EE552 Final Project

Final Report

4D Hypercube Network on Chip

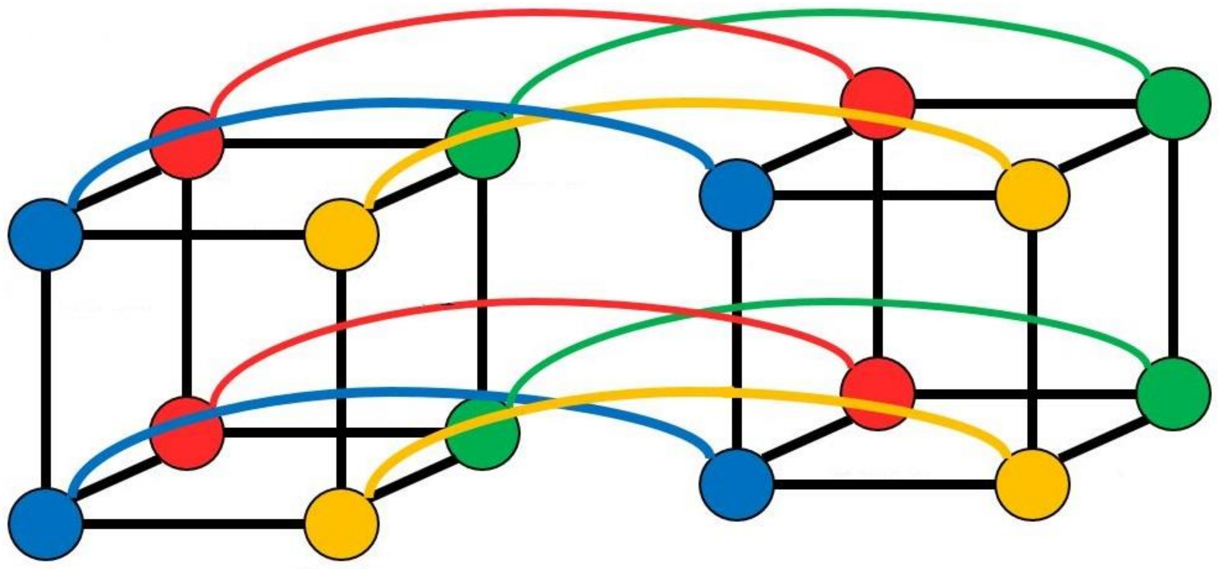
Hyeong An
Richard Chan
Danny Gil

Table of Contents

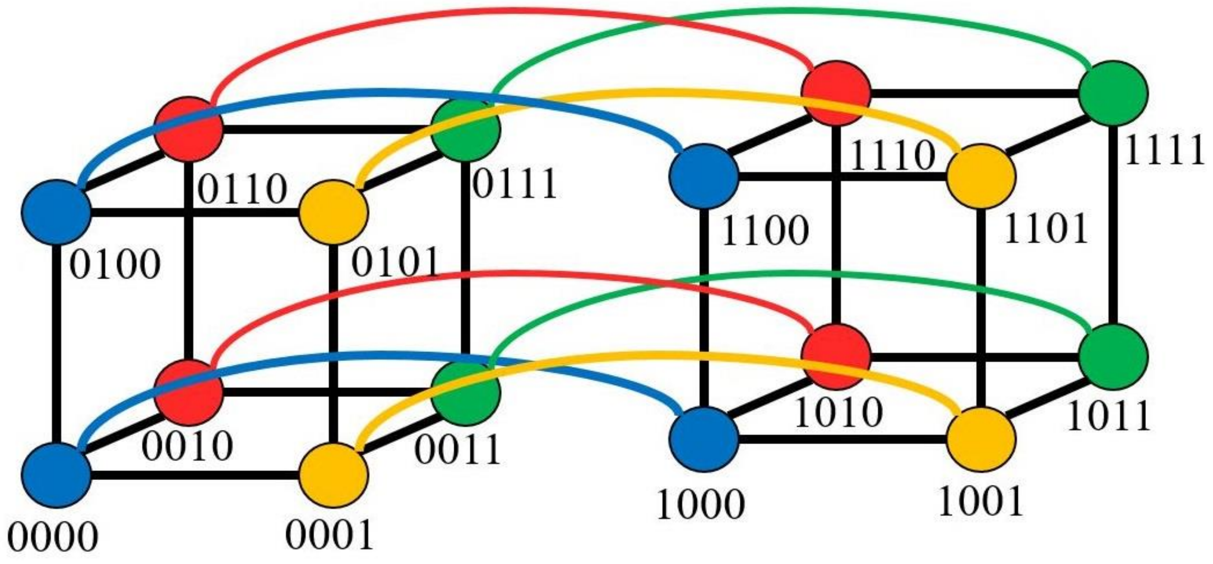
1.0 Introduction

Routing dedicated wires to connect top-level modules increases area and decreases performance. Network-On-Chips (NoC) is a good way to avoid these problems. NoCs do this by sharing wire connections between nodes. These nodes have built-in routers which use specific protocols to determine the most efficient way to transmit data from point a to point b. Error detection and correction techniques are also used to ensure the receiving end obtains the correct data. By sharing path resources together with an efficient routing algorithm NoCs make a better option that routing dedicated wires between modules.

2.0 Overall Structure

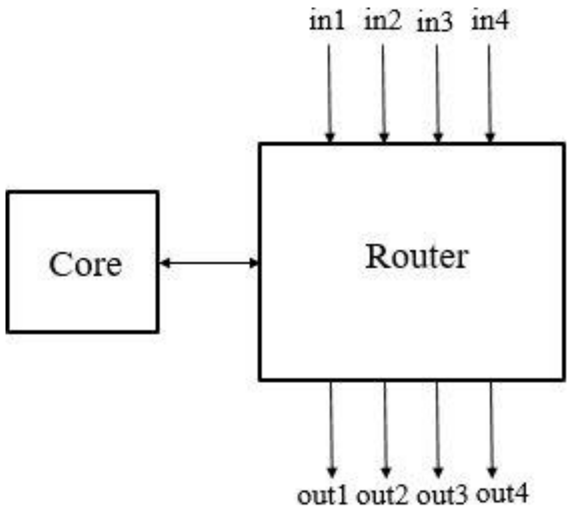


The network topology chosen was a 4D hypercube. This topology was chosen because it has a shorter worst case travel distance of 4 hops between nodes compared to the baseline topology which has a worse case travel distance of 8 hops. Each node implements a router and a core. The testbench controls the sending and receiving of each node.



The nodes were given on address based on gray code. The address of all adjacent nodes to any given node differs by only 1-bit.

2.1 Node Structure

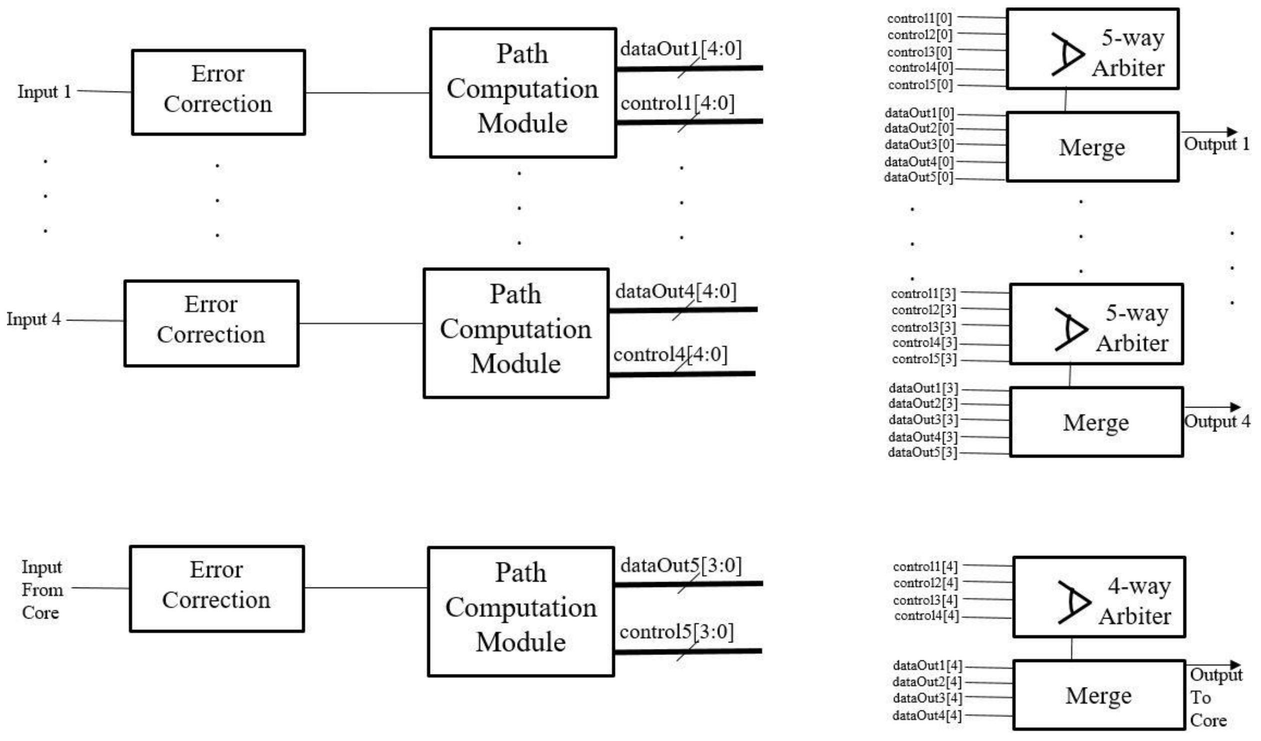


2.2 Core Structure

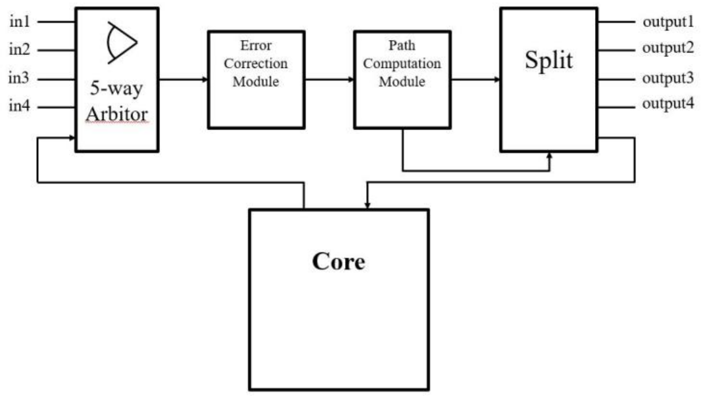
The function of the core is to encode the 8bit data (4bit data + 4bit address) received from data generator and decode the 11bit data received from the router after reaching its destination node. The encoder adds parity bits (3bit) to the data sending it to the router and the decoder removes these parity bits before sending the data to data bucket.

2.3 Router Structure

The router structure we used receives data packets from the adjacent nodes and the current core. The data packets are checked for any errors and if any exist they are corrected by our error correction module. The data packets are then passed to the path computation module which splits the data packets into the data being transmitted and a control acknowledgment data. The 5-way arbiter receives the control data and sends the control value of the winning data to merge so it can output the corresponding data that won the arbitration.
Note that we have 4 5-way arbiters, and 5 input merge modules for outputting to neighboring nodes and we have one 4-way arbiter and one 4 input merge for outputting to the current core.



Below is the original routing structure we designed for part 1. This structure serialized the computation of the input data which made it slow because of contention at input. We used this structure because we wanted to save area. we redesigned our structure to the one above to increase the performance of our design. Our new design allowed the input data to be computed in parallel. This gave our final design better performance for the cost of additional area.



2.3.1 Path Computation Module

Path computation module dynamically calculates the path a received data will take. we XOR'd the current address with the address of the destination node. Then we go from LSB to MSB to find the first 1's bit position to determine the path. Path computation module consists of a bit slicer to separate the data and the address and feeds that address data to XOR module to compute the result. After that, we find the bit position of the first value "1" it sees then it outputs the data and the control signal which will be connected to corresponding arbiter modules and merge modules.

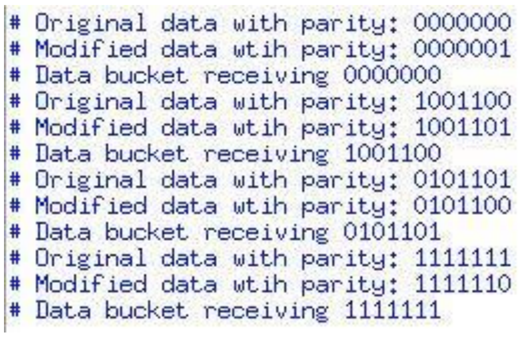
2.3.2 Arbiter

In our node structure, we used 4 input arbiter and 5 input arbiter. They are used to select control signals for our merges. In our arbiter module, we created a 2 input arbiter and we cascade 2 input arbiters to get 4 and 5 input arbiters.

]

2.3.3 Error Correcting Module

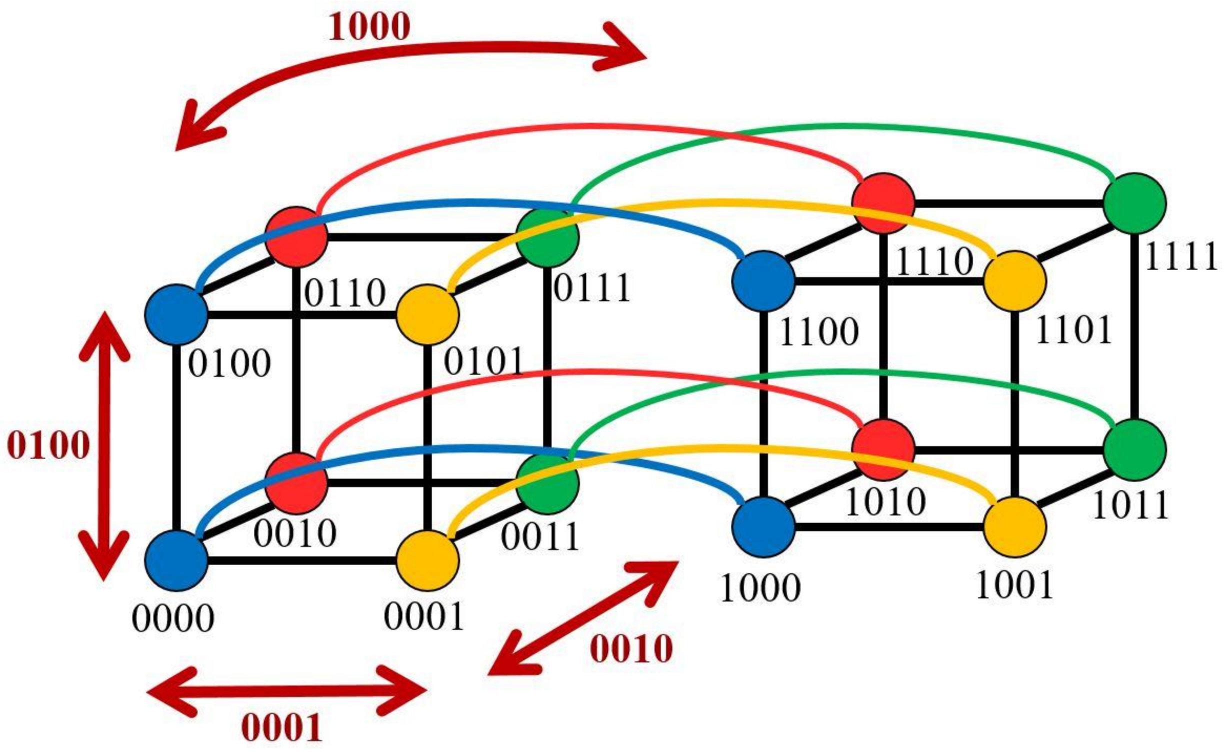
Hamming code was used to implement our error correction modules. Error correction modules perform parity-bit checks on data bits and correct single bit error. Parity bits are used to indicate the number of ones in the data prior to transmission. Parity bits checking allow us to detect data corruption during transmission and, with three parity bits, our module can correct any single bit data corruption. All data received by routers are first passed through error correction module before entering path computation module to ensure error-free data that will be used to calculate path. When data first generated by data generators in the test bench, they are passed into our cores where the parity bits are generated before data are sent to routers.



```
# Original data with parity: 0000000
# Modified data wtih parity: 0000001
# Data bucket receiving 0000000
# Original data with parity: 1001100
# Modified data wtih parity: 1001101
# Data bucket receiving 1001100
# Original data with parity: 0101101
# Modified data wtih parity: 0101100
# Data bucket receiving 0101101
# Original data with parity: 1111111
# Modified data wtih parity: 1111110
# Data bucket receiving 1111111
```

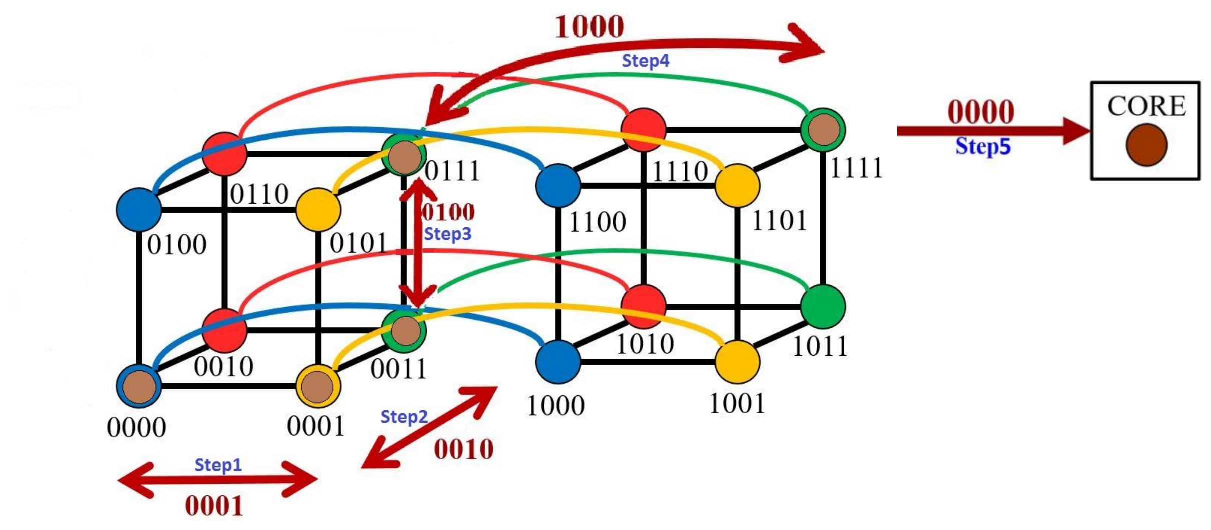
The edu test bench shows correct results

3.0 Routing Algorithm



The paths a token can take was characterised as follows:
0001 - horizontal move
0010 - forward/back move
0100 - vertical move
1000 - 4th dimensional move
To compute the path a token will take, we XOR'd the address the token was currently in with the address of the destination node. The LSB of the result where the data was 1 indicated the move direction of token. Example provided on next page.

For example, if the current location of the token is 0000 and its destination address is 1111 the routing was as follows:



Current Address | Destination Address

Step 1:

$$0000\ \mathbf{XOR}\ 1111 = 111\underset{\substack{\downarrow \\ 0001 \\ \leftrightarrow}}{\boxed{1}}$$

Step 2:

$$0001\ \mathbf{XOR}\ 1111 = 11\underset{\substack{\downarrow \\ 0010 \\ \swarrow\!\nearrow}}{\boxed{1}}0$$

Step 3:

$$0011\ \mathbf{XOR}\ 1111 = 1\underset{\substack{\downarrow \\ 0100\ \updownarrow}}{\boxed{1}}00$$

Step 4:

$$0111\ \mathbf{XOR}\ 1111 = \underset{\substack{\downarrow \\ 1000 \\ \curvearrowleft\!\curvearrowright}}{\boxed{1}}000$$

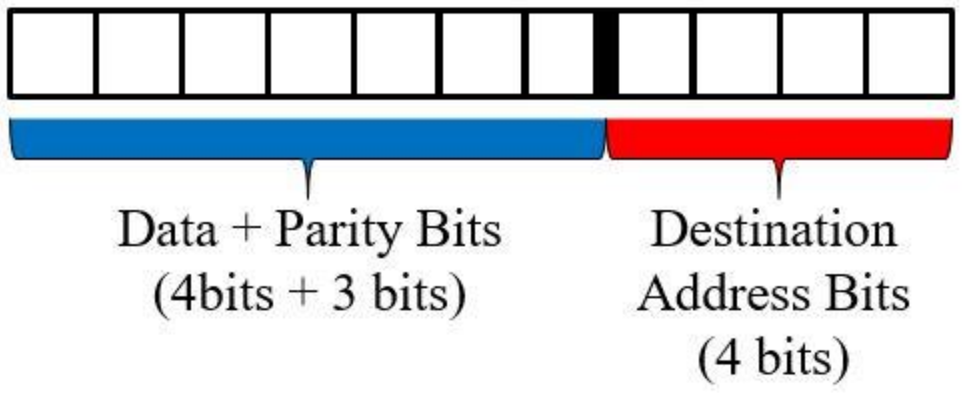
Step 5:

$$1111\ \mathbf{XOR}\ 1111 = \underbrace{0000}_{\substack{\text{Address} \\ \text{Match}}}$$

Core

4.0 Error Detection and Correction

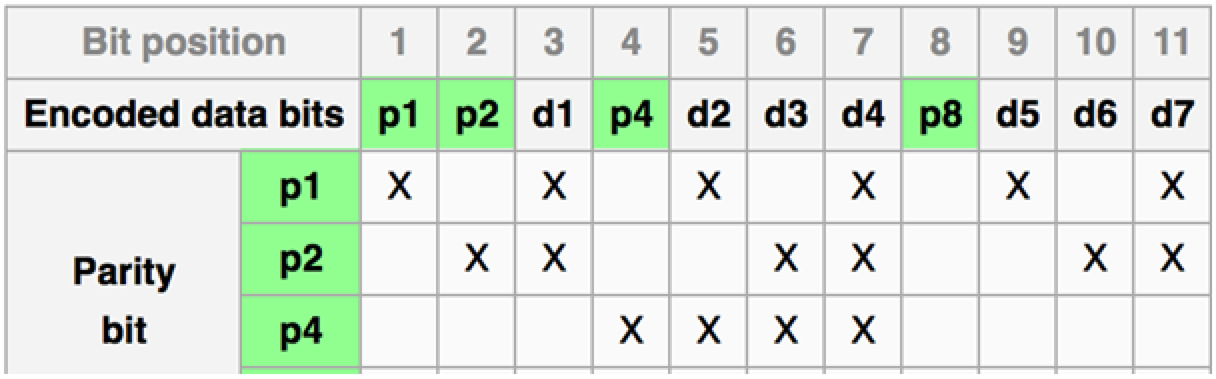
For error detection and correction, we used Hamming code. Our data packet was 11-bits which included 4 data bits + 3 parity bits + 4 address bits as shown below:



4.1 Hamming Code

Hamming code is capable of not only detecting errors but also correcting them. This is the reason for us choosing this methodology. Hamming code is able to detect errors that are up to 2-bits in length or correct single bit errors.

4.2 Encoding/Decoding



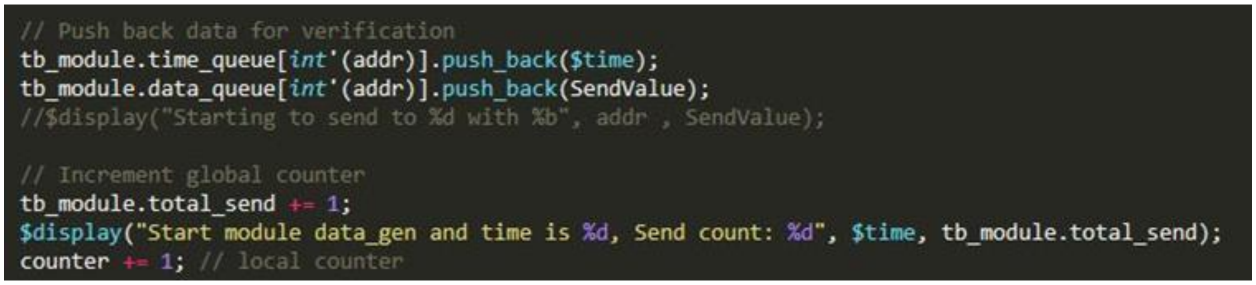
Bit position		1	2	3	4	5	6	7	8	9	10	11
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7
Parity bit	p1	X		X		X		X		X		X
	p2		X	X			X	X			X	X
	p4				X	X	X	X				

5.0 SIMULATIONS

5.1 Test bench structure

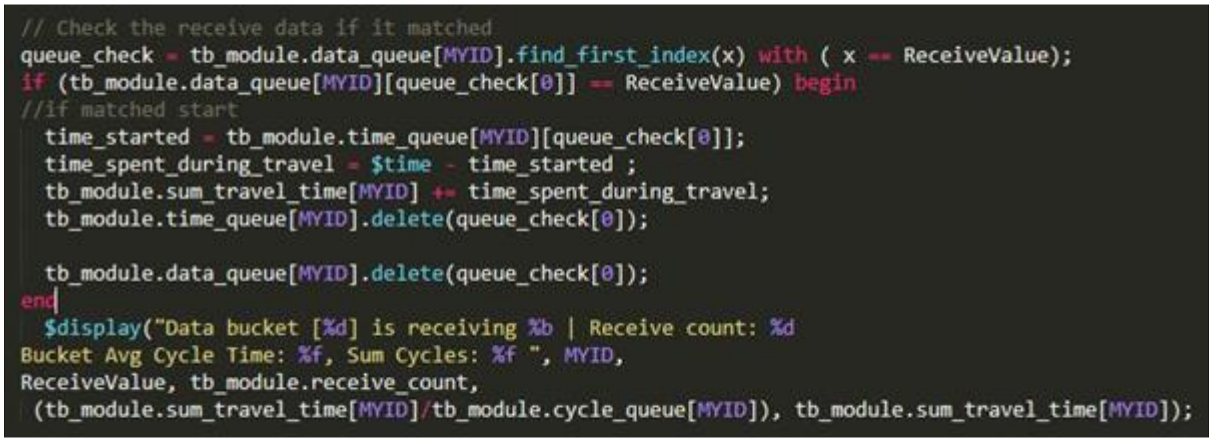
Our test bench is made up of 16 data generators and 16 data buckets. We use global data for final verification. The data generators push data into the destination's data queue. They also push timestamps into the destination's time queue. The data buckets check if the data reach the correct destination and retrieve the corresponding timestamps and performance data. They also push the result into the travel time array and cycle array.

Data Generator: Push data and time stamp



```
// Push back data for verification
tb_module.time_queue[int'(addr)].push_back($time);
tb_module.data_queue[int'(addr)].push_back(SendValue);
//$display("Starting to send to %d with %b", addr , SendValue);

// Increment global counter
tb_module.total_send += 1;
$display("Start module data_gen and time is %d, Send count: %d", $time, tb_module.total_send);
counter += 1; // local counter
```

Data Bucket: Verify data and push performance data into global arrays.

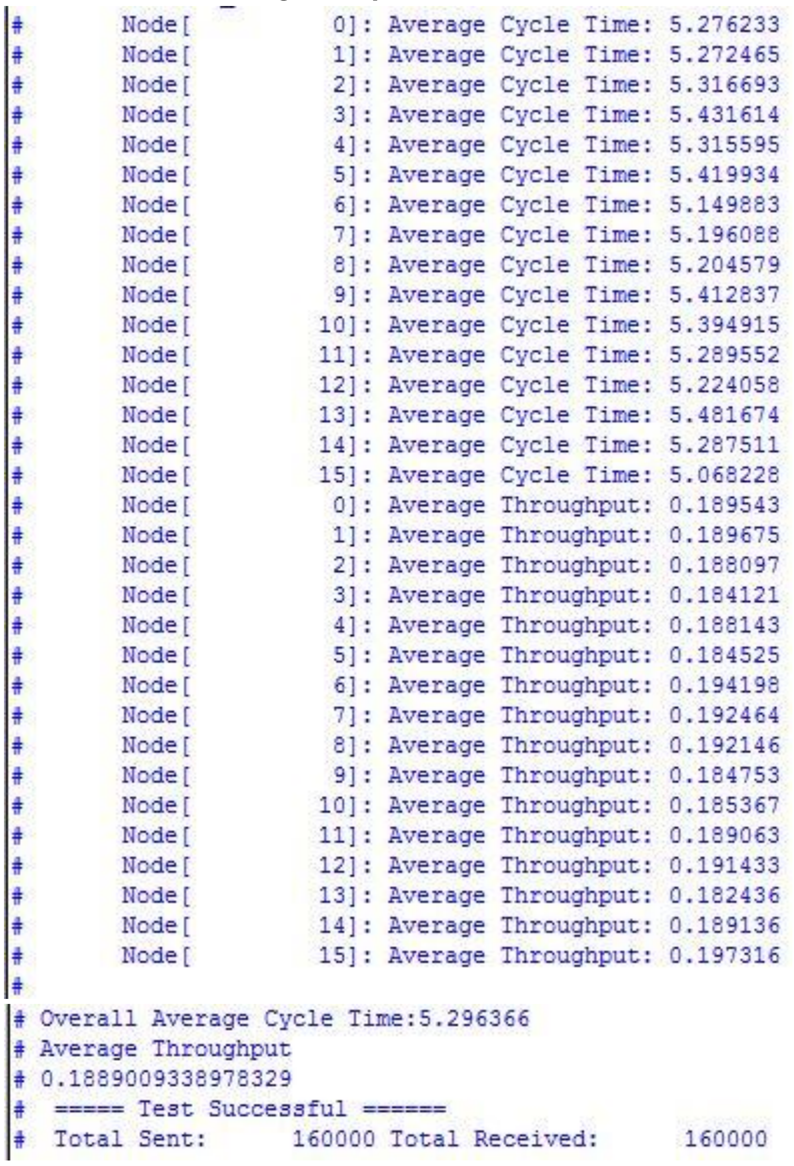


```
// Check the receive data if it matched
queue_check = tb_module.data_queue[MYID].find_first_index(x) with ( x == ReceiveValue);
if (tb_module.data_queue[MYID][queue_check[0]] == ReceiveValue) begin
//if matched start
  time_started = tb_module.time_queue[MYID][queue_check[0]];
  time_spent_during_travel = $time - time_started ;
  tb_module.sum_travel_time[MYID] += time_spent_during_travel;
  tb_module.time_queue[MYID].delete(queue_check[0]);

  tb_module.data_queue[MYID].delete(queue_check[0]);
end
  $display("Data bucket [%d] is receiving %b | Receive count: %d
Bucket Avg Cycle Time: %f, Sum Cycles: %f ", MYID,
ReceiveValue, tb_module.receive_count,
 (tb_module.sum_travel_time[MYID]/tb_module.cycle_queue[MYID]), tb_module.sum_travel_time[MYID]);
```

5.2 Test Cases and results

5.3 Performance

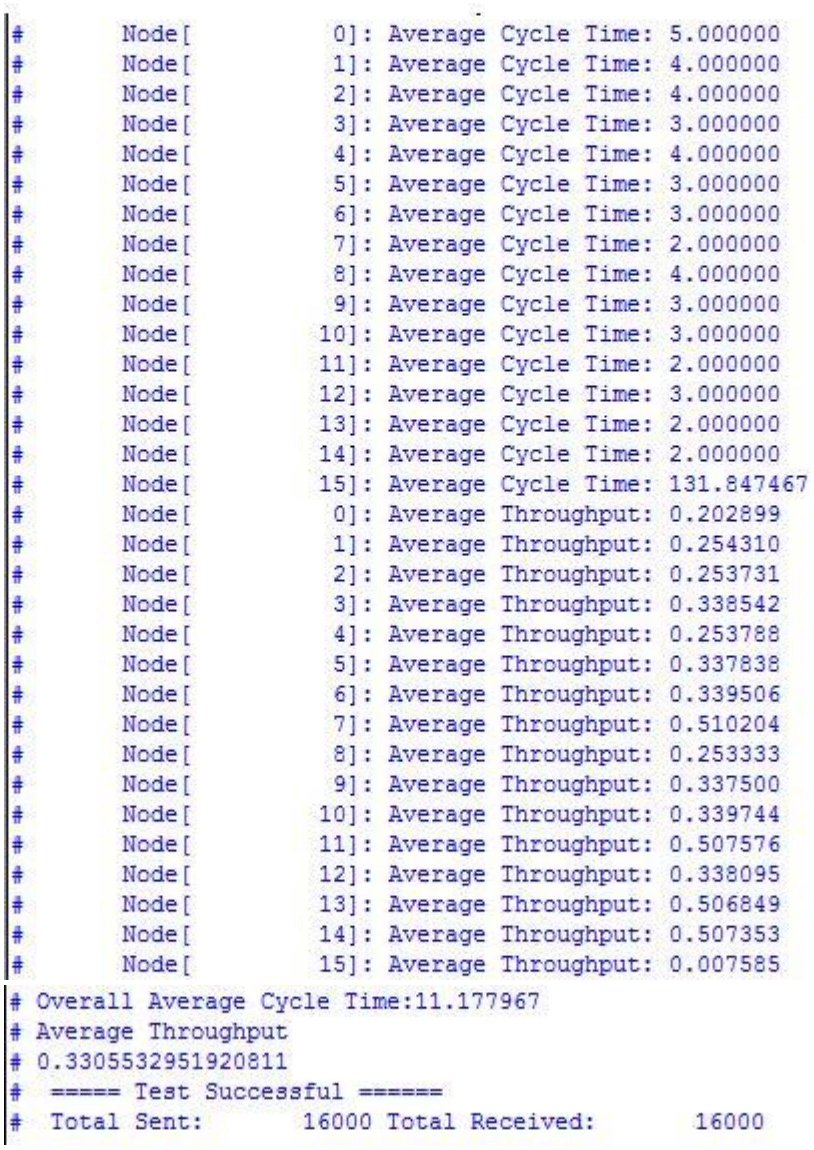
Test case1 : sending multiple data to random nodes



```
#       Node[          0]: Average Cycle Time: 5.276233
#       Node[          1]: Average Cycle Time: 5.272465
#       Node[          2]: Average Cycle Time: 5.316693
#       Node[          3]: Average Cycle Time: 5.431614
#       Node[          4]: Average Cycle Time: 5.315595
#       Node[          5]: Average Cycle Time: 5.419934
#       Node[          6]: Average Cycle Time: 5.149883
#       Node[          7]: Average Cycle Time: 5.196088
#       Node[          8]: Average Cycle Time: 5.204579
#       Node[          9]: Average Cycle Time: 5.412837
#       Node[         10]: Average Cycle Time: 5.394915
#       Node[         11]: Average Cycle Time: 5.289552
#       Node[         12]: Average Cycle Time: 5.224058
#       Node[         13]: Average Cycle Time: 5.481674
#       Node[         14]: Average Cycle Time: 5.287511
#       Node[         15]: Average Cycle Time: 5.068228
#       Node[          0]: Average Throughput: 0.189543
#       Node[          1]: Average Throughput: 0.189675
#       Node[          2]: Average Throughput: 0.188097
#       Node[          3]: Average Throughput: 0.184121
#       Node[          4]: Average Throughput: 0.188143
#       Node[          5]: Average Throughput: 0.184525
#       Node[          6]: Average Throughput: 0.194198
#       Node[          7]: Average Throughput: 0.192464
#       Node[          8]: Average Throughput: 0.192146
#       Node[          9]: Average Throughput: 0.184753
#       Node[         10]: Average Throughput: 0.185367
#       Node[         11]: Average Throughput: 0.189063
#       Node[         12]: Average Throughput: 0.191433
#       Node[         13]: Average Throughput: 0.182436
#       Node[         14]: Average Throughput: 0.189136
#       Node[         15]: Average Throughput: 0.197316
#
# Overall Average Cycle Time:5.296366
# Average Throughput
# 0.1889009338978329
#  ===== Test Successful ======
#  Total Sent:      160000 Total Received:      160000
```

To test the correctness of our design and to show that it is deadlock free, we made each node to send out 10000 random data to random nodes. Since we have 16 nodes, total number of data sent is 160000 and the test is successful as shown above.

Test case2 : sending multiple data to one specific node



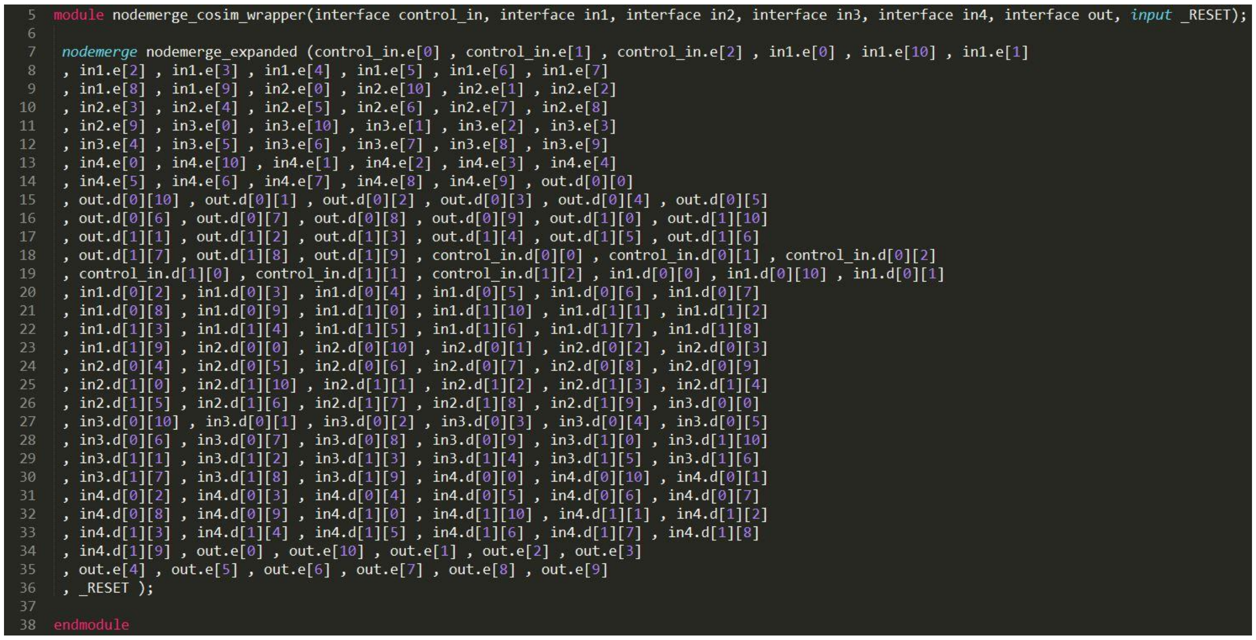
```
#       Node[          0]: Average Cycle Time: 5.000000
#       Node[          1]: Average Cycle Time: 4.000000
#       Node[          2]: Average Cycle Time: 4.000000
#       Node[          3]: Average Cycle Time: 3.000000
#       Node[          4]: Average Cycle Time: 4.000000
#       Node[          5]: Average Cycle Time: 3.000000
#       Node[          6]: Average Cycle Time: 3.000000
#       Node[          7]: Average Cycle Time: 2.000000
#       Node[          8]: Average Cycle Time: 4.000000
#       Node[          9]: Average Cycle Time: 3.000000
#       Node[         10]: Average Cycle Time: 3.000000
#       Node[         11]: Average Cycle Time: 2.000000
#       Node[         12]: Average Cycle Time: 3.000000
#       Node[         13]: Average Cycle Time: 2.000000
#       Node[         14]: Average Cycle Time: 2.000000
#       Node[         15]: Average Cycle Time: 131.847467
#       Node[          0]: Average Throughput: 0.202899
#       Node[          1]: Average Throughput: 0.254310
#       Node[          2]: Average Throughput: 0.253731
#       Node[          3]: Average Throughput: 0.338542
#       Node[          4]: Average Throughput: 0.253788
#       Node[          5]: Average Throughput: 0.337838
#       Node[          6]: Average Throughput: 0.339506
#       Node[          7]: Average Throughput: 0.510204
#       Node[          8]: Average Throughput: 0.253333
#       Node[          9]: Average Throughput: 0.337500
#       Node[         10]: Average Throughput: 0.339744
#       Node[         11]: Average Throughput: 0.507576
#       Node[         12]: Average Throughput: 0.338095
#       Node[         13]: Average Throughput: 0.506849
#       Node[         14]: Average Throughput: 0.507353
#       Node[         15]: Average Throughput: 0.007585
# Overall Average Cycle Time:11.177967
# Average Throughput
# 0.3305532951920811
#  ===== Test Successful ======
#  Total Sent:       16000 Total Received:       16000
```

The test bench shows correct results.

6.0 GATE IMPLEMENTATION

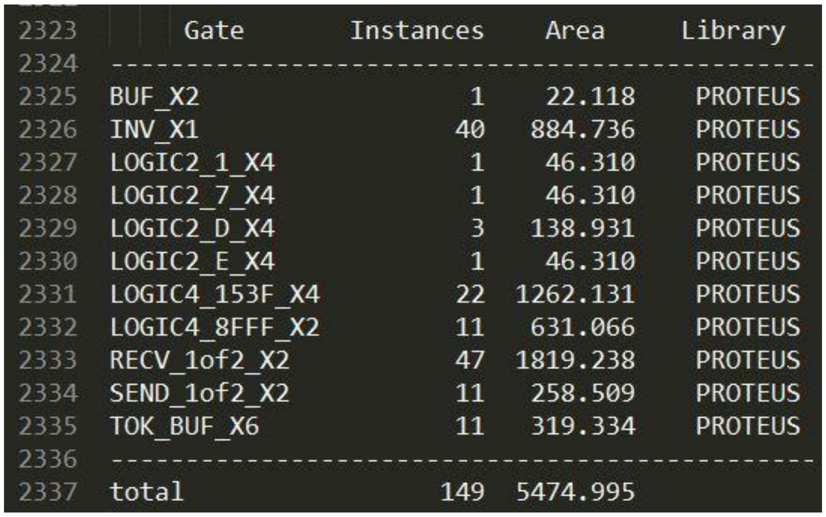
6.1 Synthesized Gate Implementation with Results

Node merge module cosim_wrapper gate implementation



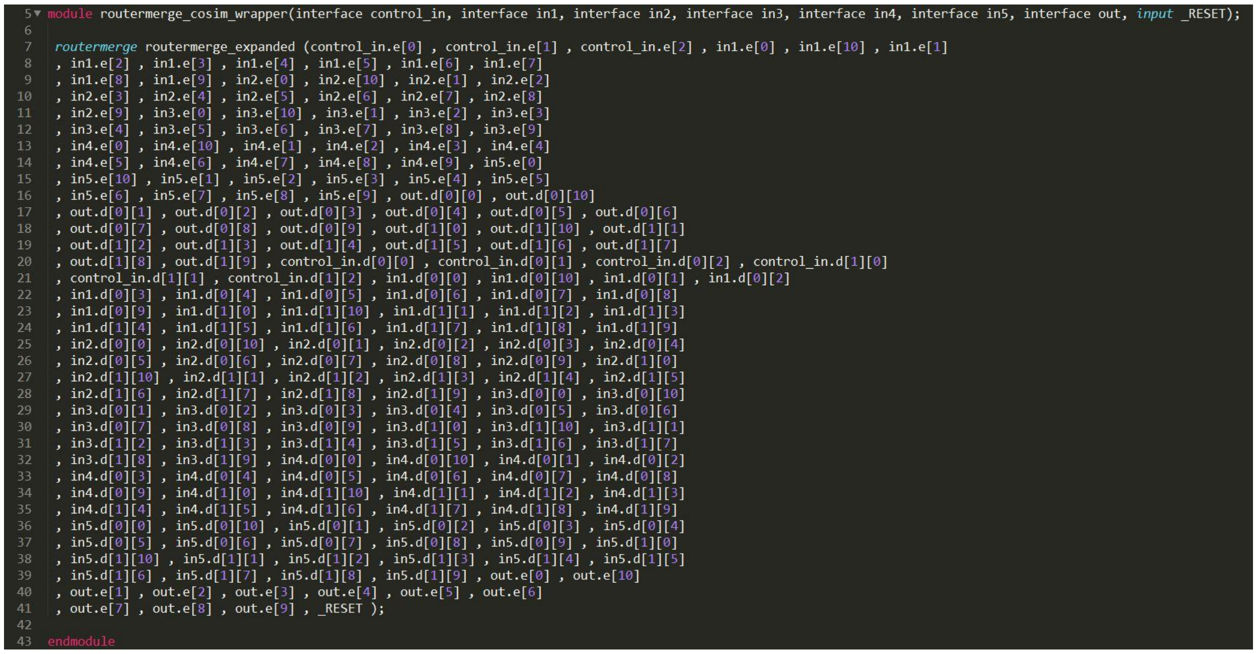
```
module nodemerge_cosim_wrapper(interface control_in, interface in1, interface in2, interface in3, interface in4, interface out, input _RESET);

 nodemerge nodemerge_expanded (control_in.e[0] , control_in.e[1] , control_in.e[2] , in1.e[0] , in1.e[10] , in1.e[1]
 , in1.e[2] , in1.e[3] , in1.e[4] , in1.e[5] , in1.e[6] , in1.e[7]
 , in1.e[8] , in1.e[9] , in2.e[0] , in2.e[10] , in2.e[1] , in2.e[2]
 , in2.e[3] , in2.e[4] , in2.e[5] , in2.e[6] , in2.e[7] , in2.e[8]
 , in2.e[9] , in3.e[0] , in3.e[10] , in3.e[1] , in3.e[2] , in3.e[3]
 , in3.e[4] , in3.e[5] , in3.e[6] , in3.e[7] , in3.e[8] , in3.e[9]
 , in4.e[0] , in4.e[10] , in4.e[1] , in4.e[2] , in4.e[3] , in4.e[4]
 , in4.e[5] , in4.e[6] , in4.e[7] , in4.e[8] , in4.e[9] , out.d[0][0]
 , out.d[0][10] , out.d[0][1] , out.d[0][2] , out.d[0][3] , out.d[0][4] , out.d[0][5]
 , out.d[0][6] , out.d[0][7] , out.d[0][8] , out.d[0][9] , out.d[1][0] , out.d[1][10]
 , out.d[1][1] , out.d[1][2] , out.d[1][3] , out.d[1][4] , out.d[1][5] , out.d[1][6]
 , out.d[1][7] , out.d[1][8] , out.d[1][9] , control_in.d[0][0] , control_in.d[0][1] , control_in.d[0][2]
 , control_in.d[1][0] , control_in.d[1][1] , control_in.d[1][2] , in1.d[0][0] , in1.d[0][10] , in1.d[0][1]
 , in1.d[0][2] , in1.d[0][3] , in1.d[0][4] , in1.d[0][5] , in1.d[0][6] , in1.d[0][7]
 , in1.d[0][8] , in1.d[0][9] , in1.d[1][0] , in1.d[1][10] , in1.d[1][1] , in1.d[1][2]
 , in1.d[1][3] , in1.d[1][4] , in1.d[1][5] , in1.d[1][6] , in1.d[1][7] , in1.d[1][8]
 , in1.d[1][9] , in2.d[0][0] , in2.d[0][10] , in2.d[0][1] , in2.d[0][2] , in2.d[0][3]
 , in2.d[0][4] , in2.d[0][5] , in2.d[0][6] , in2.d[0][7] , in2.d[0][8] , in2.d[0][9]
 , in2.d[1][0] , in2.d[1][10] , in2.d[1][1] , in2.d[1][2] , in2.d[1][3] , in2.d[1][4]
 , in2.d[1][5] , in2.d[1][6] , in2.d[1][7] , in2.d[1][8] , in2.d[1][9] , in3.d[0][0]
 , in3.d[0][10] , in3.d[0][1] , in3.d[0][2] , in3.d[0][3] , in3.d[0][4] , in3.d[0][5]
 , in3.d[0][6] , in3.d[0][7] , in3.d[0][8] , in3.d[0][9] , in3.d[1][0] , in3.d[1][10]
 , in3.d[1][1] , in3.d[1][2] , in3.d[1][3] , in3.d[1][4] , in3.d[1][5] , in3.d[1][6]
 , in3.d[1][7] , in3.d[1][8] , in3.d[1][9] , in4.d[0][0] , in4.d[0][10] , in4.d[0][1]
 , in4.d[0][2] , in4.d[0][3] , in4.d[0][4] , in4.d[0][5] , in4.d[0][6] , in4.d[0][7]
 , in4.d[0][8] , in4.d[0][9] , in4.d[1][0] , in4.d[1][10] , in4.d[1][1] , in4.d[1][2]
 , in4.d[1][3] , in4.d[1][4] , in4.d[1][5] , in4.d[1][6] , in4.d[1][7] , in4.d[1][8]
 , in4.d[1][9] , out.e[0] , out.e[10] , out.e[1] , out.e[2] , out.e[3]
 , out.e[4] , out.e[5] , out.e[6] , out.e[7] , out.e[8] , out.e[9]
 , _RESET );

endmodule
```

Node merge gate implementation stats



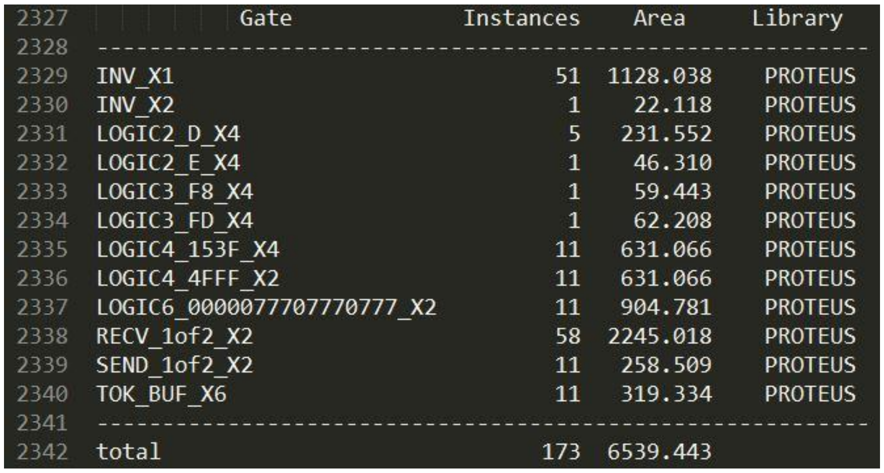
```
      Gate          Instances    Area      Library
----------------------------------------------------
BUF_X2                    1      22.118     PROTEUS
INV_X1                   40     884.736     PROTEUS
LOGIC2_1_X4               1      46.310     PROTEUS
LOGIC2_7_X4               1      46.310     PROTEUS
LOGIC2_D_X4               3     138.931     PROTEUS
LOGIC2_E_X4               1      46.310     PROTEUS
LOGIC4_153F_X4           22    1262.131     PROTEUS
LOGIC4_8FFF_X2           11     631.066     PROTEUS
RECV_1of2_X2             47    1819.238     PROTEUS
SEND_1of2_X2             11     258.509     PROTEUS
TOK_BUF_X6               11     319.334     PROTEUS
----------------------------------------------------
total                   149    5474.995
```

Router merge module cosim_wrapper gate implementation



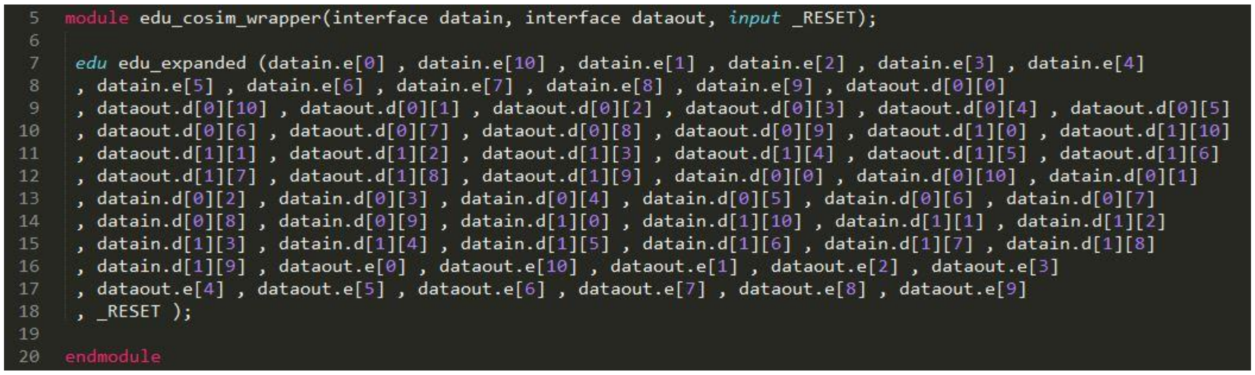
```
module routermerge_cosim_wrapper(interface control_in, interface in1, interface in2, interface in3, interface in4, interface in5, interface out, input _RESET);

 routermerge routermerge_expanded (control_in.e[0] , control_in.e[1] , control_in.e[2] , in1.e[0] , in1.e[10] , in1.e[1]
 , in1.e[2] , in1.e[3] , in1.e[4] , in1.e[5] , in1.e[6] , in1.e[7]
 , in1.e[8] , in1.e[9] , in2.e[0] , in2.e[10] , in2.e[1] , in2.e[2]
 , in2.e[3] , in2.e[4] , in2.e[5] , in2.e[6] , in2.e[7] , in2.e[8]
 , in2.e[9] , in3.e[0] , in3.e[10] , in3.e[1] , in3.e[2] , in3.e[3]
 , in3.e[4] , in3.e[5] , in3.e[6] , in3.e[7] , in3.e[8] , in3.e[9]
 , in4.e[0] , in4.e[10] , in4.e[1] , in4.e[2] , in4.e[3] , in4.e[4]
 , in4.e[5] , in4.e[6] , in4.e[7] , in4.e[8] , in4.e[9] , in5.e[0]
 , in5.e[10] , in5.e[1] , in5.e[2] , in5.e[3] , in5.e[4] , in5.e[5]
 , in5.e[6] , in5.e[7] , in5.e[8] , in5.e[9] , out.d[0][0] , out.d[0][10]
 , out.d[0][1] , out.d[0][2] , out.d[0][3] , out.d[0][4] , out.d[0][5] , out.d[0][6]
 , out.d[0][7] , out.d[0][8] , out.d[0][9] , out.d[1][0] , out.d[1][10] , out.d[1][1]
 , out.d[1][2] , out.d[1][3] , out.d[1][4] , out.d[1][5] , out.d[1][6] , out.d[1][7]
 , out.d[1][8] , out.d[1][9] , control_in.d[0][0] , control_in.d[0][1] , control_in.d[0][2] , control_in.d[1][0]
 , control_in.d[1][1] , control_in.d[1][2] , in1.d[0][0] , in1.d[0][10] , in1.d[0][1] , in1.d[0][2]
 , in1.d[0][3] , in1.d[0][4] , in1.d[0][5] , in1.d[0][6] , in1.d[0][7] , in1.d[0][8]
 , in1.d[0][9] , in1.d[1][0] , in1.d[1][10] , in1.d[1][1] , in1.d[1][2] , in1.d[1][3]
 , in1.d[1][4] , in1.d[1][5] , in1.d[1][6] , in1.d[1][7] , in1.d[1][8] , in1.d[1][9]
 , in2.d[0][0] , in2.d[0][10] , in2.d[0][1] , in2.d[0][2] , in2.d[0][3] , in2.d[0][4]
 , in2.d[0][5] , in2.d[0][6] , in2.d[0][7] , in2.d[0][8] , in2.d[0][9] , in2.d[1][0]
 , in2.d[1][10] , in2.d[1][1] , in2.d[1][2] , in2.d[1][3] , in2.d[1][4] , in2.d[1][5]
 , in2.d[1][6] , in2.d[1][7] , in2.d[1][8] , in2.d[1][9] , in3.d[0][0] , in3.d[0][10]
 , in3.d[0][1] , in3.d[0][2] , in3.d[0][3] , in3.d[0][4] , in3.d[0][5] , in3.d[0][6]
 , in3.d[0][7] , in3.d[0][8] , in3.d[0][9] , in3.d[1][0] , in3.d[1][10] , in3.d[1][1]
 , in3.d[1][2] , in3.d[1][3] , in3.d[1][4] , in3.d[1][5] , in3.d[1][6] , in3.d[1][7]
 , in3.d[1][8] , in3.d[1][9] , in4.d[0][0] , in4.d[0][10] , in4.d[0][1] , in4.d[0][2]
 , in4.d[0][3] , in4.d[0][4] , in4.d[0][5] , in4.d[0][6] , in4.d[0][7] , in4.d[0][8]
 , in4.d[0][9] , in4.d[1][0] , in4.d[1][10] , in4.d[1][1] , in4.d[1][2] , in4.d[1][3]
 , in4.d[1][4] , in4.d[1][5] , in4.d[1][6] , in4.d[1][7] , in4.d[1][8] , in4.d[1][9]
 , in5.d[0][0] , in5.d[0][10] , in5.d[0][1] , in5.d[0][2] , in5.d[0][3] , in5.d[0][4]
 , in5.d[0][5] , in5.d[0][6] , in5.d[0][7] , in5.d[0][8] , in5.d[0][9] , in5.d[1][0]
 , in5.d[1][10] , in5.d[1][1] , in5.d[1][2] , in5.d[1][3] , in5.d[1][4] , in5.d[1][5]
 , in5.d[1][6] , in5.d[1][7] , in5.d[1][8] , in5.d[1][9] , out.e[0] , out.e[10]
 , out.e[1] , out.e[2] , out.e[3] , out.e[4] , out.e[5] , out.e[6]
 , out.e[7] , out.e[8] , out.e[9] , _RESET );

endmodule
```

Router merge gate implementation stats



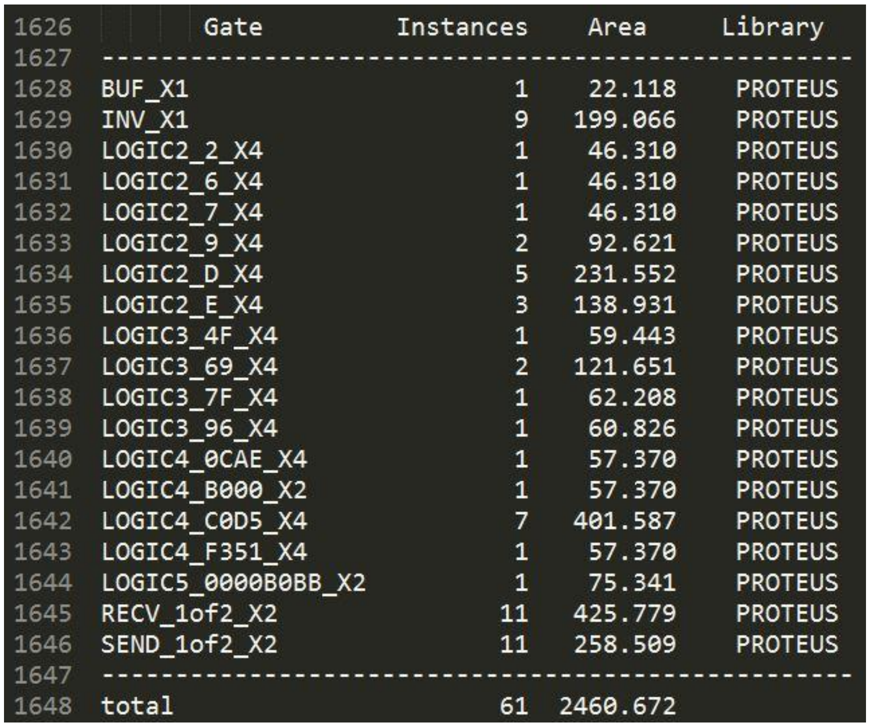
```
             Gate               Instances     Area      Library
---------------------------------------------------------------
INV_X1                                 51  1128.038     PROTEUS
INV_X2                                  1    22.118     PROTEUS
LOGIC2_D_X4                             5   231.552     PROTEUS
LOGIC2_E_X4                             1    46.310     PROTEUS
LOGIC3_F8_X4                            1    59.443     PROTEUS
LOGIC3_FD_X4                            1    62.208     PROTEUS
LOGIC4_153F_X4                         11   631.066     PROTEUS
LOGIC4_4FFF_X2                         11   631.066     PROTEUS
LOGIC6_0000777707770777_X2             11   904.781     PROTEUS
RECV_1of2_X2                           58  2245.018     PROTEUS
SEND_1of2_X2                           11   258.509     PROTEUS
TOK_BUF_X6                             11   319.334     PROTEUS
---------------------------------------------------------------
total                                 173  6539.443
```

Router merge module cosim_wrapper gate implementation



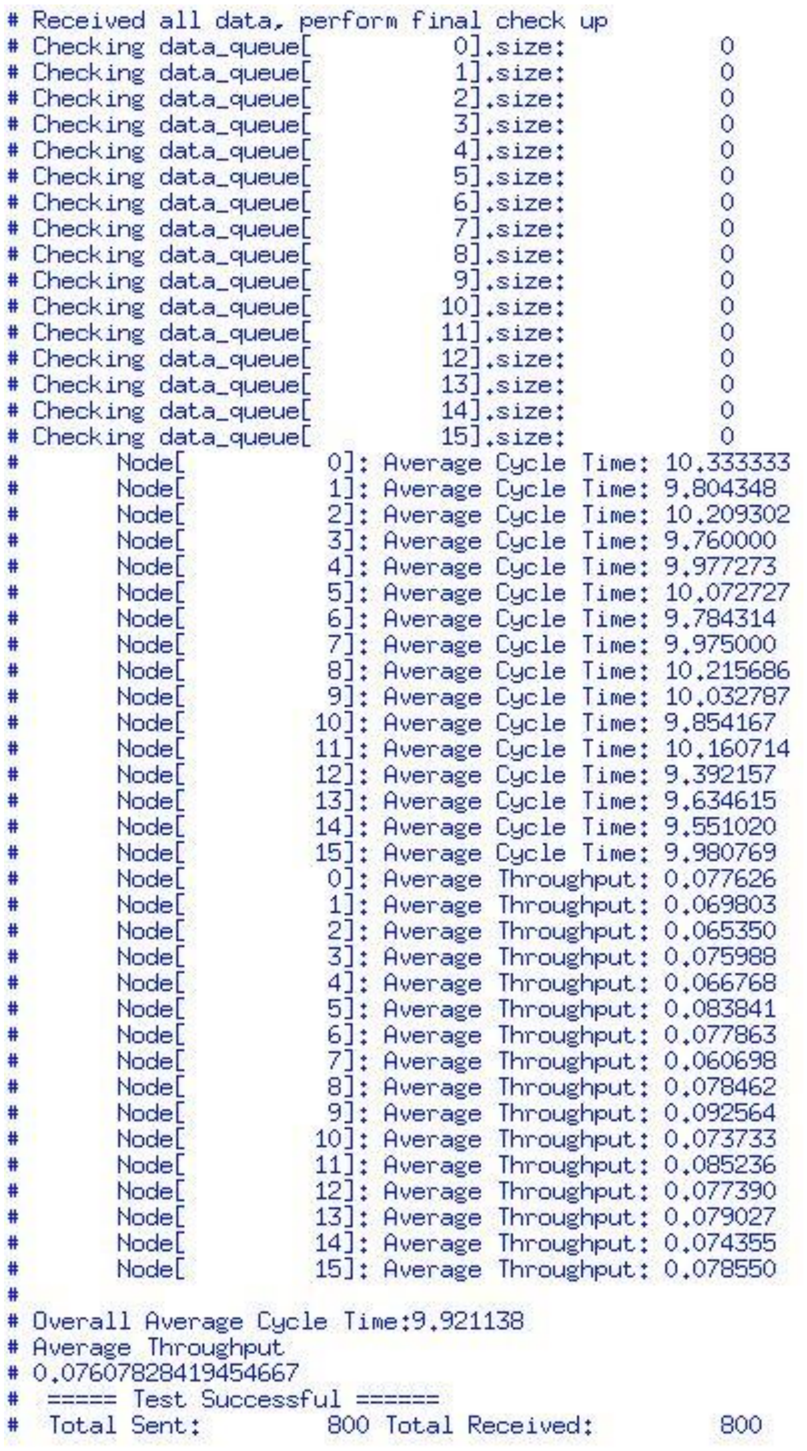
```
module edu_cosim_wrapper(interface datain, interface dataout, input _RESET);

 edu edu_expanded (datain.e[0] , datain.e[10] , datain.e[1] , datain.e[2] , datain.e[3] , datain.e[4]
 , datain.e[5] , datain.e[6] , datain.e[7] , datain.e[8] , datain.e[9] , dataout.d[0][0]
 , dataout.d[0][10] , dataout.d[0][1] , dataout.d[0][2] , dataout.d[0][3] , dataout.d[0][4] , dataout.d[0][5]
 , dataout.d[0][6] , dataout.d[0][7] , dataout.d[0][8] , dataout.d[0][9] , dataout.d[1][0] , dataout.d[1][10]
 , dataout.d[1][1] , dataout.d[1][2] , dataout.d[1][3] , dataout.d[1][4] , dataout.d[1][5] , dataout.d[1][6]
 , dataout.d[1][7] , dataout.d[1][8] , dataout.d[1][9] , datain.d[0][0] , datain.d[0][10] , datain.d[0][1]
 , datain.d[0][2] , datain.d[0][3] , datain.d[0][4] , datain.d[0][5] , datain.d[0][6] , datain.d[0][7]
 , datain.d[0][8] , datain.d[0][9] , datain.d[1][0] , datain.d[1][10] , datain.d[1][1] , datain.d[1][2]
 , datain.d[1][3] , datain.d[1][4] , datain.d[1][5] , datain.d[1][6] , datain.d[1][7] , datain.d[1][8]
 , datain.d[1][9] , dataout.e[0] , dataout.e[10] , dataout.e[1] , dataout.e[2] , dataout.e[3]
 , dataout.e[4] , dataout.e[5] , dataout.e[6] , dataout.e[7] , dataout.e[8] , dataout.e[9]
 , _RESET );

endmodule
```

Router merge gate implementation stats



```
             Gate          Instances     Area      Library
---------------------------------------------------
BUF_X1                             1     22.118     PROTEUS
INV_X1                             9    199.066     PROTEUS
LOGIC2_2_X4                        1     46.310     PROTEUS
LOGIC2_6_X4                        1     46.310     PROTEUS
LOGIC2_7_X4                        1     46.310     PROTEUS
LOGIC2_9_X4                        2     92.621     PROTEUS
LOGIC2_D_X4                        5    231.552     PROTEUS
LOGIC2_E_X4                        3    138.931     PROTEUS
LOGIC3_4F_X4                       1     59.443     PROTEUS
LOGIC3_69_X4                       2    121.651     PROTEUS
LOGIC3_7F_X4                       1     62.208     PROTEUS
LOGIC3_96_X4                       1     60.826     PROTEUS
LOGIC4_0CAE_X4                     1     57.370     PROTEUS
LOGIC4_B000_X2                     1     57.370     PROTEUS
LOGIC4_C0D5_X4                     7    401.587     PROTEUS
LOGIC4_F351_X4                     1     57.370     PROTEUS
LOGIC5_0000B0BB_X2                 1     75.341     PROTEUS
RECV_1of2_X2                      11    425.779     PROTEUS
SEND_1of2_X2                      11    258.509     PROTEUS
---------------------------------------------------
total                             61   2460.672
```

6.2 Synthesized Top Level Performance



```
# Received all data, perform final check up
# Checking data_queue[          0].size:           0
# Checking data_queue[          1].size:           0
# Checking data_queue[          2].size:           0
# Checking data_queue[          3].size:           0
# Checking data_queue[          4].size:           0
# Checking data_queue[          5].size:           0
# Checking data_queue[          6].size:           0
# Checking data_queue[          7].size:           0
# Checking data_queue[          8].size:           0
# Checking data_queue[          9].size:           0
# Checking data_queue[         10].size:           0
# Checking data_queue[         11].size:           0
# Checking data_queue[         12].size:           0
# Checking data_queue[         13].size:           0
# Checking data_queue[         14].size:           0
# Checking data_queue[         15].size:           0
#       Node[          0]: Average Cycle Time: 10.333333
#       Node[          1]: Average Cycle Time: 9.804348
#       Node[          2]: Average Cycle Time: 10.209302
#       Node[          3]: Average Cycle Time: 9.760000
#       Node[          4]: Average Cycle Time: 9.977273
#       Node[          5]: Average Cycle Time: 10.072727
#       Node[          6]: Average Cycle Time: 9.784314
#       Node[          7]: Average Cycle Time: 9.975000
#       Node[          8]: Average Cycle Time: 10.215686
#       Node[          9]: Average Cycle Time: 10.032787
#       Node[         10]: Average Cycle Time: 9.854167
#       Node[         11]: Average Cycle Time: 10.160714
#       Node[         12]: Average Cycle Time: 9.392157
#       Node[         13]: Average Cycle Time: 9.634615
#       Node[         14]: Average Cycle Time: 9.551020
#       Node[         15]: Average Cycle Time: 9.980769
#       Node[          0]: Average Throughput: 0.077626
#       Node[          1]: Average Throughput: 0.069803
#       Node[          2]: Average Throughput: 0.065350
#       Node[          3]: Average Throughput: 0.075988
#       Node[          4]: Average Throughput: 0.066768
#       Node[          5]: Average Throughput: 0.083841
#       Node[          6]: Average Throughput: 0.077863
#       Node[          7]: Average Throughput: 0.060698
#       Node[          8]: Average Throughput: 0.078462
#       Node[          9]: Average Throughput: 0.092564
#       Node[         10]: Average Throughput: 0.073733
#       Node[         11]: Average Throughput: 0.085236
#       Node[         12]: Average Throughput: 0.077390
#       Node[         13]: Average Throughput: 0.079027
#       Node[         14]: Average Throughput: 0.074355
#       Node[         15]: Average Throughput: 0.078550
#
# Overall Average Cycle Time:9.921138
# Average Throughput
# 0.07607828419454667
#  ===== Test Successful ======
#  Total Sent:         800 Total Received:         800
```

The test bench shows correct results.