University of Southern California

Cardinal NIC and Chip Multiprocessor
Project Phase 3

Objective:

The goal is for students to complete the design of a 16-node Cardinal Chip Multiprocessor, organized as a 4×4 mesh NoC. Each node in the system consists of one instance of the Cardinal Processor (developed in Part 2), one Cardinal Network Interface Component (NIC), and one Mesh Router. You will first design the NIC, which provides communication between the Processor and the NoC. Then, you will instantiate 16 identical nodes, each containing a Processor, NIC, and Router, and connect the Routers in a 4×4 mesh topology, where each Router communicates with its neighboring Routers in the North, South, East, and West directions when applicable.

The emphasis in this part of the project is on correct functionality, rather than performance or timing. Therefore, no delays should be used in the RTL design, since the design will be synthesized in later phases. However, delays may be used in the testbench to simulate the design at a chosen clock rate. To maintain consistency across all projects, simulations should assume a 4 ns clock period (250 MHz).

Team Policies:

Everyone should continue to work in their 2-student teams for the project.

Verilog Setup:

Continue to use the same Verilog simulation environment used in the previous project phases. Specifically, use the NC-Sim environment recommended in class.

Code that compiles in other tools but not in NC-Sim will be penalized equivalently to non-compiling code.

What to do?

- Download the Cardinal Network Interface Component (NIC) Architecture Specification from the course website.
- Write RTL Verilog code to implement all NIC functionality described in the specification. Use a top-level module name cardinal_nic. Your NIC module port names must match the input/output naming conventions shown in the specification (e.g., processor-side ports, router-side ports, handshake signals). Your NIC must be synthesizable to support later implementation phases.
- Write a testbench to verify your NIC. Pay close attention to handshake behavior and conditions under which the NIC must stall or block. Ensure the NIC can send and receive packets concurrently (one from the processor and one from the network) in the same clock cycle. You should also test virtual-channel or message-type selection as described in the specification.
- Simulate and validate your NIC using your testbench. Confirm your NIC behavior exactly matches the functional requirements before integrating it into the full system.
- Modify your Cardinal Processor (if needed) to correctly interact with the NIC for sending and receiving messages. Follow the communication protocol described in the specification.
- Integrate 16 Cardinal NICs, 16 Cardinal Processors, and 16 Routers into a 16-core multiprocessor system. The connection is given in Figure 1. First, connect the 16 Cardinal Routers to form a 4×4 mesh NoC. Each router should connect to its neighbors in the North, South, East, and West directions when applicable (border routers will naturally have fewer neighbors). Then, attach one Cardinal NIC and one Cardinal Processor to each Router. In total, this forms a 16-node multiprocessor system. Name your top-level module as

cardinal_cmp. Each processor continues to use the same private instruction and data memory modules from Part 2 (these serve as L1 instruction and L1 data memories). Follow the I/O interface naming conventions provided in the block diagrams in this specification to ensure compatibility with automated testbenches.

- Finally, write a testbench for the sixteen-core multiprocessor system. In the testbench, each processor sends one packet to each of the other fifteen processors (All-to-all communication pattern). The packets to be sent by each processor should be pre-loaded into its private data memory. Meanwhile, the instruction memory for each processor should be pre-loaded with code that: "Moves packet data from data memory to the register file", and "Sends the packet from the register file to the NIC whenever the output channel is available". Each receiving processor should store incoming packets into its data memory. At the end of the testbench, the contents of each processor's data memory should be checked to verify that packets were correctly delivered and handled according to the NIC and processor specifications.
  Use the same rules from Part 1 of the project to guarantee shortest-path, deadlock-free routing:
  - The shortest path route between any source and destination node should be determined ahead of time when the packets are loaded into the data memory. The direction, hop count, and vc fields of each packet must be set correctly before simulation.
  - The NIC must inject packets into the correct virtual channel as specified by the packet's vc field. Only one NIC design should be created and instantiated for all 16 processors. The NIC must decode the vc field from each packet and send it using the correct virtual channel on the network side. Do not create separate NICs for injecting even-channel packets and odd-channel packets. The same NIC must support both cases.

  The packet must be constructed correctly. The vc, direction, and hop count fields must be set according to the deterministic XY shortest-path routing rule used in the mesh (i.e., route in the X dimension first, then the Y dimension).The source field should encode the ID of the sending processor. Your design must not modify the reserved bits in the header or the payload content.
  To improve testing visibility, you may optionally add snooping logic between: the processor and the NIC, and the NIC and the router. The snooping logic records packets injected into the network and packets delivered to the processor. The correctness of system behavior can then be verified by inspecting these log files. This snooping mechanism is recommended for debugging, but not required for submission.

What to Submit?

Submit in the directory structure specified for your complete design including the top-level multiprocessor design, all submodules, the testbench, and testcases used for your verification. Please include sufficient comments in both your design and testbench code to justify that your testbenches properly verify the intended functionality. Also, ensure that your signal names match those defined in the architecture specification.
In addition, include a short report (no more than 3 pages) that lists all team members, describes how the work was divided, and explains the rationale behind your testbench strategy. If there are any known issues or limitations in your submission, please describe them in the report.
Submission Checklist:

- cardinal_nic.v (single NIC module)
- cardinal_mesh.v (4*4 Interconnect module)
- cardinal_router.v (single Router module)
- cardinal_cpu.v (single Processor module)

- cardinal_cmp.v (top-level 16-core multiprocessor module)
- Any supporting Verilog design modules
- tb_cardinal_cmp.v (testbench for the full 16-core system)
- Any supporting testbench modules, monitors, or snooping logic (optional but recommended)
- cmp_test.asm, the .asm source program used to generate the memory initialization files
- Instruction memory initialization files: cmp_test.imem.00.fill, cmp_test.imem.01.fill ... cmp_test.imem.33.fill (16 nodes)
- Data memory initialization files: cmp_test.dmem.00.fill, cmp_test.dmem.01.fill ... cmp_test.dmem.33.fill (16 nodes)
- Memory Dump Files (After Execution): cmp_test.dmem.00.dump, cmp_test.dmem.01.dump ... cmp_test.dmem.33.dump. Submit the contents of MEM[0]–MEM[31] only. Make sure your assembly code stores packet data only within MEM[0]–MEM[31].
- report.pdf

How to Submit?

"Tar" the directory which contains all the requested files, and use the following commands for electronic submission:

- cd to the directory which contains all the files requested and only the files requested.
- "Tar" the directory (the tar filename should include the names of both project partners):
  FirstName1_LastName1_FirstName2_LastName2
  Do NOT include other files such as waveform files and log files.
  > tar cvf tar_filename.tar . (← Don't forget "the dot" at the end.)

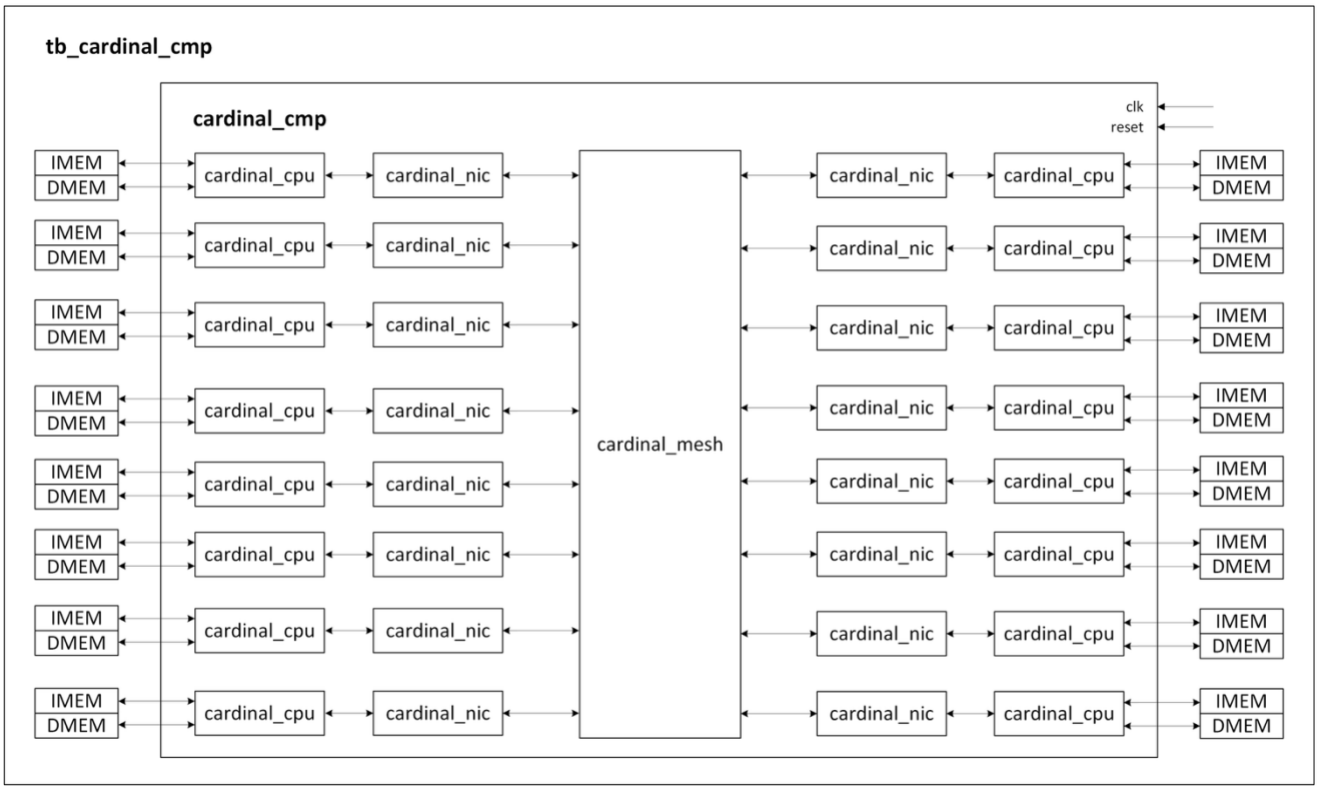
tb_cardinal_cmp
cardinal_cmp
clk
reset
IMEM
DMEM
cardinal_cpu
cardinal_nic
cardinal_mesh

Figure 1

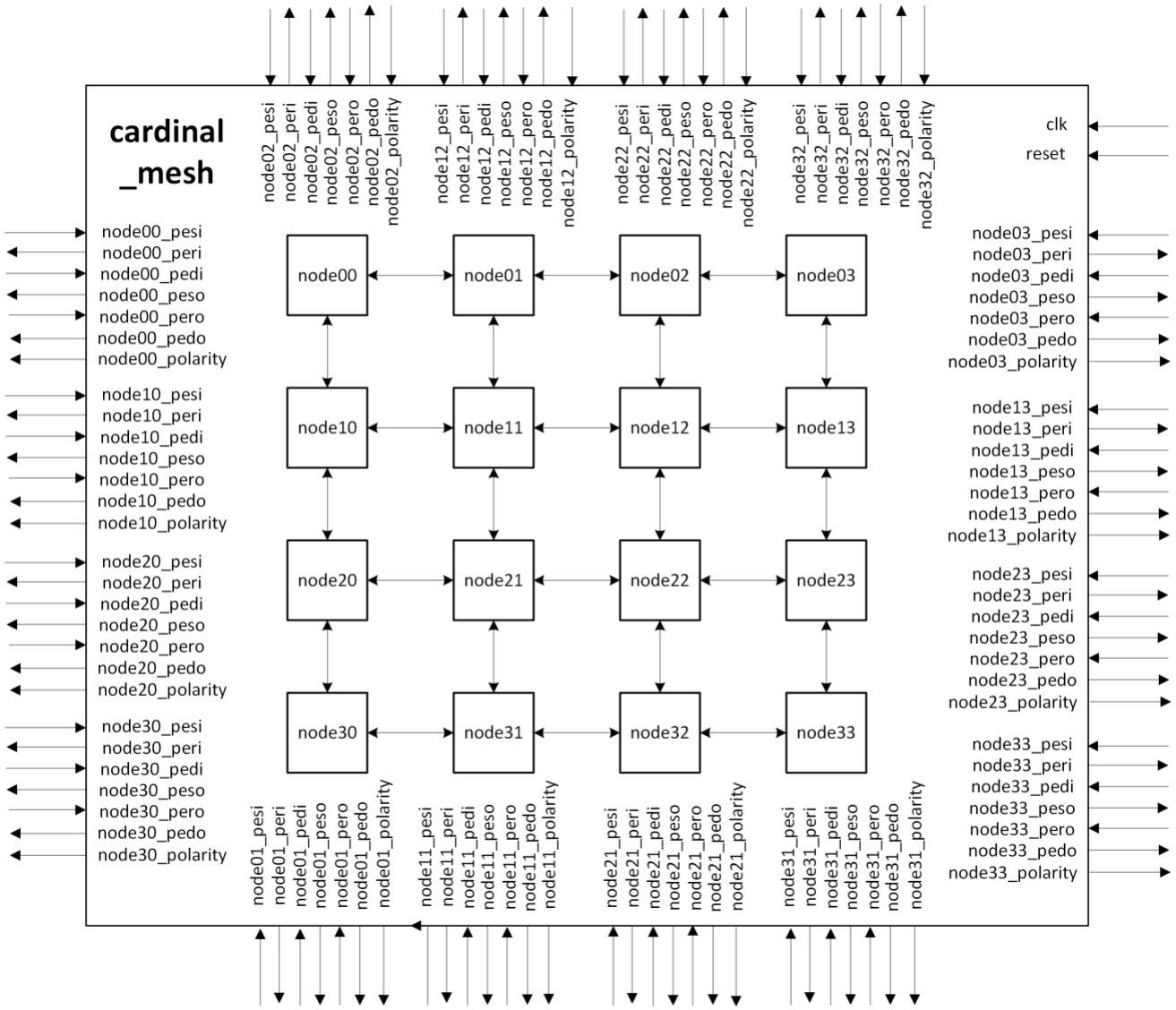

Figure 2

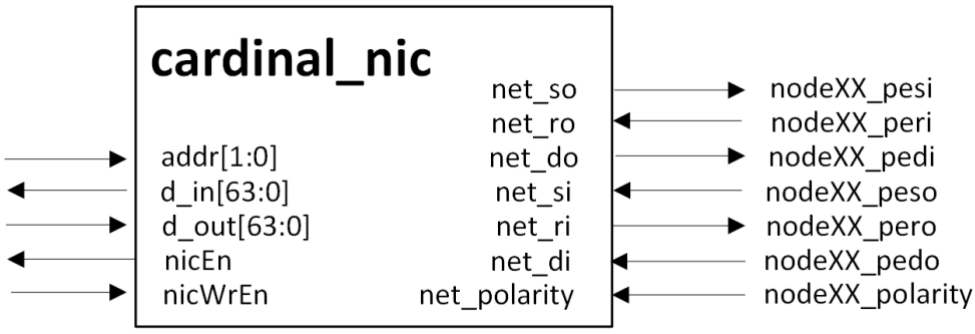

Figure 3