Problem statement 4:

Design Contention Resolution and switching logic in VHDL using Xilinx ISE. Write a testbench that include all possible test cases in order to verify your design using simulation. Implement your verified design on the ATLYS board to verify the proper functionality of your design on hardware.

Top-Level block diagram for your design is shown in Fig.9. The descriptions of the interfaces are given in Table-V. For design of your logic you need to use only the given interfaces at the top level of the design. Internally you can have temporary signals of your choice.

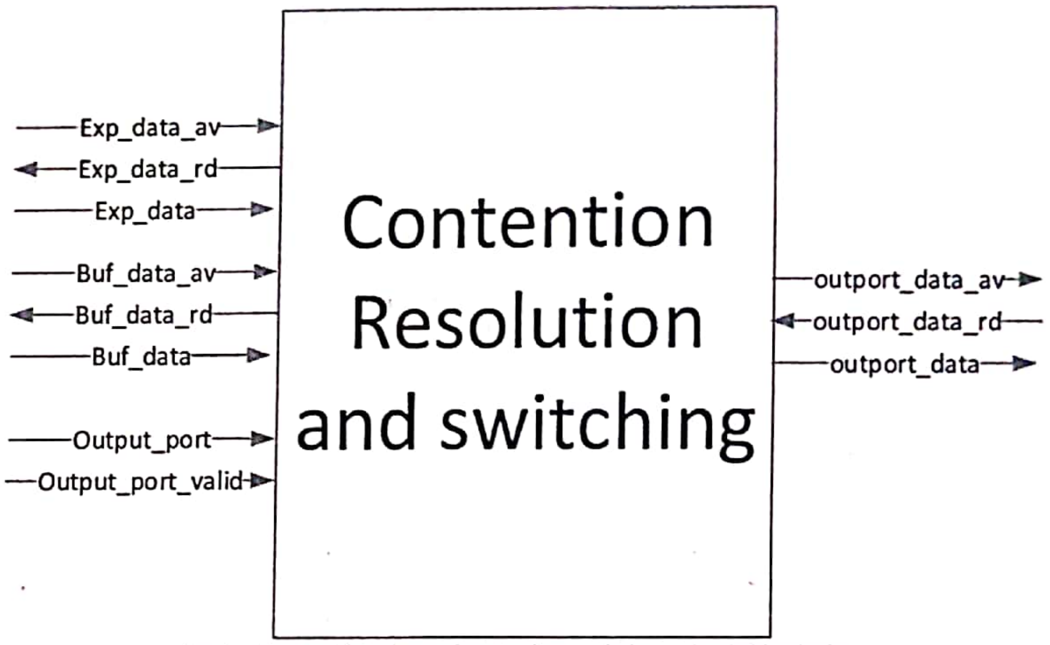

Fig. 9. Top level interfaces of contention resolution and switching logic.

Table V. Signal description.

Interface	Direction	Description
`Outport_Data_av`	Output	This is a Nx1-bit signal. Where N represents the number of ports in the switch. A logic high represents the availability of the data at your logic for respective remote interface logic.
`Outport_Data_rd`	Input	This is a Nx1-bit signal. This signal is driven by remote interface logic for reading data for respective remote interface.
`Outport_Data`	Output	This is a Nx144 bit signal. This is the 144-bit outgoing data from your logic to respective N-ports. Remote interface gets a valid data for each clock cycle when itassertExp_Data_rd signal to your logic.
`Exp_Data_av`	Input	This is a Nx1-bit signal. Where N represents the number of ports in the switch. A logic high represents the availability of the data at the respective remote interface logic.
`Exp_Data_rd`	Output	This is a Nx1-bit signal. This signal is driven by your logic for reading data from the respective remote interface.
`Exp_Data`	Input	This is a Nx144 bit signal. This is the incoming data to your logic from N-ports. You receive a valid data for each clock cycle when you assert respective `Exp_Data_rd` signal in your logic.
`Buf_Data_av`	Input	This is a Nx1-bit signal. Where N represents the number of ports in the switch. A logic high represents the availability of the data at the respective remote interface logic.

Express
(High prio.)

`Buf_Data_rd`	Output	This is a Nx1-bit signal. This signal is driven by your logic for reading data from the respective remote interface.
`Buf_Data`	Input	This is a Nx144 bit signal. This is the incoming data to your logic from N-ports. You receive a valid data for each clock cycle when you assert respective `Buf_Data_rd` signal in your logic.
`Output_port`	In ~~Output~~	This is an ~~8~~-bit signal, which represents the output port for the packet. nx8
`Output_port_valid`	In ~~Output~~	This is a ~~1~~-bit signal, indicate the `Output_port` signal is valid. n
`clk`	Input	This is a 1-bit clock signal for the logic.
`rst`	Input	This is a 1-bit reset signal for the logic.

Design description:
There are N input and N output ports in the switch. A packet coming from a port can go to one of the N output port. In your design you need to create a virtual output queueing logic, where each input have N output buffers corresponding to N output port. This logic helps in avoiding head of line blocking. This way you will have total NxN buffers in your logic and your output port scheduling logic will read packets from these N buffers. There is possibility that packets coming at two different input port are destined to same output port at the same time. Any output port can only be accessed by one of the input port at any given instance of the time. Therefore, you also need to design a scheduling logic such that each output port runs their individual scheduling logic and allow access to appropriate input port.

For example, a packet coming at input port 3 which have its destination at output port 5, then you need to store packet in buffer number 5 of N buffers allocated to input port 3. Similarly, if a packet is coming at input port 8, which have its destination at output port 5, then you need to store packet in buffer number 5 of N buffers allocated to input port 8. Port-5 faces contention from Input port-3 and port-8. Therefore, there is a requirement of scheduling mechanism to allow only one port to transmit data to output port at a time.

Tasks to perform:

1. Create NxN VOQs.
2. Read packet over `exp_data` or `buf_data`signal.
3. Based on the input port and requested output port, write packet in respective VOQ.
4. Design an arbiter to schedule packet transfer from N inputs to a particular output port.
5. Arbiter should connect only one input port to a output port at any given instance of time.
6. Identify the `end_of_packet` marker. Once `end_of_packet`marker detected allow next input port to transfer packet.
7. Create N instances of such arbiter one for each output port and run them in parallel.

Expected Outcome:
Commented synthesizable VHDL code of your design, which functions as described in problem statement.
Testbench written in VHDL for simulating the different possible test cases for your design.