from N\_in, and destination address is (3,2) in decimal. Apparently, this packet cannot reach the destination in just one step, but we can check whether the packet follows the order rule and changes the source address. We can see that in the waveform, the packet goes to E\_out and changes the source address successfully.

For address “e0” in hex, it means source address is (3,2) from E\_in, and destination address is (0,0) in decimal. Apparently, this packet cannot reach the destination in just one step, but we can check whether the packet follows the order rule and changes the source address. We can see that in the waveform, the packet goes to W\_out and changes the source address successfully.

For address “0b” in hex, it means source address is (0,0)from W\_in, and destination address is (2,3) in decimal. Apparently, this packet cannot reach the destination in just one step, but we can check whether the packet follows the order rule and changes the source address. We can see that in the waveform, the packet goes to E\_out and changes the source address successfully.

For address “f0” in hex, it means source address is (3,3) from S\_in, and destination address is (0,0) in decimal. Apparently, this packet cannot reach the destination in just one step, but we can check whether the packet follows the order rule and changes the source address. We can see that in the waveform, the packet goes to W\_out and changes the source address successfully.

For address “7c” in hex, it means source address is (1,2) from PE\_in, and destination address is (3,0) in decimal. Apparently, this packet cannot reach the destination in just one step, but we can check whether the packet follows the order rule and changes the source address. We can see that in the waveform, the packet goes to E\_out and changes the source address successfully.