EE 552, Spring 2016 Class Project
Dr. Peter A. Beerel
Network-On-Chip w/ FEC
Rev. 1.0 (subject to update)
Last updated 3/11/2016

Network-On-Chip (NoC) with Forward Error Correction

The key idea of using NoC is to replace design-specific global on-chip wiring with a general-purpose on-chip interconnection network. A chip employing an on chip network is composed of a number of network clients (Nodes): processors, DSPs, memories, peripheral controllers, gateways to networks on other chips, and custom logic. Instead of connecting these top-level modules by routing dedicated wires, they are connected to a network that route packets between them. Sharing the wiring resources between many communication flows makes more efficient use of wires: when one client is idle, other clients continue to make use of the network resources.

Some important properties of a NoC are

1) **NoC topology:** NoC comprised of routers interconnected by point-to-point links. Network topology can vary depending on system needs, module sizes and placements. Some common topologies include tree, mesh and ring.
2) **NoC routing techniques**: Routing techniques can be classified into mainly two types, deterministic and adaptive. In deterministic routing the path from source to destination are fixed while adaptive routing allows the path to change depending upon the current state of load on the network
3) **NoC switching techniques**: Switching techniques can be classified into mainly two types, circuit switching and packet switching. In circuit switching entire path is set up and reserved before an actual data is sent, while in packet switching data is forwarded on per hop basis. Each packet contains data as well as control information.
4) **NoC flow control**: Flow control mainly addresses the issue of correct operation of the network. It determines how the networks' resources, such as channel bandwidth and buffer capacity, are allocated to packets traversing the network, ensuring that the network does not accept more packets than it can transmit successfully.
5) **Error detection/correction**: At the output of the network and the interface to the receiver, an error detecting circuit should verify if the data it received is error-free and either discard the packets that are corrupted or correct the error before passing the data.

Problem Definition:

The base-line project definition is to create an asynchronous NoC with a tree topology with 16 asynchronous blocks. The routing technique will be deterministic and the switching technique will be packet switching. You have to design the NoC using the

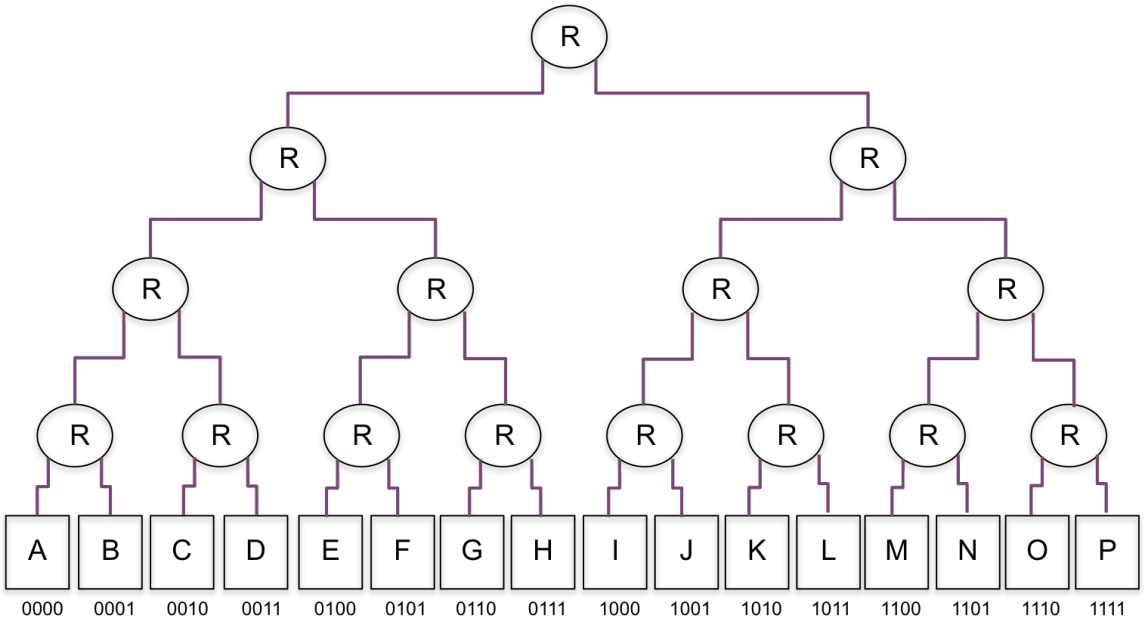
templates you have learned such that there is no data loss. You will also design an error detection circuit using Proteus. The base-line project is only acceptable for groups consisting of a single member. Groups consisting of two to four people will need to propose some improvements over the base-line project description in Stage-0 of the project.

Specification

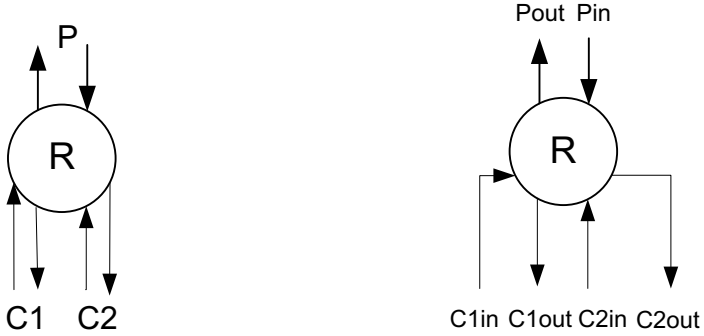
1) Base-line

Number of blocks: 16
Topology used: tree
Routing: deterministic
Switching technique: packet switching
Data : 4bits

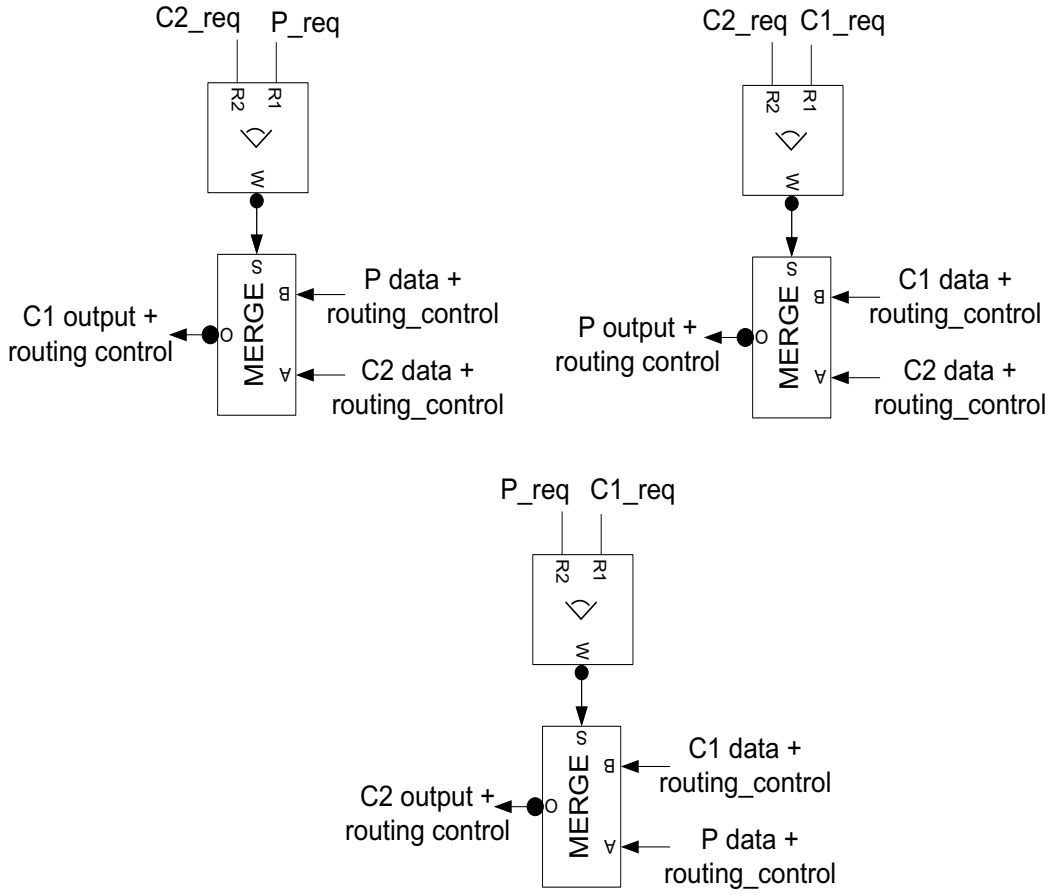
The probability of any block sending to any other block is equal so the topology will be a balanced tree. The figure below shows the topology of the base-line project. More complicated topologies, such as mesh and torus, might be considered as improvements on network topology



A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, and P are the 16 asynchronous blocks with addresses assigned below them. If we look at one particular instance of a router it will look like this:



It has three ports C1, C2, and P where each port is implemented using two asynchronous channels; one input and one output channel. Altogether, each router has 6 channels. The core of the micro-architecture is three two-way arbitrated merge units because each pair of ports must arbitrate access to the third. These are shown in the figure below. Note that the _req on the name of the input channels of the arbiters does not necessarily mean that the channels are bundled data. You can select encoding of the channels based on your implementation method (e.g. QDI, Micro-pipeline, …). The recommended method is PCHB as it is currently the best supported template in Proteus. Your design should implement appropriate logic for these _req channels based on the routing scheme of your choice.



Routing scheme:

Packet switching is used for the baseline design. Each packet has a 4-bit destination address in the header and a 4-bit data as the payload. The header should also have an error detection/correction field. You may add any other field(s) you deem necessary, but the size of a packet must be fixed.

The suggested baseline routing uses destination addresses to route the packets. Each router has an assigned, fixed address and mask. The address of the router is the common address bits of all its children nodes padded with zeros on the LSB to reach a 4-bit address. For example, the *address:mask* pair for the router connected to nodes C and D is 0010:1110. The common address bits of the children are 001, which is padded with a 0 on the LSB. The mask is set to 1110 because the 3MSB bits of the address of the children are common and equal to the three-bit MSB address of the router. As another example, the router at level 2, which has E, F, G, and H as child nodes, has *address:mask* equal to 0100:1100.

The routing algorithm works as follows: On router with *address:mask*

- On receiving any packet with address *dest* from a child node, if (*dest & mask*) equals to *address*, the packet is forwarded to the other child. Otherwise, it is sent to the parent router.
- On receiving a packet with address *dest* from the parent link, the router looks up the first non-masked bit of *dest*. If zero, the packet is forwarded to the left child; otherwise, the packet is sent to the right child.

Design Template (More specific requirement will be announced later)

- ❖ After the channel-based pipelined micro-architecture is completed, you must design/synthesize a subset of your blocks (at least the router) in gate-level Verilog.
- ❖ Teams must choose between a bundled-data or a 1-of-N template to make a portion of your abstract design concrete. You need not make every module concrete, but you do need to make at least one sub-block concrete.
- ❖ Bundled-data designs can be done with controllers designed with behavioral Verilog (baseline), Click Controllers, STGs based on Petrify, or using burst-mode designs using 3D or Minimalist. The latter two will be taught in the next couple of weeks.
- ❖ Forge might be used to synthesize bundled-data designs.
- ❖ Bundled-data design may be resiliency in which given the added complexity here behavioral controllers are more than sufficient.
- ❖ 1-of-N templates must be designed using gate-level PCHB/WCHB, or STFB templates. Proteus can be used to synthesize PCHB designs.

The Error Detector / Checker

- ❖ You can choose to encode your packets with any block code of your choice. At minimum, the simplest code will have an additional parity bit associated with each packet. This bit will allow you to detect if a single-bit error occurs, but not fix it.

Alternatively, you can design a Hamming-based block codes to not only detect 1-bit of error, but also correct it.

- You will design your testbench to generate packets with your code and the ability to randomly insert errors in such a way that you can verify the error detection/correction is working properly.
- You will implement a decoder of each packet in SystemVerilogCSP that detects whether or not the packet received has any errors and optionally fixes the errors. For simple parity-based code this can be as simple as a parity check tree (i.e., a tree of XOR gates). For one-person teams this is sufficient.
- For more complex Hamming codes combinational logic will hopefully be sufficient.
- For even more complex codes, you may need to add some form of memory. This memory should be described as a separate behavioral memory. And, to avoid needing a .lib of such a memory, you will need to partition your Verilog files such that the module you synthesize does not reference this memory. This step should be reserved to teams of 3 or more people who feel that they can handle Proteus.

Slack Elastic

- Robust asynchronous designs are ones in which channels can be pipelined late in the design cycle without causing deadlock given arbitrary delays in the system.
- All designs should be deadlock free.
- However, an optional feature of your design is to make it slack elastic, which means it would remain deadlock free if we added pipeline buffers arbitrarily to your design. This will be discussed in class.

Teams of 1 or 2 people should focus on the baseline problem. Larger teams can choose to try to take on some advanced features. In the baseline problem we assumed that chances of a block sending data to other 15 blocks are equally likely, but in the advanced features we will remove this assumption and motivate a design that uses a mixture of templates. We will give more details on this particular optional enhancement later. We will also post the measurement you need to make to analyze the performance of your design. Also the simulation details will be added.

Schedule & Important Dates

- **Stage 0: Form Teams**. You can form teams of up to 4 people. Not all team members are required to get the same grade. Smaller teams will be expected to have less ambitious design goals and will be graded accordingly. After, building your team, you need to submit a single-page proposal with details of your NoC (e. g. topology, routing scheme, and any other improvements), your chosen FEC coding, the template style and task partitioning between members of the group. Also specify which blocks of your design will be implemented to gate level in addition to the router, if any. We will approve your proposal within a week. **Your proposal should be about the right amount of work proportional to your group size to get approved.**
    - **Due-** 3/22/16 (Tuesday Midnight)

- **Stage 1: Micro-architecture specification and design**. The design should include the following sections.
  - **Micro-Architectural Design**. Design each micro-architectural block using SystemVerilogCSP.
  - **What to hand-in**
    - Hand-in SystemVerilogCSP modules of each block and block-level design of entire NoC.
    - Hand-in simulations showing correctness of each block. Identify test cases you used to make sure each block is functioning correctly in simulations.
    - Hand-in simulation of top-level, identifying why they demonstrate correctness of results.
    - Design and simulate a comprehensive set of test cases, including those that randomly back-pressure the design.

  Due – 4/12/16 (Tuesday Midnight)

- **Stage 2: Gate-level Design**
  - Design/synthesize a subset of pipeline stages at the gate-level
  - Verify functional correctness of entire system in which each pipeline stage is replaced with its gate-level model
  - Slack match the design to optimize throughput
  - Document results: The document should be the culmination, combination, and refinement of the stage 0 through 1 documents, along with the results and summaries of stage 2.
  - **Note:** the final report is due ~1 week after your presentation. Please incorporate feedback from your presentation into your final report (address bugs, clarify implementation details, suggest future work, etc).

  Due – 5/5/16 along with final report (Thursday Midnight)

- **Presentation**
  - A less than 12 slide power-point presentation overview should accompany your design. The presentation should review the architecture, micro-architecture, gate-level design, slack matching, other improvements (if any), and results. Slack matching and any other optimizations are not required to be complete by then.
    - The presentation will constitute 20% of the project grade.

  Due data – To be announced

(Some) Suggested Additional Features

1. In the base-line project description we assumed that chances of sender sending to receivers are equally likely, hence the routers as well as links are symmetric. So one possible feature you can add is to remove this restriction. For example:

A will send to B 50% of the time, A will send to C 15% of the time, to D 15% of time, E – 8%, F – 8% G- 7% and H – 7%.

B–A 50%, B-C 15% B-D 15%, B-E 8%, B-F 8%, B-G 7% B-H 7%

C-A 15% C-B 15% C-D 50% C-E 8% C-F 8% C-G 7% C-H 7%

D-A 15% D-B 15% D-C 50% D-E 8% D-F 8% D-G 7% D-H 7%

E-A 8% E-B 8% E-C 7% E-D 7% E-F 50% E-G 15% E-H 15%

F-A 8% F-B 8% F-C 7% F-D 7% F-E 50% F-G 15% F-H 15%

G-A 8% G-B 8% G-C 7% G-D 7% G-E 15% G-F 15% G-H 50%

H-A 8% H-B 8% H-C 7% H-D 7% H-E 15% H-F 15% H-G 50%

Looking at this particular traffic scenario, one will want links which connect A and B to have more bandwidth then links connecting A and H. So you can use your experience with several design styles that you learned in this class to optimize your network on chip by using different routers and link designs to improve the performance. Here you can assume that bandwidth $\propto$ 1/Cycletime.

2. In the base line we assumed the topology to be a balanced tree. Imagine this scenario:
A-B 40%, A-C 40%, A-D 5% A-E 5% A-F 4% A-G 4% A-H 2%
and

C-A 40%, C-B 40%, C-D 5% C-E 5% C-F 4% C-G 4% C-H 2%

The remaining traffic requirements are the same as in case 1. Can you *change the network topology* (i.e., consider using an unbalanced tree of routers OR increasing the number ports a router can handle) to improve the worst case latency of packets traveling from A to C? How will it impact the network congestion at other nodes?

3. Synchronous interfaces: Ideally the network on chip will support not only blocks that communicate asynchronously but have converters (e.g., synchronous to asynchronous and asynchronous to synchronous) to allow the sources and destinations to be synchronous.

4. Supporting virtual channels is something important in many NoC. Figuring out how to do that asynchronously would be very interesting!

5. Supporting some form of adaptive routing would also be interesting.

Other improvements are encouraged! Improved address schemes, error correction, automatic resending of error packets, and circuit switching routing are just some ideas.