EE577B Fall 2020

Professor: Shahin Nazarian

Final Project: NoC Base Interconnect in Multi-Core Systems

Task 1: NoC Performance under Three different Mapping

Yaxin Deng, 4704064815

Jianqi Zhang, 1052509893

Hongxiang Gao, 8095639536

2020/10/16

Task1: NoC Performance under Three different Mapping

Modified Verilog Files:

Add the verilog files IF.v, PE.v, PE_basic.v, Node_add0.v, Node_add1.v, Node_add2.v, Node_add3.v, Node_and.v, Node_mul1.v, Node_mul2.v, Node_or.v, Node_xor.v. Create testbench tb_Node.v to test the functionality of Node module. Create testbench tb_phase_task1_mapping.v for the mapping 1 in task 1. Create testbench tb_phase_task1_mapping2.v for the mapping 2 in task 2. Create testbench tb_phase_task1_mapping3.v for the mapping 3 in task 3. Task1_mapping1_(func)_dst_table.hex files are the destination parameters for each node.

Note: Node_add0 includes PE_basic module, which acts as the entrance of the input flits. That's means in every mapping, we will map Node_add0 to the left node (only 8 function nodes in the requirement) for flits input.

Algorithm:

The Node module represents one of the modules of N0~N8. It consists of PE, function and IF three sub-modules.

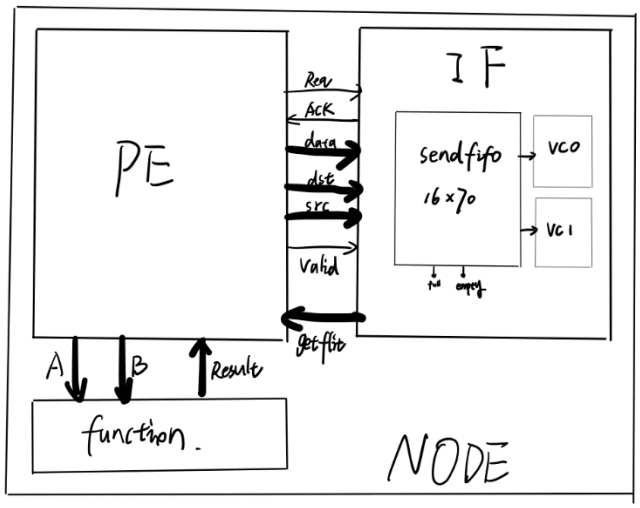

Figure 1.a General Schematic of NODE Module

To support the calculation of the requirement, we have several function modules, such as add_op, mul_op, and_op, or_op and xor_op. These modules are totally combinational logic. The inputs are DataA and DataB, which were provided by PE module, and the output is the Result also for PE module.

For PE module, it is responsible for handling the flits from mkNetwork and the communication with IF module. We design a state machine for PE module to implement the communication protocol as shown below.

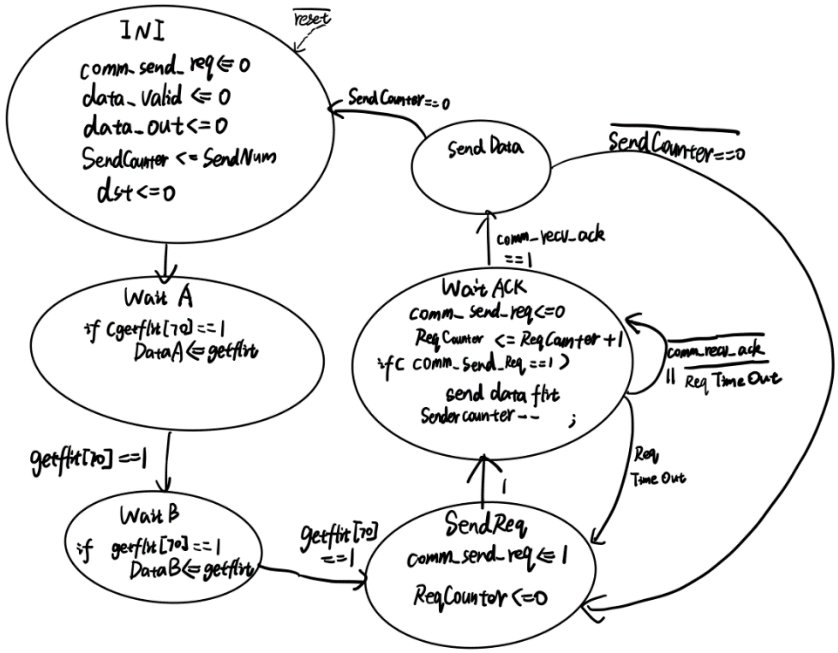

Figure 1.b State Machine Diagram of PE Module

After the initializing the variables and registers, the state machine come in to Wait A state to prepare receiving first data from the coming flit. When we get a valid flit, we register the flit data and transfer the state to Wait B state, which is same as the Wait A state. Then, in the SendReq state, PE sends request to IF and clear the ReqCounter. Then in Wait ACK state, the PE will wait the ACK signal from IF module. We have ReqCounter to record the clk numbers after we send the request. If the time out, the state machine will go back to SendReq state and send Request again. And if the PE get the request, it will send the data flit to IF module. Another counter is Send Counter, which is responsible for recording the number of copies we have sent (some PEs may send more than one copies of flit to other nodes). We also have an initial block to load its own destination table, which is determined by the mapping network and from the task1_mapping1_(func)_dst_table.hex file.

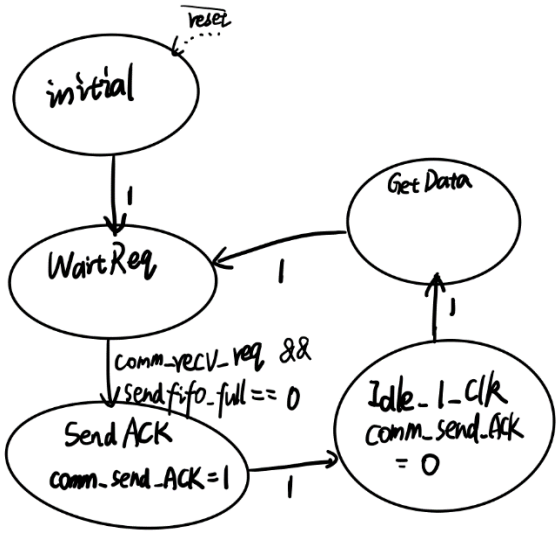

Figure 1.c State Machine Diagram of IF Module

For IF module, there is a inner sendfifo and two virtual channls in the module. When there is a valid flit from PE and the fifo is not full, it will write a flit into fifo. And when the fifo is not empty, one of the virtual channels will drain a data from fifo and send out. According to the credits of the virtual channels, if VC0's credits is not zero, we will choose VC0 as the default, and choose VC1 otherwise. Two counters for these two virtual channels are used to record the credits. Only when we don't get a credit and no sending out flit in this clock can we decrement the counter. If we get a credit and there is no sending out flit in this clock, we will increment the counter. That's our policy of credit management.

For the communication with PE, we also design a state machine to implement the communication protocol. After the initialization, the state machine will come in to Wait Req state in order to receiving the request sent from PE. After receiving Req and the inner sendfifo is not full, it will go to SendACK state to send back ACK to PE, and then IF will have an idle cycle. In the next clock, IF will get the data from PE.

Waveform Result:

First, we test the Node module. In this stage, we send in two flits and see whether we can get the calculation result from the output port. The waveform is shown below as Figure 1.d.

We can see after the node module get two flits it output the flits contains the right results three times. It means the functionality of node module is correct.

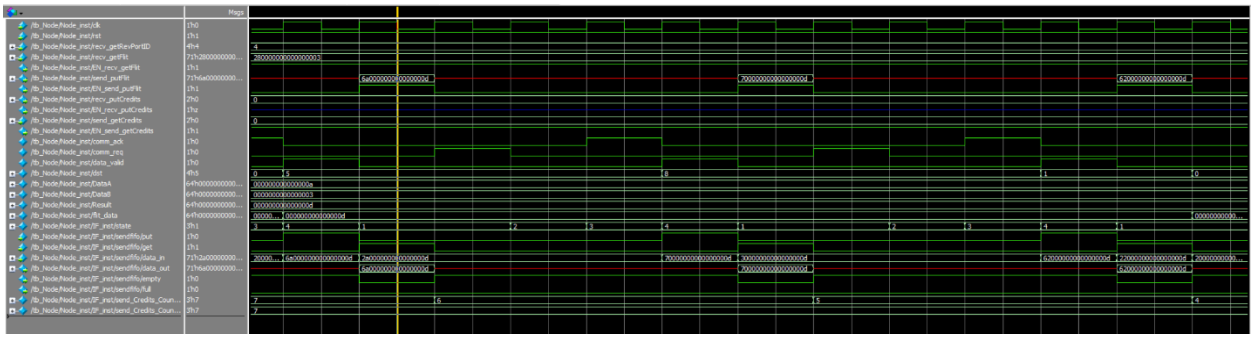

Figure 1.d Waveform of Node Module Functionality Simulation

Then we calculate the performance of different mappings:

Here, we let Node_add0 act as the entrance of the input flits. In every mapping, we will map Node_add0 to the left node (only 8 function nodes in the requirement) for flits input. And we generate 5 input flits in the testbench and send in to the mkNetwork by the port connected with Node_add0. Here, we use 10ns clock period.

(1) add1 maps to N4, or maps to N1, and maps to N3, mul2 maps to N5, xor maps to N8, add3 maps to N7, add2 maps to N6, mul1 maps to N0; The simulation waveform is shown as figure 1.e

We can see we start generating the input flit at 60000ps, and 6 input flits are generated in serial and sent to their first stage. After a while, we see the add3 node and see the calculation result 0x85 at **650000ps**.

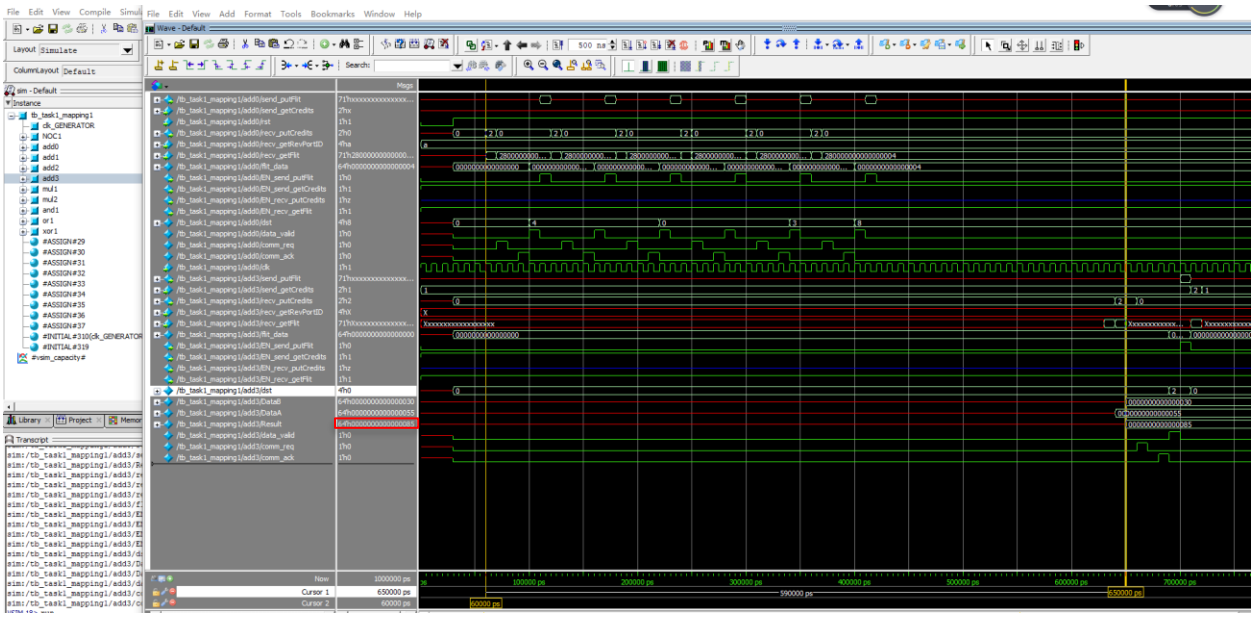

Figure 1.e Waveform of Mapping 1

(2) add1 maps to N2, or maps to N8, and maps to N0, mul2 maps to N3, xor maps to N6, add3 maps to N1, add2 maps to N4, mul1 maps to N5;

We can see we start generating the input flit at 60000ps, and 6 input flits are generated in serial and sent to their first stage. After a while, we see the add3 node and see the calculation result 0x85 at **660000ps**.

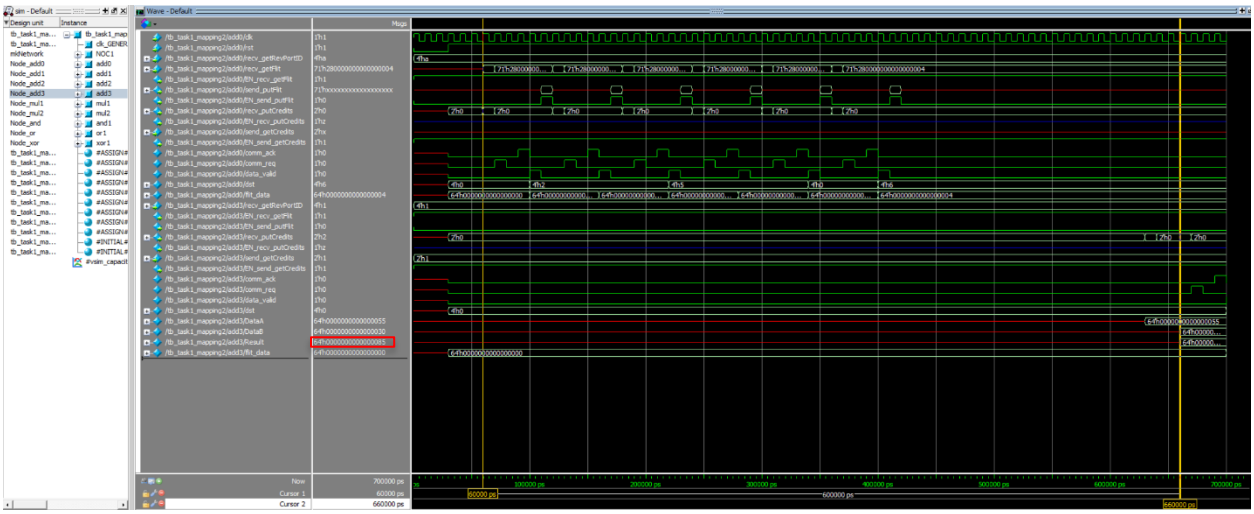

Figure 1.f Waveform of Mapping 2

(3) add1 maps to N1, or maps to N2, and maps to N3, mul2 maps to N4, xor maps to N5, add3 maps to N6, add2 maps to N7, mul1 maps to N8;

We can see we start generating the input flit at 60000ps, and 6 input flits are generated in serial and sent to their first stage. After a while, we see the add3 node and see the calculation result 0x85 at **670000ps**.

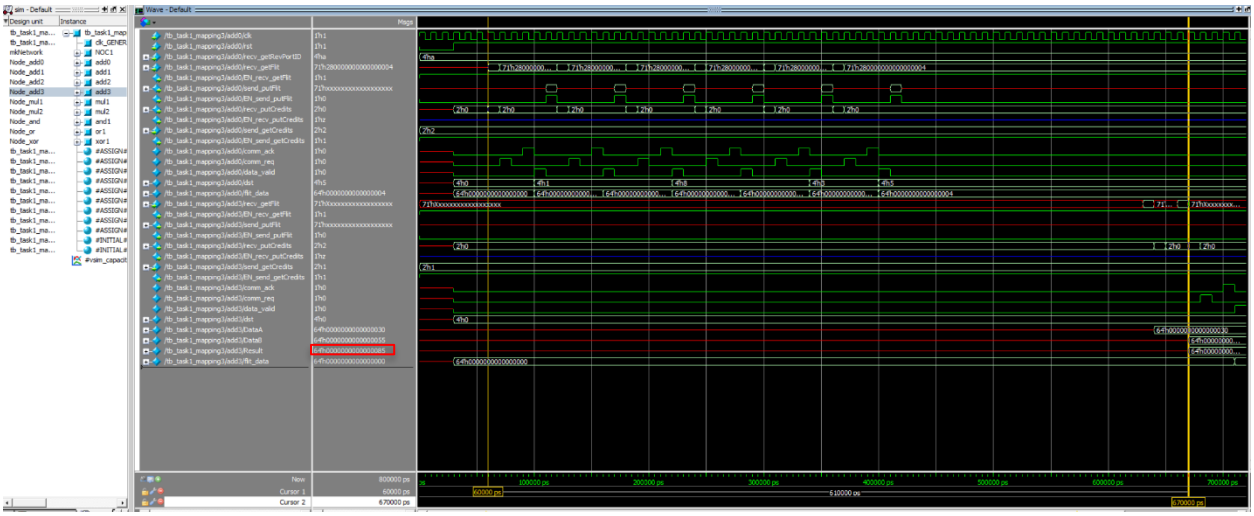

Figure 1.g Waveform of Mapping 3

Now, for three mappings, we start feed in the 6 input flits at the same time, and we can see that the first mapping has the best performance: 650000ps. Mapping 2 and mapping 3 are slower a little bit.

The performance is different. Because we assign the nodes in different mapping and the flits will experience different paths among these nodes. Take mapping1 for example, add1 maps to N4, and the next stage of add1 are AND, OR and MUL1, which are only 1 unit away from the add1. So, the intermediate flits can reach the next stage easily. While in the mapping2 and mapping3, the transmission paths are different. For every intermediate flit in different mapping, they may experience different length routing, so that the total execution time varies.

Task2: Router 2 Modification:

Modified Verilog Files:

For achieving the Task2, a new RouterCore module "R2_mkRouter" (modified under "R2_mkRouter.v") will be used to replace the "mkRouterCore" module inside "mkNetwork.v" line 1811. This new "R2_mkRouter" module includes three additional routers R21, R22, and R23 to increase the performance on original R2. A new "Round_Robin_4.v " file is added into the folder as a submodule supporting R2_mkRouter arbiter.

Algorithm:

Following the assignment request in Phase1 task2, I create three extra routers R21,R22, and R23 by using the provided mkRouterCore module. These three additional routers work as substages for R2, and they have equivalent functions as R2 to be able to communicate with N2, R1, R5. With these substages, we are able to speed up the performance for R2 by processing multiple input flits during the same clock. For example, if there are three flits inside network need to passed by R2 with different destination router (e.g. Flit1:R1->R2->R5, Flit2: N2->R2->R1, Flit3: R5->R2->N2), instead of waiting 3 clock to output, these three flits can all output to the next router during the same clock. However, due to the limited receiving port on the receive router, if there are two flit have the same destination is outputting during the same clock (e.g. Flit4: R5->R2->R1, Flit5: N2->R2->R1), then the arbiter will decide the output order and output the two results on the continuous two clock.

Arbiter Design Details:

(1) Output port arbiter: The output arbiter for both flit and credits are formatted by a Round_Robin structure. The 4-bit wide input request_vector for the round_robin is built by flits' valid bit stored inside each router, and all these valid bits follow {R2,R21,R22,R23} order for the request_vector. Depending on priority, Round_Robin will return a grant_vector back, and the output flit Enable signal can be formatted by the flit_valid_bit && grant_vector. As the same example above, if both of my Flit4(store in R2) and Flit5(store in R21) require to go to R1, then the Flit4[70] and Flit5[70] are both valid and outputting "1". Then the request_vector in this clock cycle would be 1100. Following the priority R2 will be output first, so the returning grant_vector 1000 will be received, and only R2 enable signal is logic high with Flit4 output. On the following clock, the Flit5 can be received.

(2) Input port arbiter: Each input port will have a rotational router select from R2 to R23, it depend on the usage of the stage, if the router is already being used, then the router selection counter will increase, and the flits will be input into different router on the next clock. The input port will never be bought, and each port starts with a different starting router after reset.

Waveform Result:

For easier to see the improved performance for flits and credit transfer, I introduce two testbenches for task 2 including “tb_R2_Router.v” and “tb_phase1_task2.v”. “tb_R2_Router.v” is testing the based on the ports of the router module, and “tb_phase1_task2.v” is testing for putting the new router into the network.

(1) testbench for R2_Router module (including R2,R21,R22,R23 inside): Test with “tb_R2_Router.v” , a communication ports shown in Figure 2 below:

As the I/O pin given in the provided diagram, port 0, 2, 3 can receive input flits, and port 0, 1, 4 are used to output flits. So in the output waveform, I tested the flit passing following input port -> output port: port 2 -> port 1, port 3 -> port 4, and port 0 -> port 0. As explained above on the *Algorithm* and *Arbiter Design Detail* section, due to the arbiter selection, these three flits will be stored into different routers and perform parallel processing. As shown in the following Figures 2.a and Figure 2.b in the red rectangle area, after reset, the new Router module can output flits to different destination ports during the same clock. This result proved that the performance by these additional routers. The reason for input is 74 bit and output 71 bit is the auto mapping this router gives, the input_flit[73:3] is the full flit sent from the previous router.

Similar to the flit transfer, the sending port 0,2,3 will receive an immediate credit ‘2’b10’ got transferred back from the R2 based on the input flit information, while flits get transferred out from the router as shown in Figure 2.b section. This could keep the credit-based communication for the system .

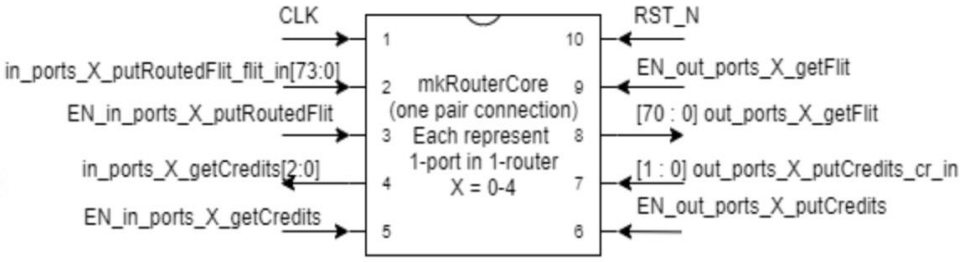

Figure 2: Communication Port for Module R2_Router

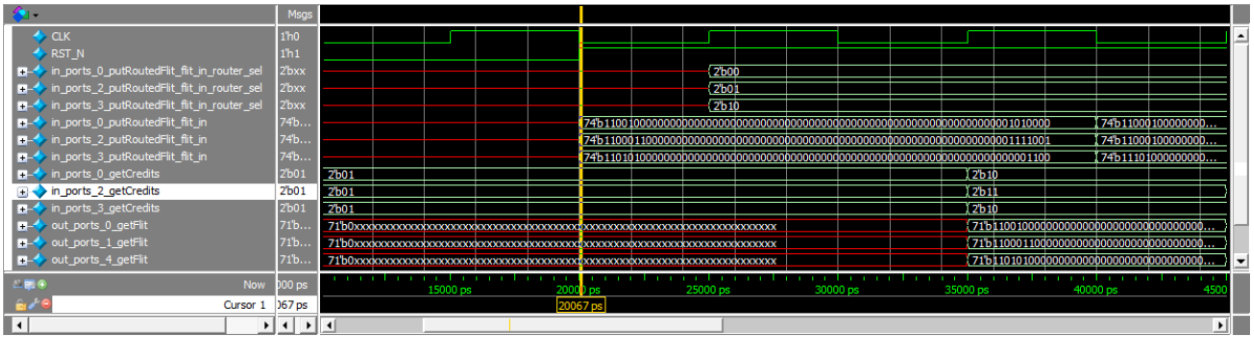

Figure 2.a: Overview for Output Flits for Different Destination Ports

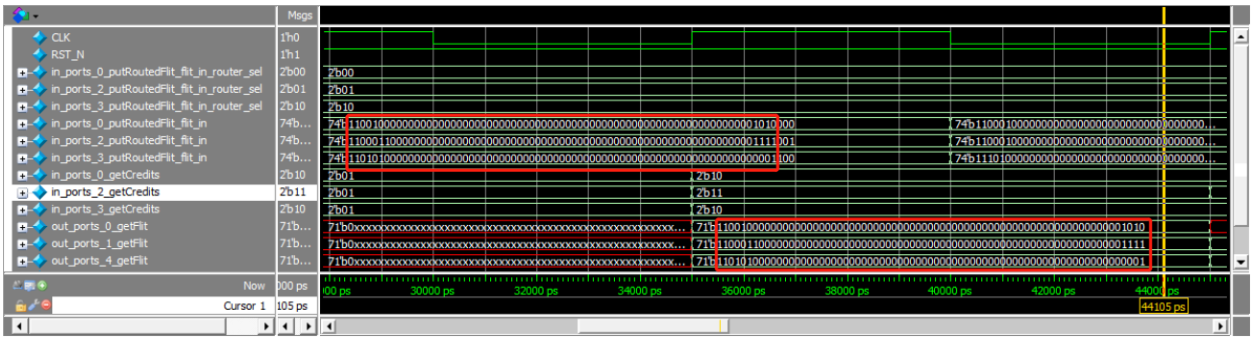

Figure 2.b: Detail Data check for Zoom in Figure 2.a

For testing the arbiter, we can see that if there is a group of data require output to the same address, then the arbiter will read out the data one by one to avoid traffic on the output port as shown below in Figure 2.c.

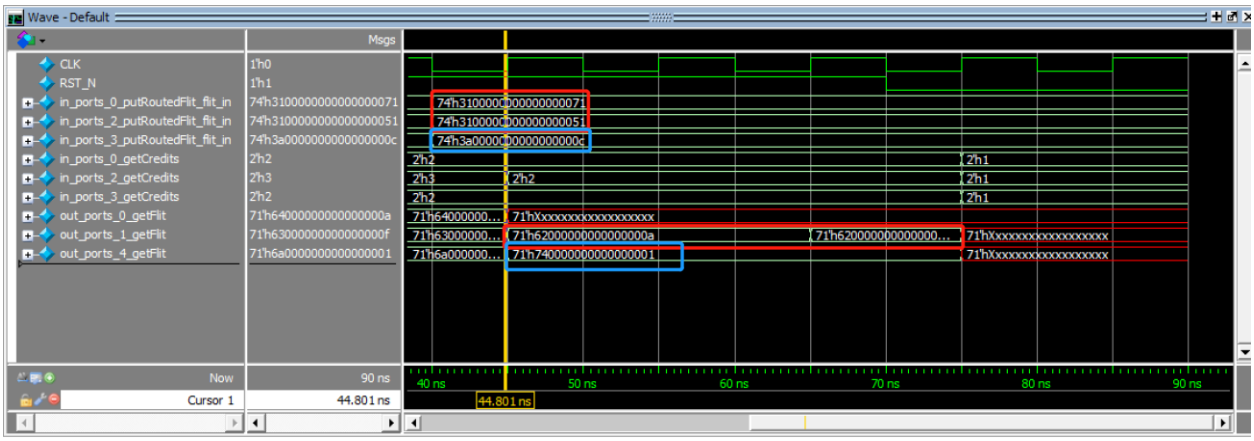

Figure 2.c: Output Traffic Control

(2)Test for entire system with multiple routers R2 inside the NoC network flow:

By combining the PE, IF and new mkNetwork, we write flits sending R1->R2 and R5->R2, these two movements will both be received by the router 2 simultaneously as shown in Figure 2.d below. Router can receive these two flits and start processing simultaneously as what we expected. This design will improve the performance for the result.

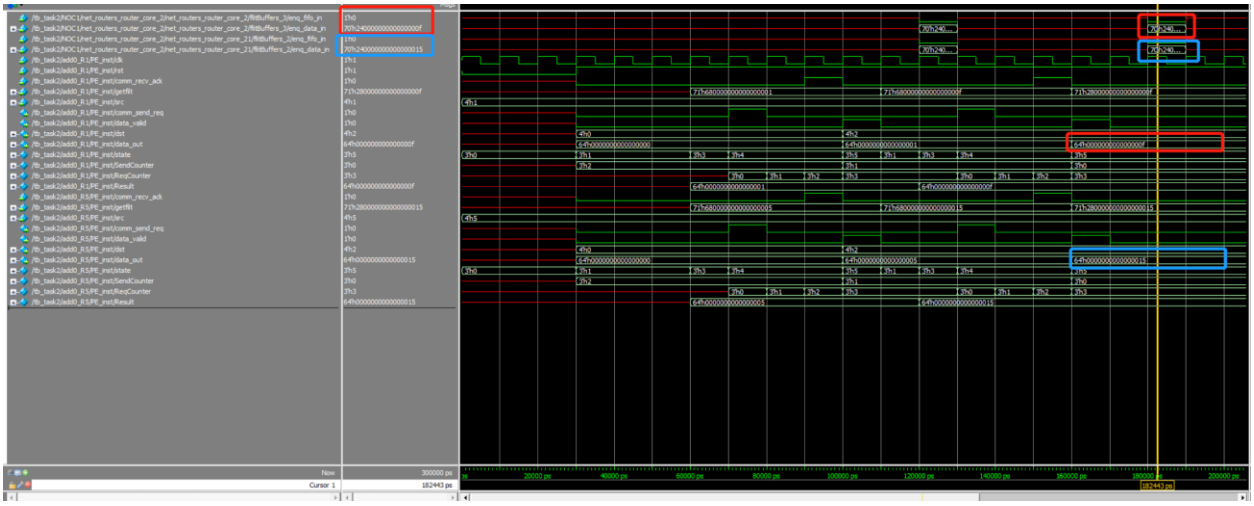

Figure 2.d: Output Waveform for Testing New Router in System