# Gold Bidirectional Ring NoC Router
# Project Part 1
**Assigned: February 24, 2016**
**Due: March 11, 2016 (11:50pm)**

Objective

The goal is for students to design the Gold bidirectional ring network-on-chip router using a synthesizeable RTL style of Verilog and verify through appropriate testbenches. The emphasis is on correct functionality, not timing, for this part of the project. You should not be using delays in the design itself, as we will attempt to further synthesize the code in later stages of the project. However, the testbench will need to use delays for simulating the design at a given clock frequency. To establish a point of consistency across all projects, simulate at a clock cycle of 4ns (250MHz). For further verification of your router's functionality, you will also design and test a ring of four Gold routers.

Teaming Policies

Everyone should work in 2-student teams for the project. To work as an individual, you must obtain the instructor's explicit permission via email, and you will only be granted this permission if it is clear that all options for working with a teammate have been exhausted. For this phase of the project, the same amount of work is required, regardless of whether you work as an individual or as a team.

Verilog Setup

Use the Verilog environment that you used for Homework 3, 4, and 5 to complete this project phase. Specifically, use the NC-Sim environment that has been recommended in the class. Codes compiling with other tools but not with the NC-Sim environment will be penalized severely in a similar manner to a non-compiling code.

What to do?

1. Download the Gold Bidirectional Ring NoC Router Architecture Specification from the course web site.
2. Write RTL verilog code that correctly implements all the functionality detailed in the architecture specification. Make sure to use a top-level module name of "gold_router" and top-level port names consistent with the inputs/outputs shown in Figure 1 of the architecture specification so that any testbenches we use to test your router will be compatible without alteration.
3. Write a testbench to verify your design. For this phase of the project, focus on testing your router's switching/forwarding capability (do packets coming in get routed to the correct output channel based on the routing header and does the packet header get updated appropriately?) and output arbitration fairness (does your output controller follow the priority given in the architecture specification when multiple input channels are requesting concurrently?). To test the arbitration fairness, your testbench will need to inject concurrent packets on the input channels, so some input channels will temporarily get blocked, specifically any input channel that is not granted access to the requested output channel. Also, you should check that handshaking operates correctly, even in cases of blocking. For completeness, you should repeat all tests for both virtual channel

levels of the router. You may wish to use the polarity output of the router as an input to your testbench to trigger actions targeted at specific virtual channels. Your testbench should use unique values in packet data content so that packets can easily be distinguished from each other.

4. Simulate your router. You may wish to perform pre-tests on subcomponent modules before integrating all the modules.
5. Synthesize your router to simply prove that your Verilog is synthesizeable with no errors. The goal of this task is to prove that you have synthesizeable code. So while the clock target is not critical, use a target clock period of 4ns, but if you must relax the target to complete synthesis, that's allowable. Be sure to submit the final script showing the clock period with your design. Submit your synthesis script (gold_router.tcl), resulting netlist (gold_router.syn.v) along with the output of the `check_design` command (gold_router.log). Check the output of the `check_design` command to make sure there are NO LATCHES generated. If so, modify your RTL to remove any latches in the design.
6. To further confirm the operational capability of your router, construct a 4-node ring network by instantiating your router 4 times and making connections as shown below in Figure 1. Name your top-level module "gold_ring".

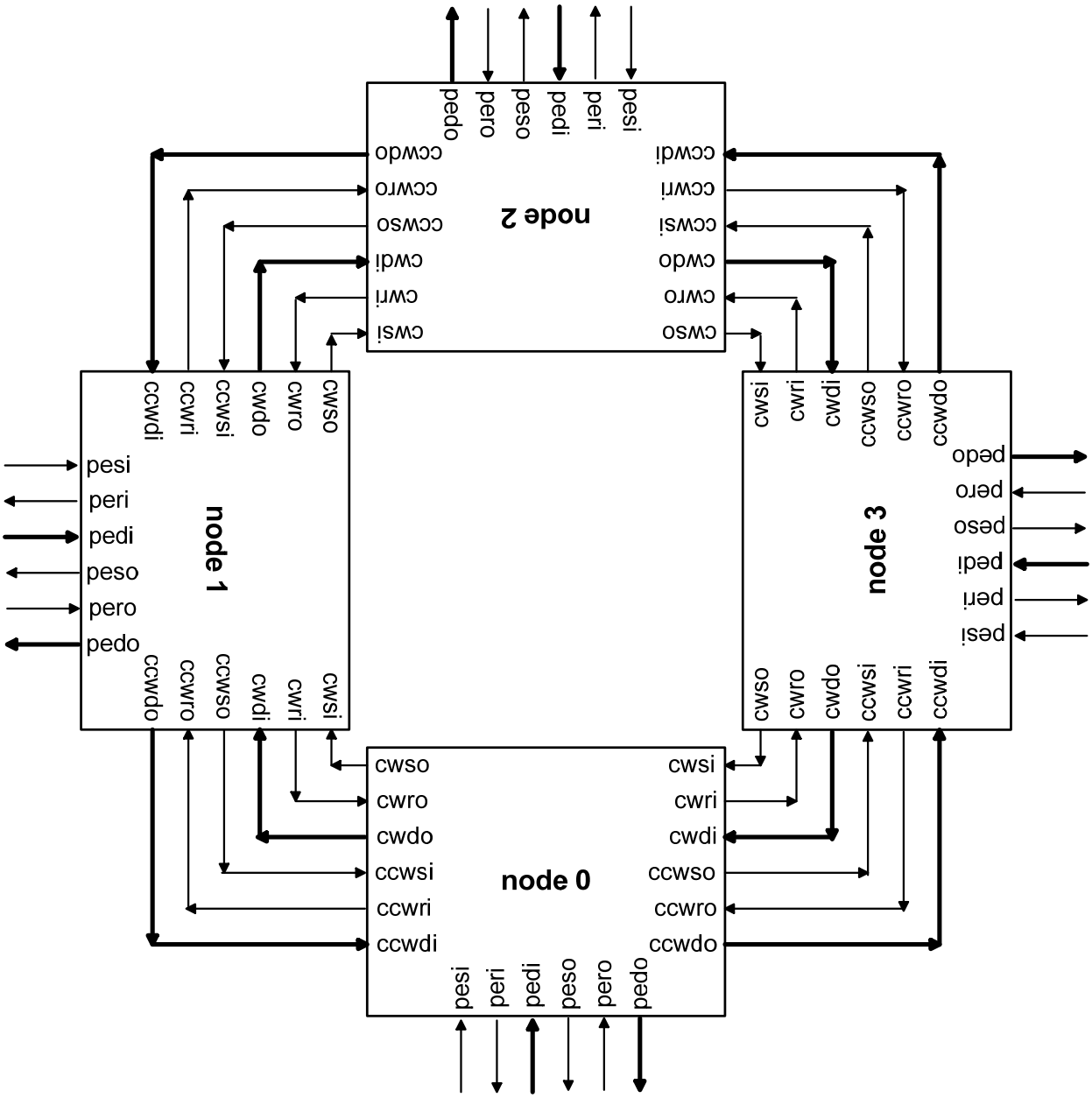


Figure 1: Gold 4-Node Ring Connections

Given the connections shown in Figure 1, the top-level ports of the “gold_ring” module are shown in **Error! Reference source not found.**. Make sure to use these top-level port names as shown in Figure 2 so that any testbenches we use to test your ring network will be compatible without alteration.

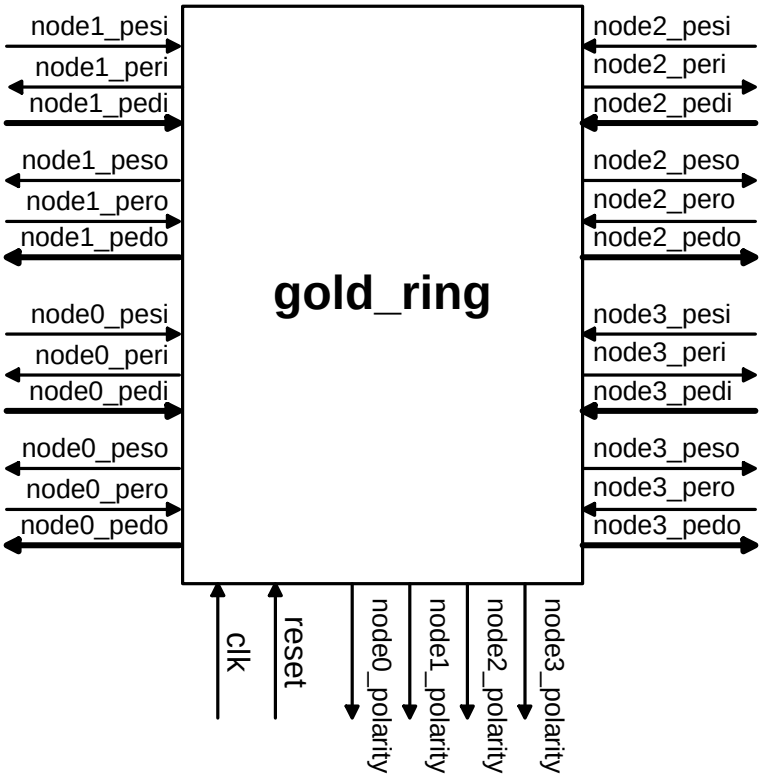


Figure 2: Top-Level Ports of the gold_ring Module
(*di and *do signals are 64-bits wide)

7. Finally, to test the 4-node Gold ring network, construct a testbench consisting of multiple phases of a *gather* traffic pattern as described below. A gather traffic pattern is an all-to-one communication traffic pattern, where every node (except the destination node) in the network sends to one node. For example, Figure 3 depicts the packets sent for a gather traffic case when every node transmits a packet to node 0. This constitutes phase 0 of this traffic application. Notice that in our network environment every node is a potential sender and a potential receiver. By term potential we mean that every node does not send and receive in every phase (e.g., node 0 is not sending a packet to itself in phase 0, Figure 3).There are four different gather traffic patterns possible in our Gold ring. In each type of gather traffic, one node of the network acts as the destination node and all other nodes (excluding the destination node) act as source nodes, which send one packet to the common destination. We use the term phase K to describe the gather operation when node K (K varies from 0 to 3, Figure 3) acts as the destination. So, there are a total of 4 phases, in which we can verify that every node in the network can send and receive packets from each other.

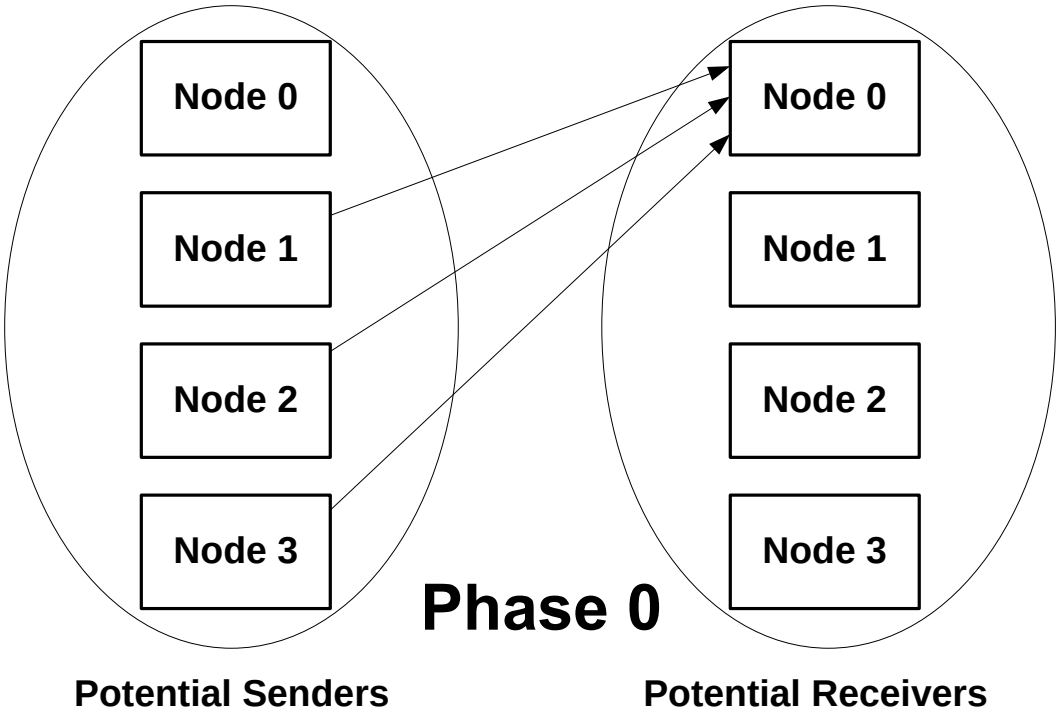


Figure 3: Illustration of Phase 0 of the gather Testbench

The following rules must be followed to ensure shortest-path deadlock-free routing in ring networks implemented with Gold routers. First, a shortest distance route between source and destination must be used. For instance, to send a packet from node 0 to node 3, a packet should be routed counter-clockwise for 1 hop rather than clockwise for 3 hops. Given this, the hop count field of any routing header of a packet injected into the 4-node Gold ring network though a pe channel will never be more than 8'h02. Secondly, a virtual channel allocation scheme must be used to ensure deadlock freedom. This is accomplished by designating that nodes 0 and 1 always inject packets into the even pe virtual channels, i.e., apply external inputs on odd polarity cycles, and nodes 2 and 3 always inject packets into the odd pe virtual channels, i.e., apply external inputs on even polarity cycles. Since packets always stay on the virtual level (even or odd) that they are injected on for their entire traversal in the network from source to destination, this scheme will ensure that deadlock cannot occur. Finally, for packets traversing the maximum distance of two hops, in practice it doesn't matter if the cw or ccw direction is followed. However, for consistency across all projects, always route such packets in the cw direction.

For this testbench, packets should be constructed as follows. First, the source field of any packet should contain the node number of the node that sent the packet. Therefore, for our simple 4-node network, the source field can have values in the range 16'h0000-16'h0003, corresponding to the nodes 0-3 being the source nodes, respectively. The payload field should be set to contain the node number of the destination for which this packet is intended. The vc, direction, and hop count fields of the routing header must be

set appropriately for each packet based on the source and destination of the router. Note, however, that the vc setting has no functional effect. The packet will need to be injected at the correct polarity for a particular source to guarantee that is injected on the right vc polarity. The vc field will be used by other components in our multi-core processor later in the semester. Given these stipulations and routing constraints described earlier, the packets sent for Phase 0 of the gather operation are shown in Figure 4. You will need to construct similar packets for the other 3 phases of the operation.

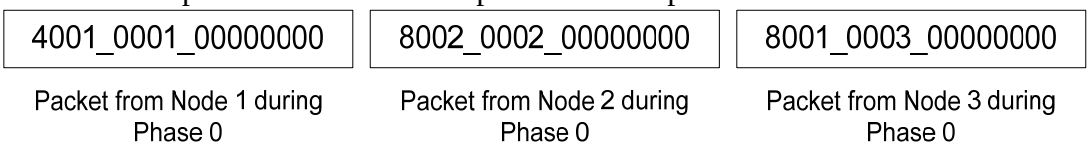


Figure 4: Packet Contents for Phase 0 of *gather* Testbench

Since every packet will contain a unique source and payload combination, it will be easy to debug and to check the proper sequencing of packets, from the point of injection until ejection from the network.

Given all of the above, you are to design a testbench, with module name "tb_gold_ring", instantiating the 4-node ring topology. The testbench should meet the following specifications to execute a 4-phase gather operation:

(a) Some mechanism of injecting packets from each node into the pe input channel of its corresponding router using appropriate signals from the port list of the module "gold_ring" and on the appropriate vc polarity should be introduced in the testbench.

(b) Some mechanism of logging the relevant information in a file for each phase, when a packet arrives at a particular node. For example, the result file containing information gathered in phase 0 will be called "gather_phase0.out". The relevant information should be in the following format for every packet arriving at a pe output channel:

```
Phase= , Time= , Destination= ,  Source= , Packet Value=
```

There should be entries in the log file only when a packet is received. (That is, do NOT create entries for every clock cycle in the log file; it is not useful to show cycles when a channel is idle.)

You will have 4 result files if you meet this specification correctly. The names of the files should be: gather_phase0.out, gather_phase1.out, gather_phase2.out, gather_phase3.out

(Hint: Use generic modules for packet injection and logging with some defparam statements, if required, for customizing each module instantiated in the top-level module. This will greatly ease the task and allow you to reuse large amounts of code.)

(c) Note that the testbench will need to drive the pero signals appropriately to allow the pe output channels to forward data as needed.

(d) Apply 4 phases (phase 0 – phase 3) of gather traffic to your network consecutively from your testbench. Before starting a next phase, your testbench should wait for the completion of the previous phase. That is, you cannot start phase 1 unless all 3 packets have been received for phase 0 at node 0.

Write the start and completion time of each phase in a file called "start_end_time.out" in the following format:

```
Phase= , Start Time=
Phase= , Completion Time=
```

Start time is defined to be the time when the first packet is injected by any node in a particular phase. Completion time is defined to be the time when the last packet is received (clk cycle in which the packet data is driven on pedo channel of destination) in that particular phase.

Given all of the above, you are to design a testbench, with module name "tb_gold_ring", instantiating the 4-node ring topology. The testbench should meet the following specifications to execute a 4-phase gather operation:

(e) Some mechanism of injecting packets from each node into the pe input channel of its corresponding router using appropriate signals from the port list of the module "gold_ring" and on the appropriate vc polarity should be introduced in the testbench.

(f) Some mechanism of logging the relevant information in a file for each phase, when a packet arrives at a particular node. For example, the result file containing information gathered in phase 0 will be called "gather_phase0.out". The relevant information should be in the following format for every packet arriving at a pe output channel:

```
Phase= , Time= , Destination= ,  Source= , Packet Value=
```

There should be entries in the log file only when a packet is received. (That is, do NOT create entries for every clock cycle in the log file; it is not useful to show cycles when a channel is idle.)

You will have 4 result files if you meet this specification correctly. The names of the files should be: gather_phase0.out, gather_phase1.out, gather_phase2.out, gather_phase3.out

(Hint: Use generic modules for packet injection and logging with some defparam statements, if required, for customizing each module instantiated in the top-level module. This will greatly ease the task and allow you to reuse large amounts of code.)

(g) Note that the testbench will need to drive the pero signals appropriately to allow the pe output channels to forward data as needed.

(h) Apply 4 phases (phase 0 – phase 3) of gather traffic to your network consecutively from your testbench. Before starting a next phase, your testbench should wait for the completion of the previous phase. That is, you cannot start phase 1 unless all 3 packets have been received for phase 0 at node 0.

Write the start and completion time of each phase in a file called "start_end_time.out" in the following format:

```
Phase= , Start Time=
```

```
Phase= , Completion Time=
```

Start time is defined to be the time when the first packet is injected by any node in a particular phase. Completion time is defined to be the time when the last packet is received (clk cycle in which the packet data is driven on pedo channel of destination) in that particular phase.

What to submit?

Submit the complete Verilog code including the router design, ring design and testbenches for both the router by itself as well as the gather testbench specified for the ring. Please include adequate comments not only in your designs but also in your testbenches to justify that your testbenches adequately verify your design. You should also submit output from display/monitor commands, especially for all input/output signals of the router design. Use good practices. For example, in your display/monitor statements, you should also include the time that corresponds with the reported values that are being monitored. Also, use signal labels that correspond to the signal names in the architecture specification and name the top-level router module gold_router and similarly name the top-level ring module gold_ring. In addition, include a short report (Word document that is 3 pages or less) that identifies both team members and tells how the work was distributed, and the rationale for the standalone router testbench design.

Submission:

1. Router files: gold_router.v (top-level design file) and any supporting Verilog files, tb_gold_router.v (testbench), tb_gold_router.res (output from running testbench), gold_router.tcl (synthesis script), gold_router.syn.v (synthesized netlist), gold_router.log (synthesis check log), gold_router.doc (report)
2. Ring network files: gold_ring.v, tb_gold_ring.v, gather_phase0.out, gather_phase1.out, gather_phase2.out, gather_phase3.out, start_end_time.out

Be sure to add files that are needed by ‘include commands in any of these files.

How to submit?

We will use electronic submission. For electronic submission, **follow the instructions given below strictly.** Use the directory provided with the assignment to organize your submission. “`tar`” all the required files for the project using the “**`make submit`**” command and use the upload link in DEN for electronic submission. Use “**`make submit`**” command to create the tar-file, this allows grading team to use automated tools to grade assignment.

(1) Modify **`AUTHORS`** file by adding name of the team and team members. Write name of the team in the first line of the file, followed with team-member names on subsequent lines.
(2) Modify the **`makefile`** to declare name of your team, this will be used to create the submission tar-file.
(3) All of your verilog design files should be in **`design/`** directory. All test-bench files should be in **`tb/`** directory. All the synthesis script files should be kept in **`script/`** directory. The generated netlists (*.syn.v) and SDF files (*.sdf) should be kept in **`netlist/`** and **`sdf/`** directory respectively. All generated outputs (.out), logs, reports, pdfs and other documents should be kept in **`reports/`** directory.

(4) Modify README file to describe status of the project, also include any additional instructions (if any) required for simulation/testing your assignment.

If you need to re-upload your submission before the deadline, please contact a TA.
One submission per group. If the group accidentally submits twice, you must coordinate with the TAs.