Mini Project

Ajinkya.S.Raghuwanshi(ID:-19307R019)

November 2020

1 Introduction

A switch is a system hardware that connects devices on a network by using packet switching to receive and forward data to the destination device. Switches manage the flow of data across a network by transmitting a received packet only to the one or more devices for which the packet is intended. It is a multi-port bridge that uses addresses to forward data. Here I have designed a 4x4 switch which provides the functionality to send data from one of the 4 inputs to the 4 outputs. The packet length is also kept variable.
The design here can handle packets with a maximum size of 256 bytes(i.e in our case here to be 64 packet length). The packet length here are random values generated at the time of execution.

2 Block Diagram

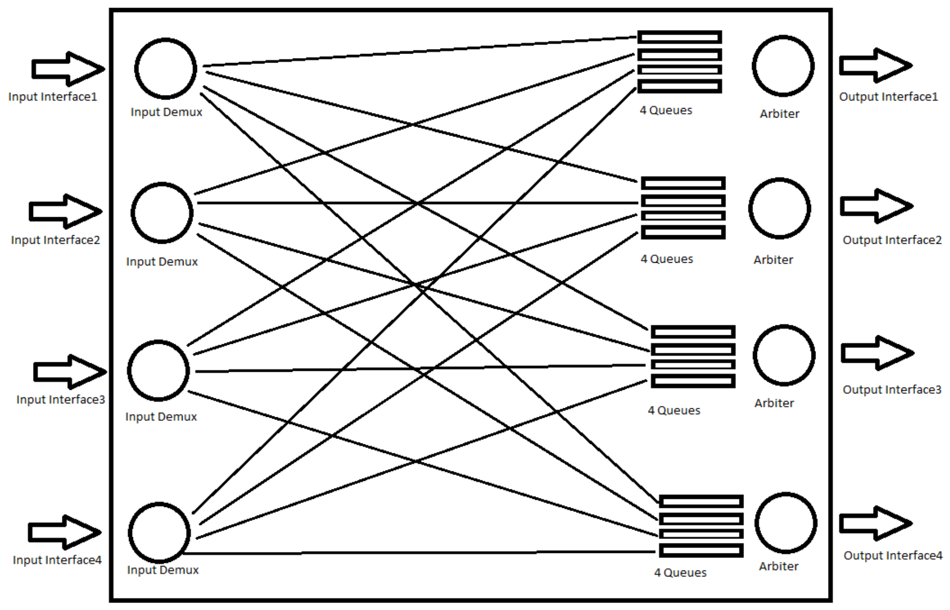

Figure 1: Block Diagram of 4x4 switch.

Here the implementation of the circuit is shown above in the block diagram. This implementation will be explained below.
Here I implemented the switch which includes input buffer, input Demux, output queues and arbiters.

3 Design Decisions

For the design of 4x4 we have the following types of implementations:

1. Monolithic model.
2. 4 seperate input-output combination model.
3. Seperate model for every block.

So, I decided build the Monolithic block as it is easier to debug and was explained in the class.
The module will contain the input buffer, input Demux, output queues and arbiters in a single body.
The components are implemented as daemons because we want them to be operated continuously waiting for the event to occur for their operation to start.

INPUT BUFFER : The input buffer will store the data coming from its respective input and will store it in the buffer.

INPUT DEMUX : The input Demux will basically read the address location from the header packet to be sent and will be forwarded to the respective output queue.

OUTPUT QUEUE : The output queue will receive the inputs from all the input Demuxes and will give it to the arbiter.

ARBITER ; The arbiter will then look at the current packet according to the priority provided to it by the priority algorithm and will give out the output at the end of the module.
The priority order here we decided is to be the Round Robin priority method in which the priority is rotated for the queues.

4 Implementation

The design of the switch is so as the internal modules will work all parallel rather than working one at a time.

INPUT BUFFER: The input port buffer stores the data that comes to the respective input port and stores it in the FIFO. These are non-blocking buffers (i.e they send the data a zero even if the queue does not receive the input to it).

INPUT PORT DAEMON: The data of the packet[0] is used to decode the destination where the complete packet is supposed to go. The packet[0] is

sliced then to remove the destination ID from it. This destination ID is given as the input to the demux and the according to the location of the output the data is sent forward.
Here the input buffer is implemented as a pipeline. The depth of the pipe of the input buffer is 2. As it seemed to utilize the model completely for the traffic reaching it. The buffer used here are all non-blocking buffers.

OUTPUT PORT DAEMON: Here the output port daemon incorporates of the output queue and the arbiter. The output queue holds the data sent yo the port. There are 4 queues at every output port in my model. All the queues used in the output port are non-blocking buffers(i.e they send the data a zero even if the queue does not receive the input to it). The arbiter take from the output queue and sends it to the output according to the priority of the data queue. The arbiter checks if the a transaction is complete, if it is, it then transfers the control of output of the data output to the particular queue and this process is continuously carried out at every output port.
The data priority is decided by the round robin method in which the priority of the queue will change in every new transaction. At every transaction the priority will be transferred to the next in the numerical order where at the initialization we start from the queue1 to then queue2 and so on.

5 Design verification

For the verification of the design I created a test-bench that sent the input to the switch and took the respective output from the switched. Now to verify the correctness of the output the the input and the output to the switches were compared and if there was a error or mismatch in the data then the system gave an error message at the output. This process is carried out for all the input combinations for the switch and their respective outputs and thus the correctness of the system was verified.

6 Results

The system gave all the outputs to be correct and what was expected from it. The switch wrote at the output of the system at every clock cycle which meant that there was 100 percent utilization of the circuit and the system was giving the best performance.

7 Conclusion

The performance of the system was found to be as expected. The use of the AA language reduce the non-fruitful work of repetitive code in the system and made it easier to code and to debug, the AA compiler also create the test-bench in VHDL and helped in the analysis of the system.