Course Project

Part 1

Design of a Matrix Arbiter for a NoC router

1 Background

The aim of the course project is to design a router for ***Network on Chip (NoC)*** applications. The canonical architecture of the router is shown in Fig. 1. The number of ports in the router varies with the interconnect architecture. However, the basic architecture of a router port remains the same.

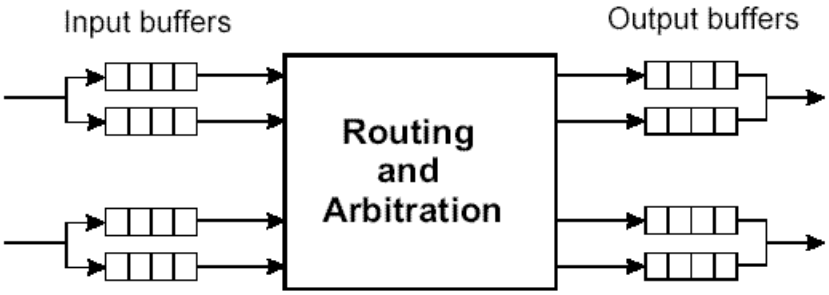

Fig.1: Virtual-channel router

The different components of a router port are shown in Fig. 2. It mainly consists of input/output FIFO buffers, input/output arbiters, MUX and DEMUX units, and a routing block. In order to have a considerably high throughput, we use a virtual channel router, where each port of the router has multiple parallel buffers.

Each physical input port has more than one virtual channel, uniquely identified by the virtual channel identifier (VCID). Flits may simultaneously arrive at more than one virtual channel. As a result, an arbitration mechanism is necessary to allow only one virtual channel to access a single physical port. Let there be *m* virtual channels corresponding to each input port; we need an *m: 1* arbiter at the input. As an example as shown in Fig. 2, *m* is equal to 4. Similarly, flits from more than one input port may simultaneously try to access a particular output port. If *k* is the number of ports in a router, then we need a *(k-1)*:1 arbiter at each output port. The routing logic block determines the output port to be taken by an incoming flit.

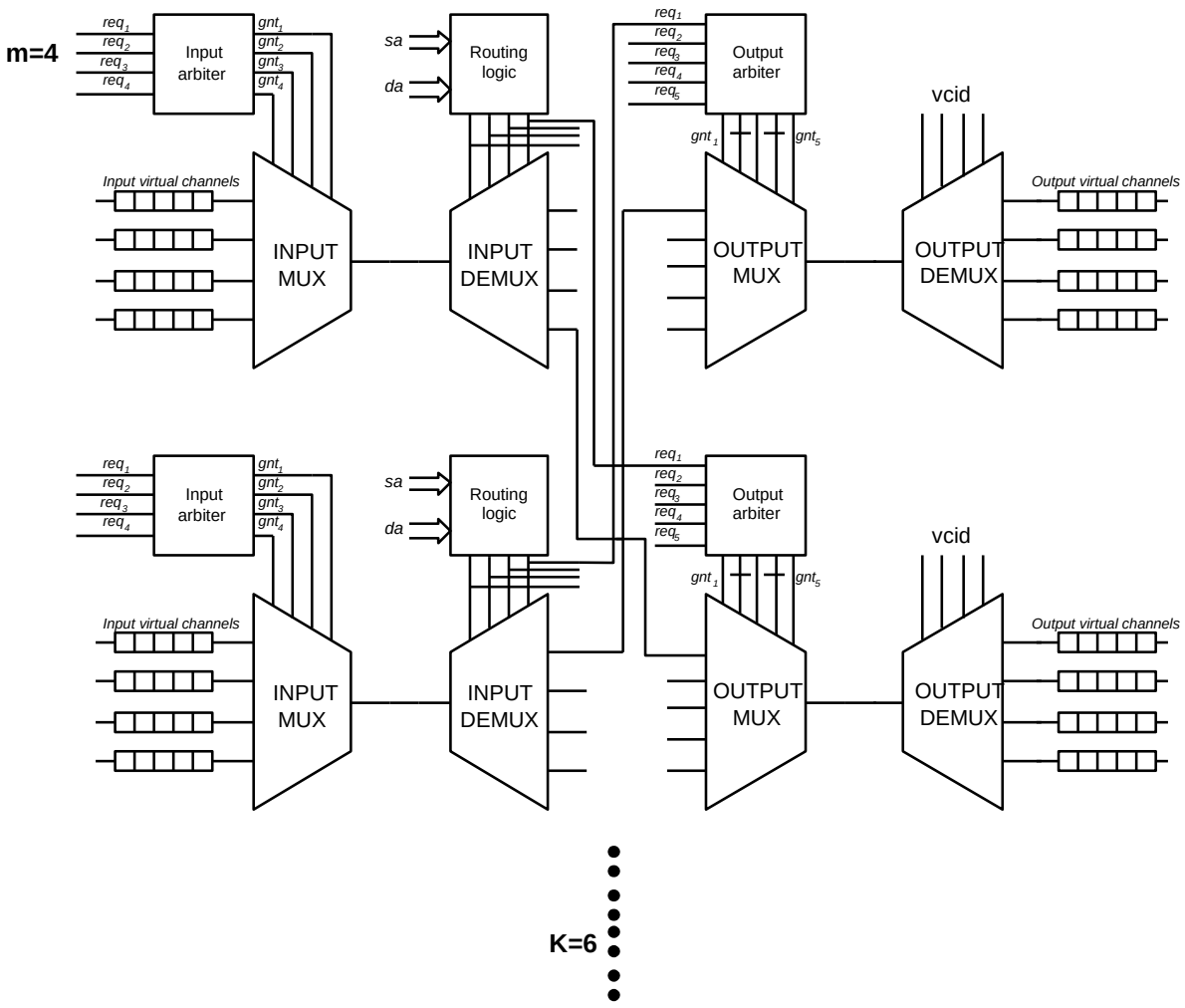

Fig.2: Block diagram of a router

In the first part of the project you are to design the input arbiter for the router. There are different possible arbitration mechanisms, like round-robin, queuing, time division multiple access (TDMA), and matrix arbitration. In this project you will design a ***matrix arbiter.*** The matrix arbiter stores priorities between ***n*** requestors in a binary ***n-by-n*** matrix. The structure of the matrix in case of four requesters is shown in Fig. 3. The priority of a requester with respect to itself does not have any physical significance and hence the elements along the main diagonal in the priority matrix are void and denoted by X.

Each matrix element *[i, j]* records the binary priority between each pair of inputs. For example, suppose requester *i* has a higher priority than requester *j*, then the matrix

element *[i, j]* will be set to 1, while the corresponding matrix element *[j, i]* will be 0. A requester will be granted the resource if no other higher priority requester is bidding for the same resource. Once a requester succeeds in being granted a resource, its priority is updated and set to be the lowest among all requesters. Once a requestor succeeds in arbitration, its priority is set to be the lowest among all requestors. Hence, when requestor ***i*** is granted the resource, its priority is set to be the lowest among all requestors by setting ***[i, *]*** to 0 (clearing row ***i***, so that requestor I has lower priority than all other requestors) and ***[*, i]*** to 1 (setting column ***i***, so that other requestors have higher priority over requestor ***i***).

$$\begin{bmatrix} X & p_{12} & p_{13} & p_{14} \\ p_{21} & X & p_{23} & p_{24} \\ p_{31} & p_{32} & X & p_{34} \\ p_{41} & p_{42} & p_{43} & X \end{bmatrix}$$

Fig. 3: Priority matrix

As an example, consider that the status of the priority matrix is as shown in left matrix in Fig. 4 and requestor *2* is granted the resource. Then after arbitration, column *2* is set to 1 and row *2* is set to 0, such that requestor *2* has the lowest priority with respect to all other requestors.

$$\begin{bmatrix} X & 0 & 0 & 1 \\ X & X & 1 & 0 \\ X & X & X & 0 \\ X & X & X & X \end{bmatrix} \rightarrow \begin{bmatrix} X & 1 & 0 & 1 \\ X & X & 0 & 0 \\ X & X & X & 0 \\ X & X & X & X \end{bmatrix}$$

Fig. 4: Priority matrix transition when requestor 2 is granted access.

The gate-level design of an ***n: 1*** matrix arbiter, which handles n requestors, is shown in Fig. 5. An arbiter has n grant circuits, each determining if a requestor should be granted. The grant signals then feed update circuits, which updates matrix priorities.

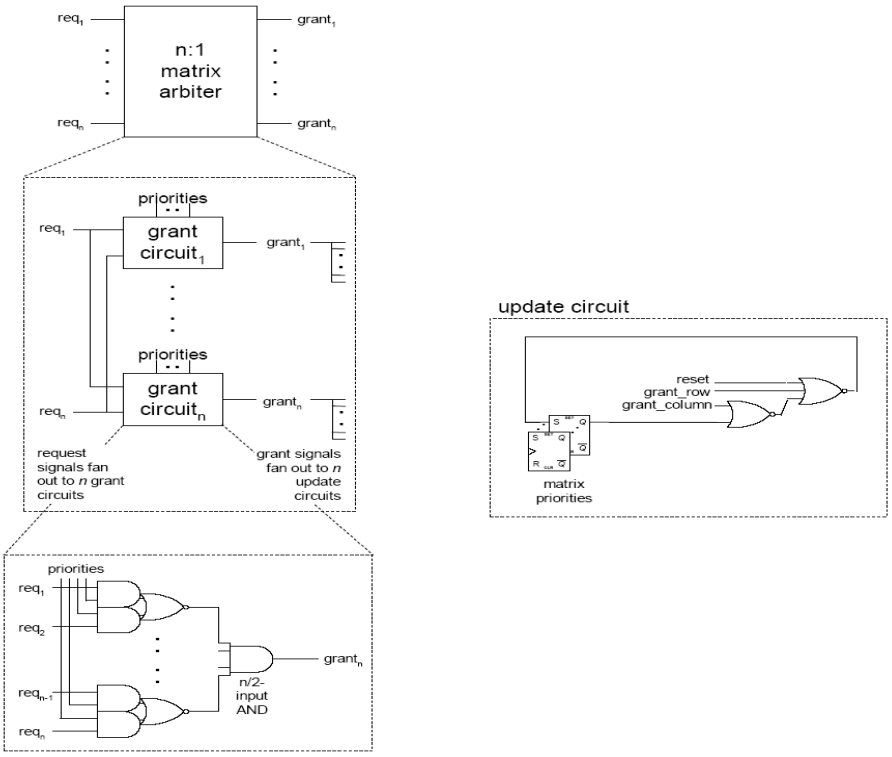

Figure 5: Gate level design of matrix arbiter

2 Design Details

In this lab you are supposed to design a 4:1 arbiter, i.e. there will be 4 requestors trying for a single resource. As shown in Fig. 1 for a 4:1 arbiter you will need 12 elements in the matrix arbiter. But remember $p_{ij} = \overline{p_{ji}}$. In this situation you will need only 6 elements in the priority matrix. You can implement each element of the priority matrix as an S-R flip flop, with Q representing p_{ij} and $\overline{Q}$ representing p_{ji}.

The block diagram of the matrix arbiter and the circuit diagram to implement one element of the priority matrix are shown in Fig 6.

First create one element of the priority matrix, combining the S-R latch and the update circuit.

The Grant block will have the requests and the priorities as inputs and it generates gnt_1 to gnt_4 signals. The ***gnt$_i$*** and ***gnt$_j$*** signals are coming from the grant block. Let *req$_i$* be the *i*th request; *gnt$_n$*, the *n*th grant; and m_{ij}, the *i*th row and *j*th column element in the priority matrix. Using these variables,

$$gnt_n = req_n \times \prod_{i<n} \left(\overline{req_i} + \overline{m_{in}} \right) \times \prod_{i>n} \left(\overline{req_i} + m_{ni} \right)$$

The Priority matrix block will have gnt_1 to gnt_4 signals as inputs and it generates the priority signals as outputs.

The top level of the arbiter will have two components, the priority matrix with the update elements and the grant circuit. The inputs to the top level are the four request signals and the outputs are the grant signals. Additionally, you will need to design the input mux and FIFO buffers.

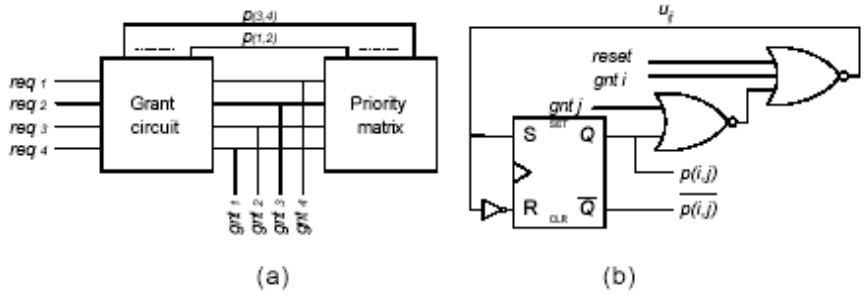

Fig. 6: (a) Block diagram of an arbiter; (b) one element of the priority matrix

After designing the entire input port (arbiter, MUX, and FIFOs), you need to test it for correct functionality by writing a test bench in Synopsys. Once you are done with the functional simulation, you need to synthesize and provide the power and area reports as well as a timing report showing you met timing requirements for your given clock.

3 Report

You are required to submit a brief 1-2 page report clearly showing how you have implemented each block. You need to submit the VHDL (or Verilog) code also. If you work in a group of two then submit one report each group.

Each report should start with the following declaration signed by each student

> "The submitted report is my own work. I/we have not copied the program from any other sources. If found guilty of copying then I/we will receive a grade of "zero" for this lab."

4 Due Date

You need to demonstrate and submit your design by 8th November.