Reconfigurable Ouroboros ONoC - System   
Design and Simulation

The structure of an N-node reconfigurable Ouroboros network is shown in Fig. 1.The Ouroboros Network (ON) [1] is an torus-based fully reconfiguration architecture. An atomic ON consists of a Head Core (HC) and a number of Body Cores (BC). The HC works as the administrator in managing all the inter-node traffic flows. Every BC connects to the HC through electronic control link. The HC and each BC are connected to a ring-shaped waveguide through an MRR-based bi-directional I/O unit. Then the waveguide ring turns to be a bi-directional bus for all data communications between any pair of the cores. Every core on the bus can be engaged or bypassed. On this optical bus, data can be transferred from one node to any other node as long as all the intermediate nodes between them are bypassed. Coordinated and controlled by the HC, multiple communication sessions among BCs will be initialized/terminated in parallel to maximize the use of the network infrastructure.



1. Reconfigurable Ouroboros Network Overview
2. Routing Scheme
3. Hardware Configuration

The Head Core is physically connected to every Body Core through a bidirectional exclusive optoelectronic channel. On the HC side, each channel is connected to a small buffer. All buffers are connected to the HC via an *N*-to-1 MUX. On the HC side, all received messages will be store in the channel buffer temporarily, then the MUX will forward them one-by-one to the HC according to their arriving sequence. The HC will process every message to update network routing and reconfiguration. On the BC side, it will received the command from HC, then engage itself onto to the data bus and start the data transmission.



1. Routing Control
2. Communication Scheme

# Request Stage

In this stage, the Body Core (BC) will send the communication request message to the Head Core (HC) through the control link. The request contains the following information: destination body node address, data size, and QoS info (optional).

For the simplicity, assume the network maximally has 16 cores (can be extended hierarchally in future), hence the address is 4-bit; assume only 4 data types, hence 2-bit for identify the data size; and optional information (QoS, etc.) is 2-bit. Then totally the request message can be contained in a byte.

# Channel Scheduling Stage

The first step is the channel assignment scheduling stage. The HC holds a dynamic channel assignment list. In this stage the HC will check in every clock cycle to see if any new communication requests are received from any BCs, then add a new schedule on the bottom of the list to handle these requests later. Simultaneously in the pipelined channel assignment stage the top line will be processed and removed from the list.

# Basic Sequencing Rules

The key of scheduling is to deal with dependencies correctly, which is to maintain the original sequence in processing requests. Here are basic rules to follow:

1. If two requests are from the same source core, the sequence must be maintained when these requests are scheduled and implemented.
2. Multiple requests may be generated from different source cores to communicate to the same destination core. If they are generated on different on different clock cycle (they are in sequence), then the sequence must be maintained when these requests are scheduled and implemented by HC.
3. If multiple requests are generated on the same clock cycle (no sequence) from different source cores to communicate to the same destination core, unless other mechanism (cache/memory controller, etc.) specifies their sequence, we assume they have no dependency among each other, and can be handled in any sequence.
4. If two requests are from different source cores to different destination core, we assume they and not dependent to each other and can be handled in any sequence.

# Handling Simultaneous Channel Requests

There are two basic rules on the sequencing:

When more than one communication requests arrived in the same clock cycle, no matter if they are requesting same channel(s) or not, according to rule (c) we assume there is no dependency among them. But the HC must make decision which request should be processed first. Here are basic rules to follow:

* If requests are in different QoS levels, process accordingly.
* When QoS is missing, process them by following one of these basic policies: random, Round-Robbin, best fit, etc.
* Furthermore, one advanced policies called Big-One-First can be applied. In this policy, the request that asks for more channels is in high priority to be processed. Because such requests are not easy to get satisfied, this policy can help to improve the efficiency and network throughput.

# Channel Scheduling Policy

After all requests are sequenced properly, next step is process request one by one sequentially and scheduling channels for them. Each schedule will be added in to the channel assignment list.

There are three basic rules, Isolation, Tetris and Wormhole to follow in channel scheduling.

* Isolation Rule

Isolation rule is, if two requests share either same source node or destination node, then the later one can start only after the first one is finished. An example is shown in Fig. 3, the request R7 can start on *t*3, but not on *t*2.

Discussion: Can a core transmit and receive simultaneously?

* Tetris Rule

The Tetris rule is, a request will be assigned to start from the earliest moment as long as it satisfies the Isolation Rule. It just likes in the Tetris game, a block will drop down straightly until any part of it reaches the bottom. An example is shown in Fig. 3, the request R6 can start on *t*4.

* Wormhole Rule

The Wormhole rule is, a request can cross existed channel schedule session to fit into a hole, as long as it shares neither source node nor destination node with any of those schedules it crossed. An example is shown in Fig. 3, the request R5 can cross two requests R1 and R3, since R5 (Cr0 - Cr13) shares neither source node nor destination node with R1 (Cr1 - Cr10) or R3 (Cr14 - Cr15).



1. Channel Assignment List

# Channel Assignment Stage

# New Assignment Triggering

The next stage is the channel assignment stage. In every clock cycle, the Head Core will check to see if any communication has finished. If yes, it will check the channel assignment list to see if the condition to implement a new communication is satisfied. If yes, the new communication will get channels assigned. Otherwise, the HC will wait until the next update. An example can be seen in Fig. 3. When R5 is done, the HC will check the top of the assignment list, and see nothing to do, then it will wait. On time *t*1, when R5 is done, the HC will check again and start R1 and R2.

The channel assignment table will be updated dynamically to remove completed requests.

# Implementation of Channel Assignment

The physical process of channel assignment can be seen from Fig. 4.



1. Transmission Process.

Besides the control link between HC and each BC which BC uses to send request to HC, there are two directional single-bit links between them, which is shown in Fig. 4. For *N* BCs, the HC is going to operate two *N*-bit binary vectors, one for input and one for output, to monitor and control the engagement/bypass of each BC to/from the data bus. It will scan in every clock cycle the input vector, update information, make decision and operate the output vector accordingly.

1. Simulation Configuration
2. Basic Parameters:

* Network Size: 16 body nodes
* Optical Frequency: 40GHz
* Electrical Frequency: 2GHz
* WDM: 4
* Request: 1 byte
* Packet: 4 types: 1 word (32-bit), 2 words, 8 words and 32 words.
* Request Buffer Size: 8 (bytes)
* Request Buffering: 1 electrical clock cycle
* Request MUX: 1 electrical clock cycle
* Request Sequencing: 1 electrical clock cycle
* Request Scheduling: 1 electrical clock cycle
* Transmission Time: (Fig. 4) 4 electrical clock cycle + data transmission time
* Data Transmission: Data/(Optical Frequency\*WDM)

Reference

[1] L. Zhang, X. Tan, M. Yang, and Y. Jiang, "A centralized optical network-on-chip architecture with Space-Division Multiple Access," in *Optical Interconnects Conference, 2014 IEEE*, 2014, pp. 43-44.