Algorithmic Design of Digital Systems
Mini-Project:
A 4X4 Switch

Mitul Tyagi, Roll no. : 193079017

18th October 2020

1 Introduction

An NxM switch provides functionality to switch data from 1 of the N input ports to 1 of the M output ports. Let the input ports be numbered 0,1,2,..., N-1 and the output ports are numbered 0,1,2,...,M-1. Input packets arrive at the switch. This mini-project implements a 4x4 Switch. The first word in the packet is called the header, and its format is as follows:

Interpretation	Bits
Destination	[31:24]
Length	[23:8]
Sequence-Id	[7:0]

2 An output-queued switch

Incoming data at an input de-multiplexer is written into the appropriate destination queue. At each output port, there is an **Arbiter** which checks the queues that it is managing to receive the packet. It serves the queues in a fair manner using the **Round-Robin Policy**. The packets in the selected queue are then sent out in its entirety, to the output port.The block diagram of a 4x4 Switch is shown below:

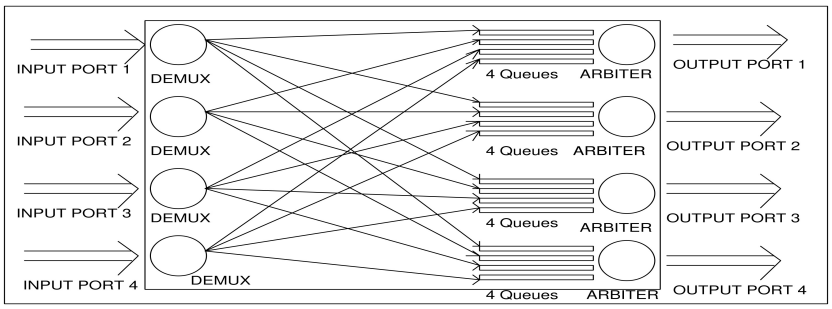

Figure 1: Block Diagram

3 Target

The goal was to design a 4x4 Switch, which can handle packets with varying lengths for following kinds of traffic:

- Data comes in from a single port and goes out to a single port
- Data comes in all ports and goes out to the same destination port
- Data comes in all ports and is distributed across the destination ports.

4 Design Principles

The two entities/modules, **Arbiter** and **Sender**, were designed using the Aa Language and the entire system was tested using a C Testbench. The Sender Module receives the packet from the input port pipe and routes them to the corresponding output port queue. The packets are then sent to the output port pipes by the Arbiter Module. The Arbiter Module at each output port implements a **Round-Robin Policy** to select the queue from which the packet will be sent.

The C Testbench validates the packet for correct destination as well as data. The design tools convert this Aa Code to the VHDL Code and the simulation was performed using **GHDL**.

4.1 Sender Module

This module reads the data from the input port and routes the packet to the corresponding output port queue. Since the packets are of varying length. The routing algorithm always start by reading the header packet to get the length of the packet and then routes all the following packets accordingly. Once a transmission is completed then it restarts with a new transmission by reading the packets from the input port pipe.

4.2 Arbiter Module

This module reads the packet from the four queues and routes the packet to the destination port pipes. It is interacting with a **Priority_Select Module** which implements the round robin scheduling algorithm. This module selects the queue from which new packet has to be read for transmission. The algorithm is fair and does not cause any starvation issue among the queues.

4.3 Priority_Select Module

The **Arbiter Module** interacts with this module to select the next packet queue after a successful transmission of the packet. This module implements the *Round-Robin Policy* to select the next queue for the transmission of the packet.

4.4 C Testbench

This consists of 4 input threads sending the data to the input port pipes and 4 output threads receiving the data from the output port pipes. This testbench helps in validation and functional verification of the system.

4.5 Design Parameters

Following design parameters were used while performing simulation:

- Depth of Pipes at input and output ports was 3.
- Queues at the output ports have the capacity to hold 4 packets at a time. Simulation with queue size of capacity of 4 packets was done.From the simulation it was observed that a queue size of 4 packets is sufficient. Reducing the queue size to 3 or 2 reduces the performance of the system.

5 Compiling the Project

It involves following three main steps:

- Compile the Aa Files to generate the C files and VHDL files for testing and simulation. Run the script **compile.sh** present in the **hw folder** .
- Compile the testbench file using the script **build_aa2c_tb.sh**.
- Build **GHDL Model** and **GHDL Testbench**.

A bash script **build.sh** can alone be used to run all the steps mentioned above

6 Running the Project

To verify using the C Testbench run the command **./bin/testbench_aa2c** with appropriate inputs. To verify using the VHDL Simulation run the command **./bin/testbench_vhdl** followed by **./ahir_system_test_bench** with appropriate inputs. The simulation result can be dumped to a file and can be viewed using **GTKWave Viewer**.

7 Debugging the Project

To debug the system provide the file option to generate the *trace file* and then use the script **separate.sh** to generate the logs for all the four input ports and output ports. The logs are generated in the *debug_files folder*.

8 Performance Analysis

From the VHDL simulation we observed that the packets were received at every clock cycle on the output port and hence utilising the full capacity of the system. The same can be inferred from the Figure 5.

9 Results and Outcomes

The system design was complied successfully and was run for the case when packets were being sent from all the input ports to all the output ports. The results are shown below.

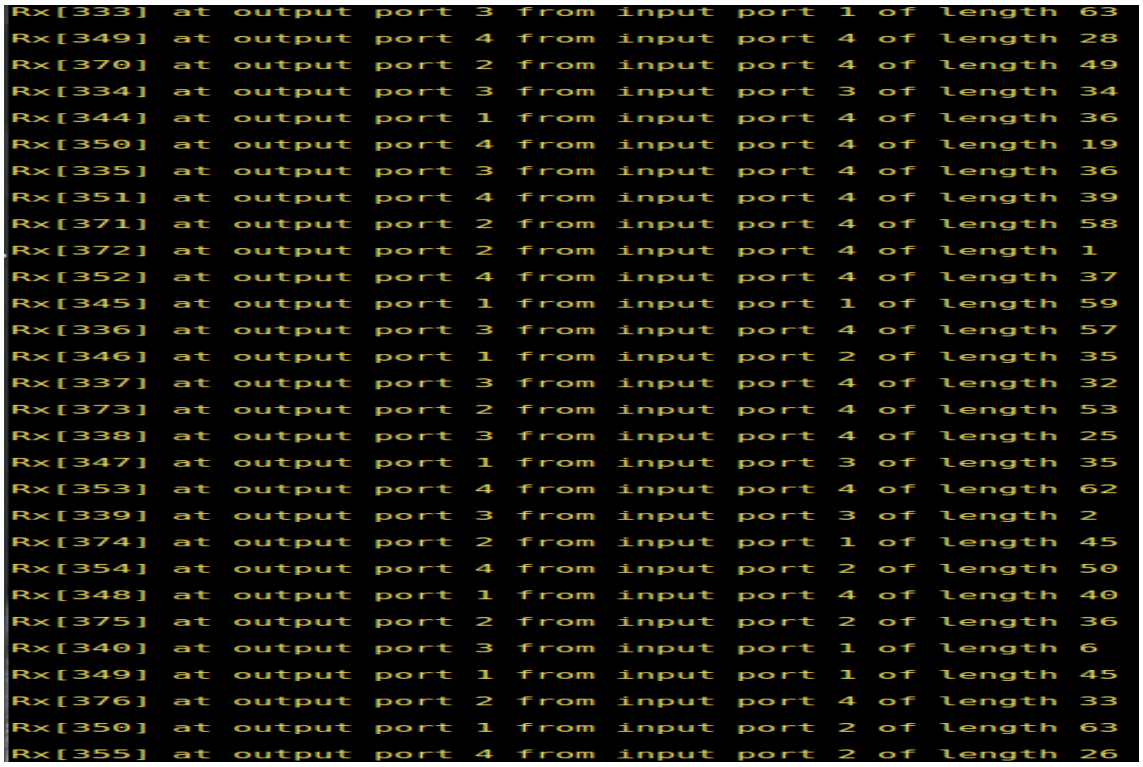

Figure 2: C Testbench Verification for **alltoall** Case

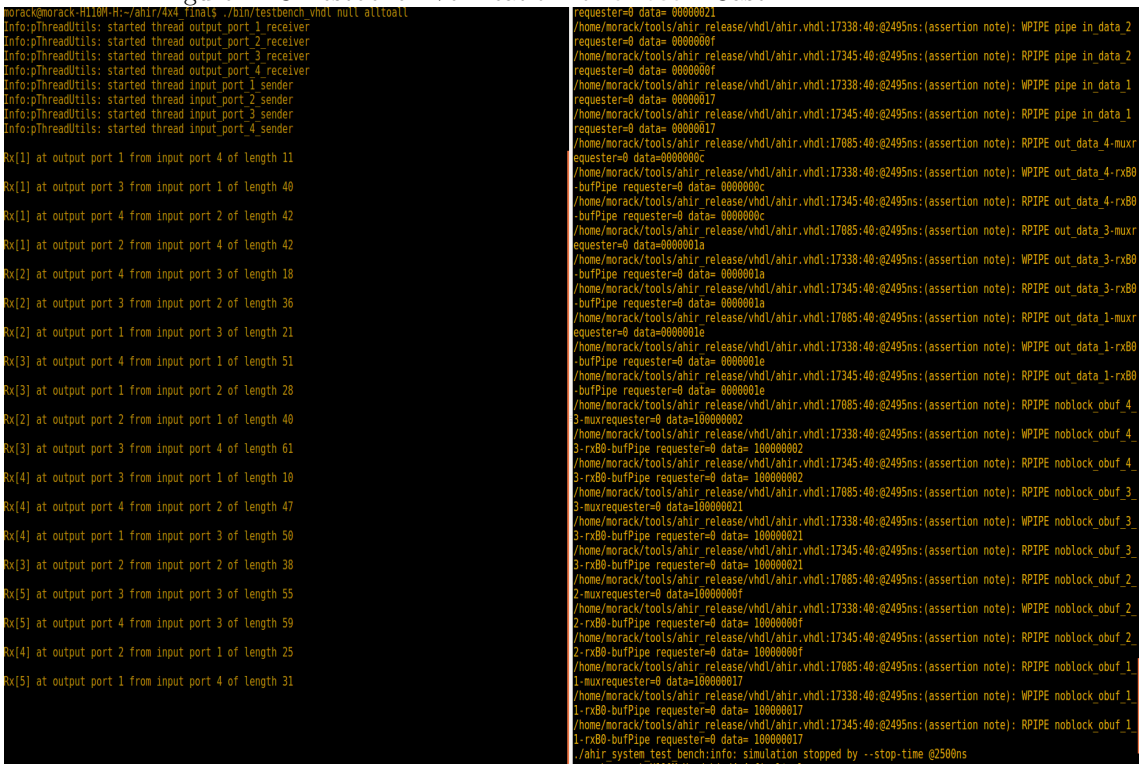

Figure 3: VHDL Simulation for **alltoall** Case

Figure 4: GTK Wave Output for **alltoall** Case

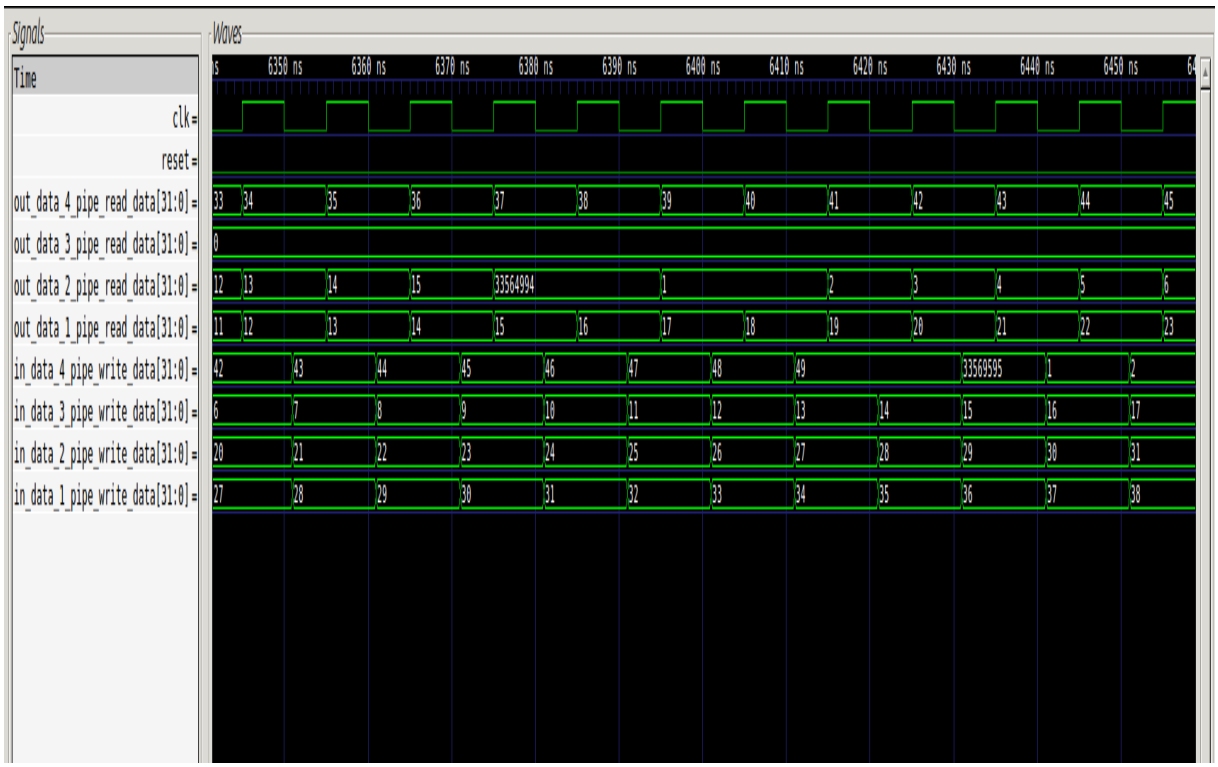

Figure 5: GTK Wave Output Verification