

Designing and Implementing a Simple Router for Multiprocessor Systems

CMP 301B Project

Computer Engineering Department
Faculty of Engineering, Cairo University
April 26, 2012

Contents

1	Introduction	2
2	Project objective	3
3	Project description	3
3.1	Input Buffer	4
3.2	Switch Fabric	4
3.3	Output Queue	5
3.4	Output Buffer	5
3.5	Controller	6
4	Project tasks	7
4.1	Task No. DT-ROU-01	8
4.2	Task No. DT-ROU-02	9
4.3	Task No. DT-ROU-03	10
4.4	Task No. DT-ROU-04	11
4.5	Task No. DT-ROU-05	12
4.6	Task No. DT-ROU-06	13
4.7	Task No. DT-ROU-07	14
5	Evaluation Scheme	15
5.1	General Notes:	15

1 Introduction

A router is a networking module that buffers and forwards data (in packets format) across a network toward their destinations, through a process known as routing. Routers use headers and forwarding tables to determine the best path for forwarding the packets according to the design requirements, and they use communication protocols to communicate with each other and configure the best route between any two nodes. Routers are not only used in computer networks applications, but also have been integrated in System-on-Chip (SoC) and multiprocessor based designs to form what is known as Networks-on-Chip (NoC) applications. With modern designs that have many processors and hundreds of other Intellectual Property (IP) cores¹, interconnection using a single shared bus is not sufficient anymore. The NoC is a new paradigm that provides an integrated solution for achieving efficient inter-module communication [1]. Unlike computer networks, NoC has shorter communication delay and limited silicon area which put more limitations on the NoC-based system design and require different approaches to overcome design challenges specially in router design [2]. A router block diagram is shown in Figure 1. Once packets arrive at the input

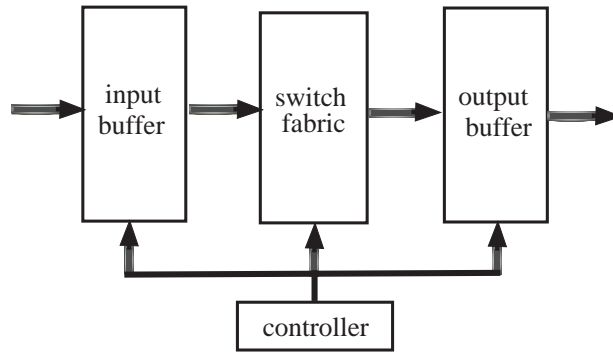


Figure 1: Router block diagram.

of the router, they are buffered at the input buffer, then the controller reads the packet header and configures the switch fabric to direct the packet to the appropriate output buffer. The output buffer is a group of FIFOs designed to handle multiple access requests at the output ports. Routing tables and timing synchronization are done in the controller part. In this project, it is required to design and implement a simple router using VHDL language. The following sections explain in details the project objective, description, tasks, and evaluation method.

¹An IP core is a block of logic or data that is used in making a field programmable gate array (FPGA) or application-specific integrated circuit (ASIC) for a product.

2 Project objective

The aim of this project is to design, implement and evaluate the performance of a simple router. The project main deliverables are a synthesisable VHDL code for the router and a test bench to validate the correctness of the design.

3 Project description

Routers can be classified according to two different criteria [3]: the type of the switch fabric (SF) and the location of buffers and queues within the router. In this section, the functional description of one router type is explained. The output-queuing router with a point to point switch fabric is chosen as an example for simplicity. Digital systems are usually divided into datapath and controller.

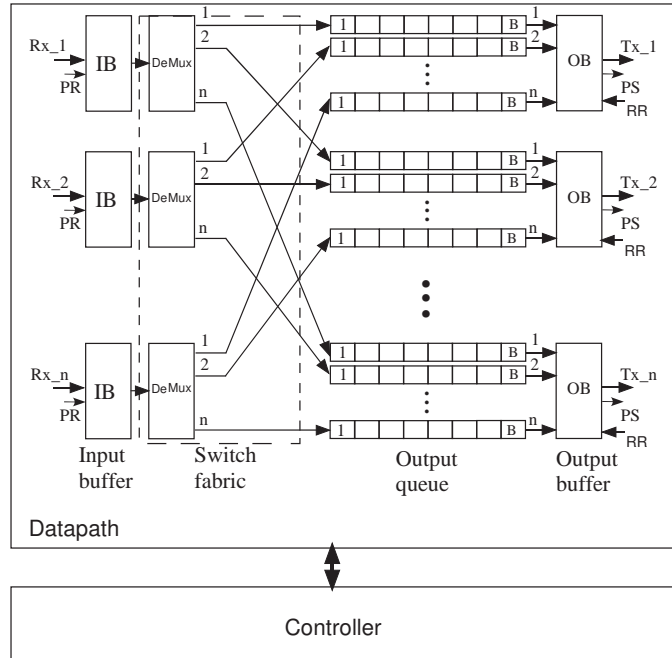


Figure 2: Output queuing router architecture.

The datapath consists of modules that can be implemented using combinational logic, while the controller is usually implemented using finite state machine (FSM). Figure 2 shows the architecture of an n -ports output queuing router. Packets arrive at the input of the router asynchronously with a packet ready signal indicator (PR). The controller reads the packet header, then configures the switch fabric to direct the packet to the corresponding queue. There are n queues for each output port. These queues serve as FIFO buffers that are synchronized with both clock edges. Finally, packets are served in a random

service probability depending on the target destination availability. A round robin scheduler algorithm serves backlogged queues one after another in a fixed order at the output. An active high Receive Ready (RR) signal indicates the target destination availability. Packets are sent to the next hop with Packet Sent (PS) signal for synchronization purposes. The packet format, for an 8-port router, is shown in Figure 3. The first three bits represents the destination port number, the following four bits contain the sent data, and the last bit is reserved for parity check.

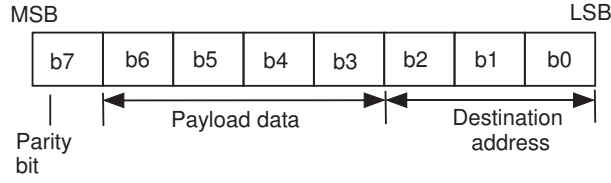


Figure 3: Packet format.

As shown in Figure 2, a simple router datapath consists of four main modules: input buffer, switch fabric, output queue, and output buffer. In the following subsections, we explore various design options for each internal module.

3.1 Input Buffer

The implementation of the input buffer can be done in many different ways. The input buffer simply is a register consists of 8 flip-flops. The main function of the input buffer is to store the packet once it arrived. It's for the designer to decide how will this module synchronize with the output world. For example, designers can add extra modules to synchronize with the incoming bits or extract the clock from the incoming packet.

3.2 Switch Fabric

The switch fabric (SF) unit is responsible of providing full connectivity between the inputs and outputs of the switch with reasonable hardware and delay [3]. In that sense, the input and output ports share the SF as a common communication resource, and, as such, conflicts may arise that have to be resolved using some contention-resolving technique or arbitration protocol. Reference [4] reviews and models five protocols and explains how they can be implemented in hardware.

A simple SF can be implemented as a set of demultiplexers as shown in Figure 2. These demultiplexers are controlled from the controller module. The controller module reads the first 3 bits in the incoming packet, then set the connection to output queue according to destination address.

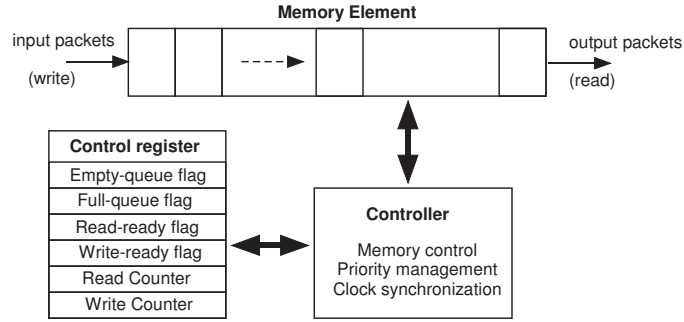


Figure 4: FIFO building blocks

3.3 Output Queue

Queues are usually implemented as First-in/First-out (FIFO) buffers. FIFO implementation has many design options and varieties. Starting from synchronous or asynchronous design, using one-port memory or dual-port memory, whether all arrived packets are served with equal priority or there is an arbitration rule, to the type of scheduler at the output (i.e., static priority, round robin, etc.)

As shown in Fig. 4, the main building blocks of a buffer are: memory element, control register, and controller. The memory unit consists of a set of words. Number of these words (buffer length) is defined by designers to match target traffic characteristics. The size of each word equals the packet width (assuming fixed packet size). Inaccurate estimation of the buffer length may cause poor system performance if the burstiness ratio exceeds the buffer capacity. On the other hand, if the buffer length is much greater than the actual number of bursty packets, excessive use of the silicon area will occur. Therefore, the modeling process is very important for choosing the appropriate buffer length.

The controller mainly manages the data flow inside the buffer to serve as a FIFO, in this experiment, and synchronize the internal system clock with the external interfaces. The control register consists of a set of flags to indicate the status of the FIFO and pointers (counters) to write/read to/from any memory address.

There are many proposed IP cores for various FIFO designs. [5] However, for NoC-based system, many design parameters must be considered such as synchronization problem, complexity of the controller, and the area constraints. It is for designers to choose the appropriate design that matches target application constraints and requirements.

3.4 Output Buffer

Just like the input buffer, the output buffer is a register consists of 8 flip-flops. The main function of the output buffer is to pick the packet from the output queue and deliver it to the output port according to the chosen scheduler

algorithm. Again, it's for designers to decide which scheduler technique they can use (round robin, processor sharing, etc.).

3.5 Controller

The controller module is designed to:

1. Synchronize router internal modules with the incoming packets.
2. Read packet header and configure the switch fabric.
3. Control data flow inside the router and apply round-robin scheduler at the output ports.

4 Project tasks

The main task is to design a 4-port router that could be used in a multiprocessor Network on Chips (NoC). To achieve this goal, you might need some or all of the following subtasks

4.1 Task No. DT-ROU-01

Objective: To write VHDL code for 8-bit Register with Positive-Edge Clock, Asynchronous Reset, and Clock Enable.

The VHDL code should follow pin definitions and design parameters mentioned in Table 1.

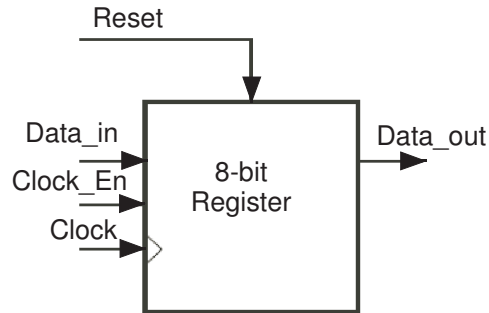


Figure 5: Input buffer.

Pin Description:

Table 1: Input buffer

IO pin	Description
<i>Data_in</i> [7:0]	Data Input
Clock	Positive-Edge Clock
<i>Data_out</i> [7:0]	Data Output
<i>Clock_En</i>	Clock Enable (active High)
Reset	Asynchronous Reset (active High)

1. Data is transferred from input port to output port at the clock positive edge if and only if $Clock_En = 1$ and $Reset = 0$.
2. If $Reset = 1$, then output port is set to "00000000".
3. If $Clock_En = 0$, then output port keeps its current value.

The delivery has to contain:

Synthesizable VHDL Code, total equivalent gate count for the design, and screen shot from Modelsim wave window.

Deadline: Last tutorial of the semester

4.2 Task No. DT-ROU-02

Objective: To write VHDL code for 1-to-4 8-bit DeMUX.

The VHDL code should follow pin definitions and design parameters mentioned in Table 2.

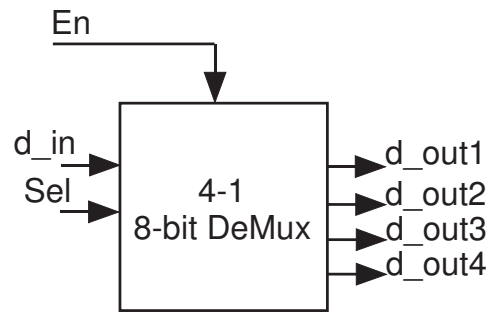


Figure 6: Demultiplexer.

Pin Description:

Table 2: Demultiplexer

IO pin	Description
<i>d_in</i> [7:0]	Data Input
<i>d_out1</i> [7:0]	Data Output - port 1 ; <i>d_out1</i> = <i>d_in</i> ONLY when <i>Sel</i> ="00"
<i>d_out2</i> [7:0]	Data Output - Port 2 ; <i>d_out2</i> = <i>d_in</i> ONLY when <i>Sel</i> ="01"
<i>d_out3</i> [7:0]	Data Output - Port 3 ; <i>d_out3</i> = <i>d_in</i> ONLY when <i>Sel</i> ="10"
<i>d_out4</i> [7:0]	Data Output - port 4 ; <i>d_out4</i> = <i>d_in</i> ONLY when <i>Sel</i> ="11"
<i>Sel</i> [1:0]	Selector
<i>En</i>	Enable

1. Data is transferred from input port to the appropriate output port according to selector value if $En = 1$.
2. If $En = 0$, then, all output ports keep their current value.

The delivery has to contain:

Synthesizable VHDL Code, total equivalent gate count for the design, and screen shot from Modelsim wave window.

Deadline: Last tutorial of the semester

4.3 Task No. DT-ROU-03

Objective: To write VHDL code for Dual-Port Block RAM with Different Clocks.

The VHDL code should follow pin definitions and design parameters mentioned in Table 3.

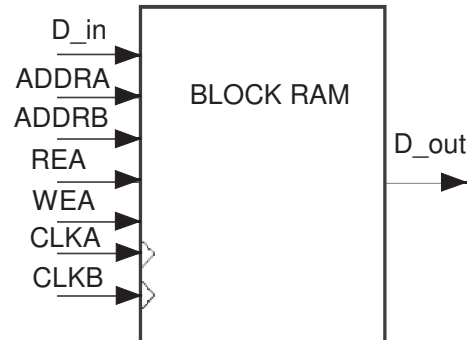


Figure 7: Dual-port RAM.

Pin Description:

Table 3: Dual-port RAM

IO pin	Description
<i>d_in</i> [7:0]	Data Input
<i>d_out</i> [7:0]	Data Output
WEA	Write enable (active high)
REA	Read enable (active high)
ADDRA [2:0]	Write port (A) address bus
ADDRB [2:0]	Read port (B) address bus
CLKA	Clock signal for port A
CLKB	Clock signal for port B

1. Port A (write) is synchronized with CLKA and port B (read) is synchronized with CLKB.

The delivery has to contain:

Synthesizable VHDL Code, total equivalent gate count for the design, and screen shot from Modelsim wave window.

Deadline: Last tutorial of the semester

4.4 Task No. DT-ROU-04

Objective: To write VHDL code for a FIFO controller.

The VHDL code should follow pin definitions and design parameters mentioned in Table 4.

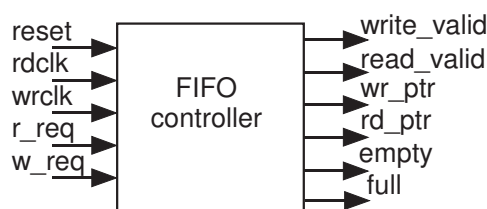


Figure 8: FIFO controller.

Pin Description:

Table 4: FIFO controller

IO pin	Description
reset	reset signal (active high)
rdclk	input read clock
wrclk	input write clock
<i>r_req</i>	request read signal
<i>w_req</i>	request write signal
<i>write_valid</i>	valid write output indication
<i>read_valid</i>	valid read output indication
<i>wr_ptr</i> [3 : 0]	write address
<i>rd_ptr</i> [3 : 0]	read address
empty	FIFO empty flag
full	FIFO full flag

FIFO controller receives read request and write request from *r_req* and *w_req* signals respectively. Then, it checks the validity of read and write operation and generates valid-indication-signal on *read_valid* and *write_valid* ports respectively. Finally, controller outputs the corresponding read and write addresses. When FIFO is full, write operation is disabled, and when it is empty, read operation is disabled. FIFO controller sets the *empty* output port to high when the FIFO is empty and sets *full* output port to high when FIFO is full.

The delivery has to contain:

Synthesizable VHDL Code, total equivalent gate count for the design, and screen shot from Modelsim wave window.

Deadline: Last tutorial of the semester

4.5 Task No. DT-ROU-05

Objective: To write VHDL code for a FIFO

The VHDL code should follow pin definitions and design parameters mentioned in Table 5.

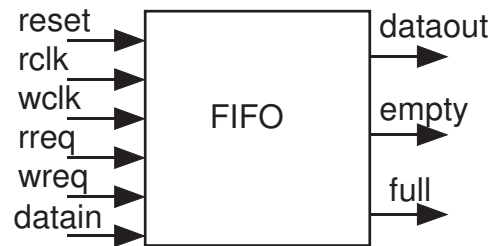


Figure 9: FIFO.

Pin Description:

Table 5: FIFO controller

IO pin	Description
reset	reset signal (active high)
rclk	input read clock
wclk	input write clock
rreq	request read signal
wreq	request write signal
datain [7:0]	input data to FIFO
dataout [7:0]	output data from FIFO
empty	FIFO empty flag
full	FIFO full flag

FIFO receives read request and write request from *rreq* and *wreq* signals respectively. When FIFO is full, write operation is disabled, and when it is empty, read operation is disabled. FIFO *empty* output port is set to high when the FIFO is empty and *full* output port is set to high when FIFO is full.

The delivery has to contain:

Synthesizable VHDL Code, total equivalent gate count for the design, and screen shot from Modelsim wave window.

Deadline: Last tutorial of the semester

4.6 Task No. DT-ROU-06

Objective: To write VHDL code for a Round Robin Scheduler

The VHDL code should follow pin definitions and design parameters mentioned in Table 6.

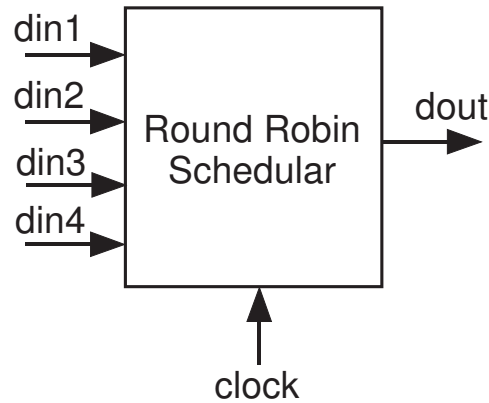


Figure 10: Round robin Scheduler.

Pin Description:

Table 6: Round robin Scheduler

IO pin	Description
clock	module clock (rising edge)
din1 [7:0]	input data port 1
din2 [7:0]	input data port 2
din3 [7:0]	input data port 3
din4 [7:0]	input data port 4
dout [7:0]	output data

Round-robin scheduling is a scheduling algorithm which repeatedly runs through a list of users. In this module, we have 4 input ports and one output port. A round robin scheduler algorithm serves backlogged ports one after another in a fixed order at the output. That means in the first clock cycle, data from input port 1 is transferred to the output port. The next clock cycle port 2 is served and so on.

The delivery has to contain:

Synthesizable VHDL Code, total equivalent gate count for the design, and screen shot from Modelsim wave window.

Deadline: Last tutorial of the semester

4.7 Task No. DT-ROU-07

Objective: To write VHDL code for a 4-port router

The VHDL code should follow pin definitions and design parameters mentioned in Table 7.

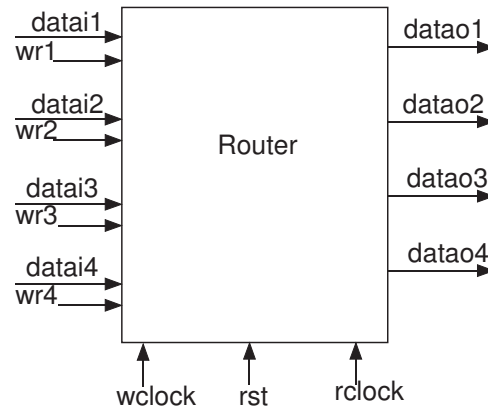


Figure 11: Round robin Scheduler.

Pin Description:

Table 7: Round robin Scheduler

IO pin	Description
datai1,datai2,datai3,datai4 [7:0]	input ports
wr1, wr2, wr3, wr4	packet available indicator (active high)
datao1,datao2,datao3,datao4 [7:0]	output ports
wclock	arrival clock for input ports synchronization
rclock	service clock for output ports synchronization
rst	reset signal

Data arrived at the input port in a fixed packet length format (8 bits). The first two bits are the output port address. The router is to transfer input packets to the appropriate output ports. The router input ports are synchronized with an input clock source (wclock) whereas the output ports are synchronized with the scheduler clock source (rclock).

The delivery has to contain:

Synthesizable VHDL Code, total equivalent gate count for the design, and screen shot from Modelsim wave window.

Deadline: Last tutorial of the semester

5 Evaluation Scheme

Each team shall consist of a maximum of four members, and no less than three. Each team should submit their VHDL files and demo their design orally. Each project will be evaluated according to the number of tasks that are implemented according to the following distribution

Task No. DT-ROU-01: 1 mark

Task No. DT-ROU-02: 1 mark

Task No. DT-ROU-03: 2 marks

Task No. DT-ROU-04: 2 marks

Task No. DT-ROU-05: 2 marks

Task No. DT-ROU-06: 2 marks

Task No. DT-ROU-07: 10 marks

5.1 General Notes:

- You MUST prepare a waveform with the main signals showing that your router is working correctly.
- Failing to have a synthesizable VHDL code will result in deducting two marks.
- At the demo day, prepare a table with the gate count of all the modules in your design and the router itself. The four groups, with a working router, that have the minimum number of gates will be rewarded with bonus 2, 1.5, 1, and 0.5 marks, respectively.

References

- [1] L. Benini and G. D. Micheli, “Networks on chips: A new SoC paradigm,” *IEEE Computer Magazine*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
- [2] A. Ivanov and G. De Micheli, “Guest editors’ introduction: The network-on-chip paradigm in practice and research,” *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 399–403, Sep. 2005.
- [3] F. Gebali, *Computer Communications networks: Analysis and Design*, 3rd ed. Victoria, B.C., Canada: Northstar Digital Design, inc., 2005.
- [4] F. Elguibaly, “Analysis and design of arbitration protocols,” *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 1168–1175, 1989.
- [5] LogicCore. (2004, Nov. 11,) Xilinx Technical Document, Asynchronous FIFO v6.1. Product Specification DS232. [Online]. Available: http://www.xilinx.com/ipcenter/catalog/logicore/docs/async_fifo.pdf