بنام خدا

Digital Systems Design Using VHDL: Project

Mohammadreza Binesh Marvasti

E-mail: marvasti@khu.ac.ir

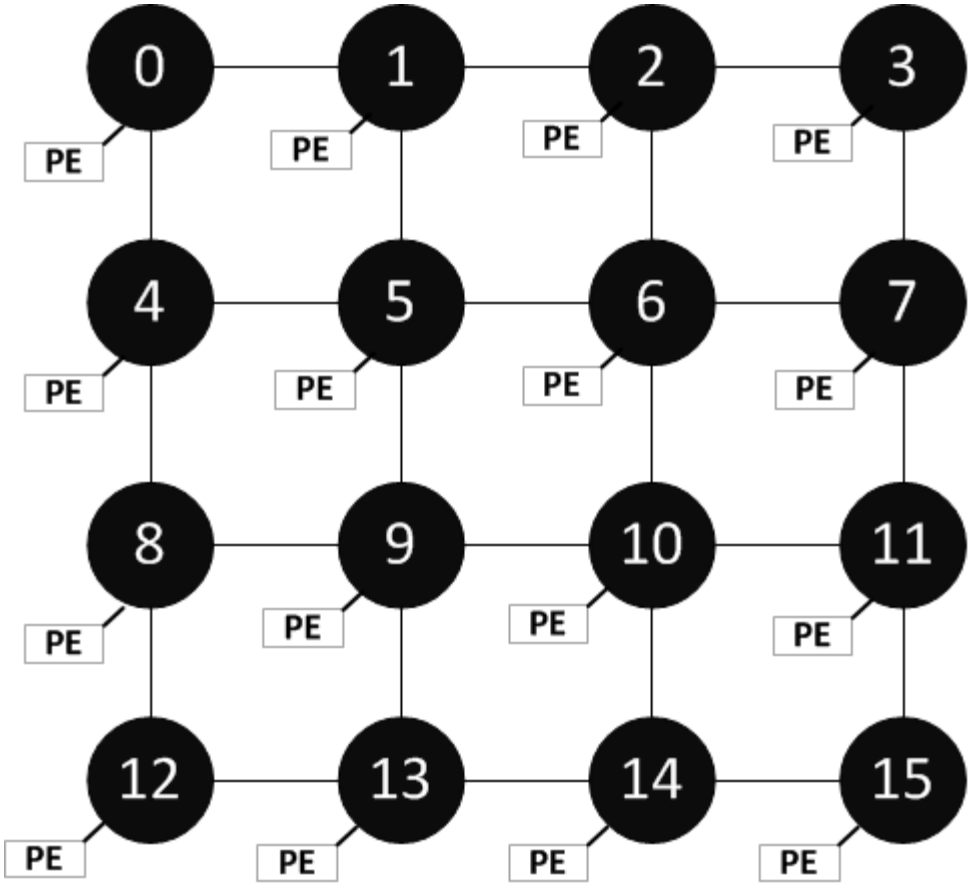
Project

- In this NoC design we use a VHDL based methodology to design a NoC system. You will work in a practical FPGA design environment using Quartus-II from Intel and Modelsim.

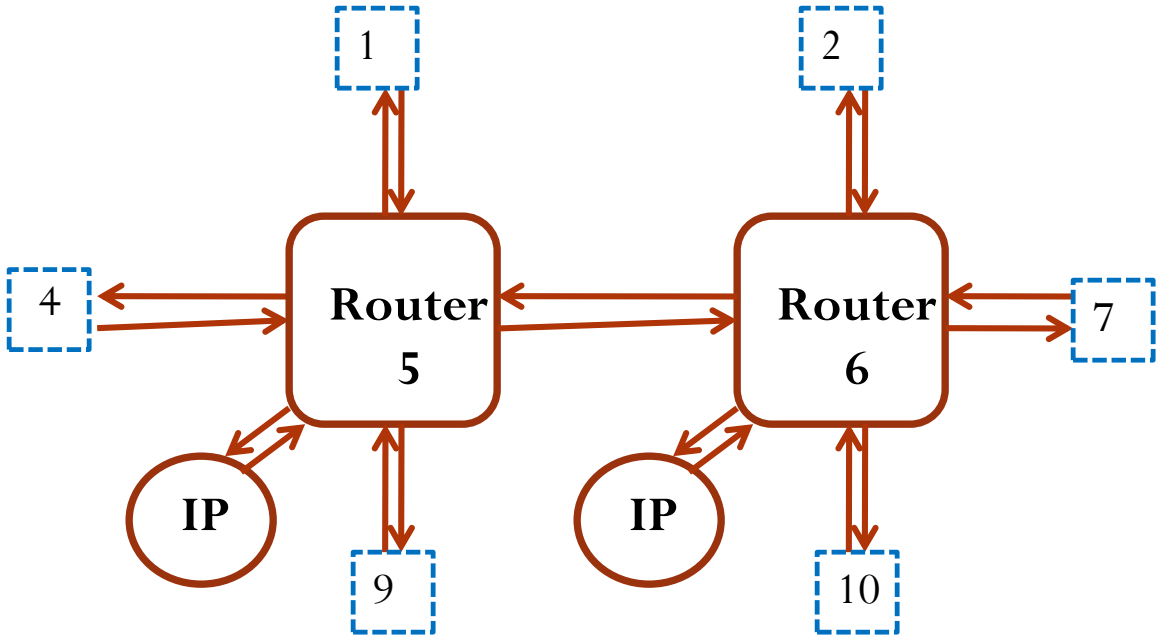
- In this project, students will investigate and design a NoC system consisting of the router/switch, IPs (CPU or other hardware module), and interconnection structure (topology) such as **Mesh**.

- The communication among the NoC elements (e.g. Router, source and destination IP cores, interconnection, etc.) is based on asynchronous handshaking that each module can have different clock modules.

Project



Interconnection Network

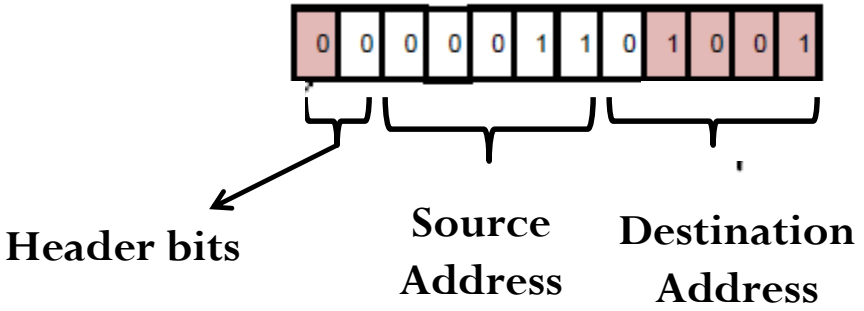


Packet Format

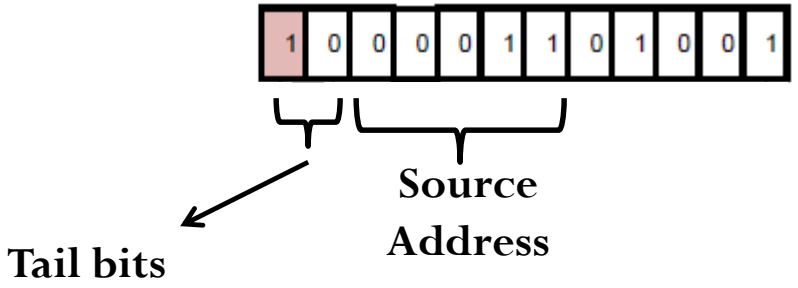
- Communication among various NoC components takes place via routing packets. The IP core generate and inject packets to the NoC.
- In packet switching, a packet is broken into multiple bits.
- A it is the smallest physical unit of data that is routed by the network. The first it of the packet is called *header flit*. It contains the destination address (i.e. routing information) of the packet.
- The last flit of the packet is called *tail flit* which indicated the end of transmission for the packet. The actual data of the packet are in the rest of its which are called *payloads*.
- Each flit is **12** bits, and a packet is **2** flits.

Packet Format

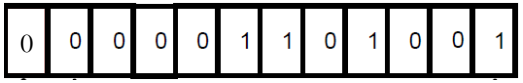
- Header flit



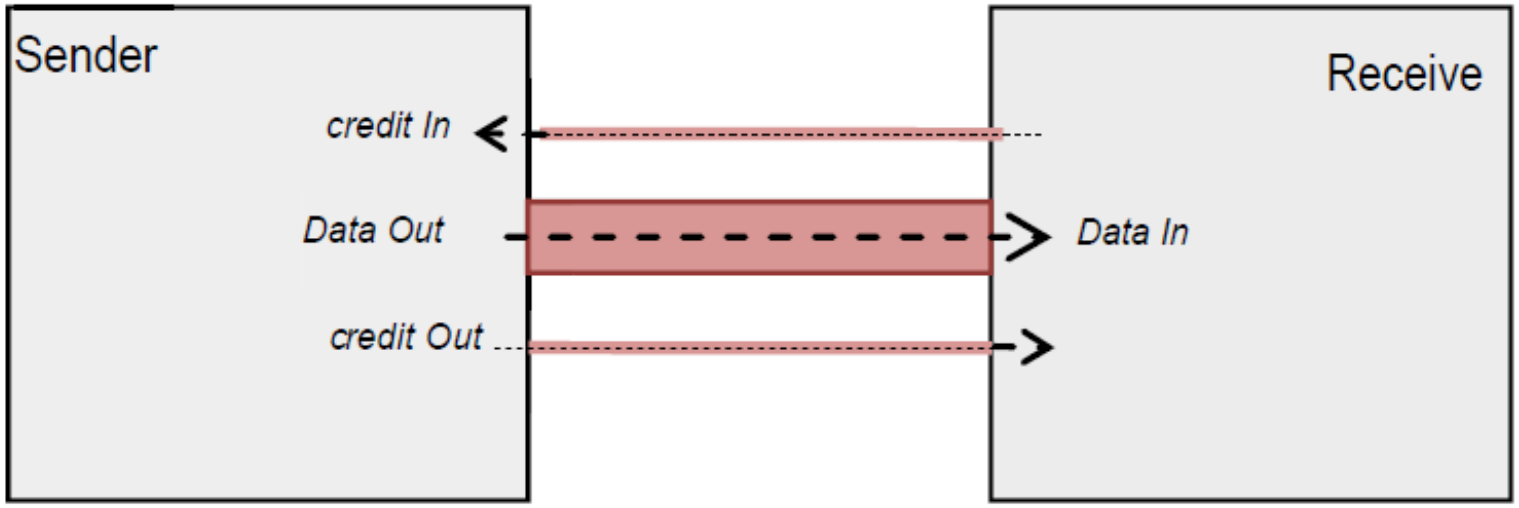
- Tail flit

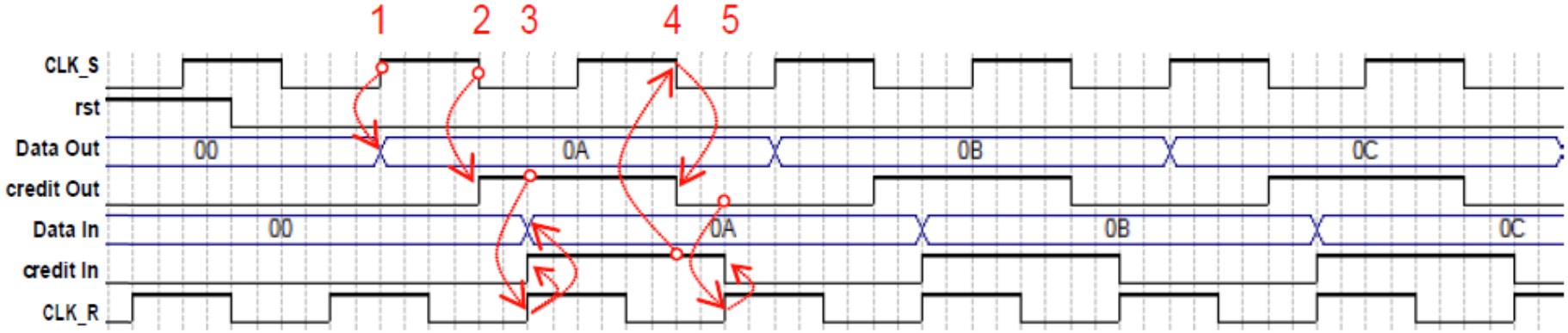


- The rest flits are payload and the two MSBs must be 00



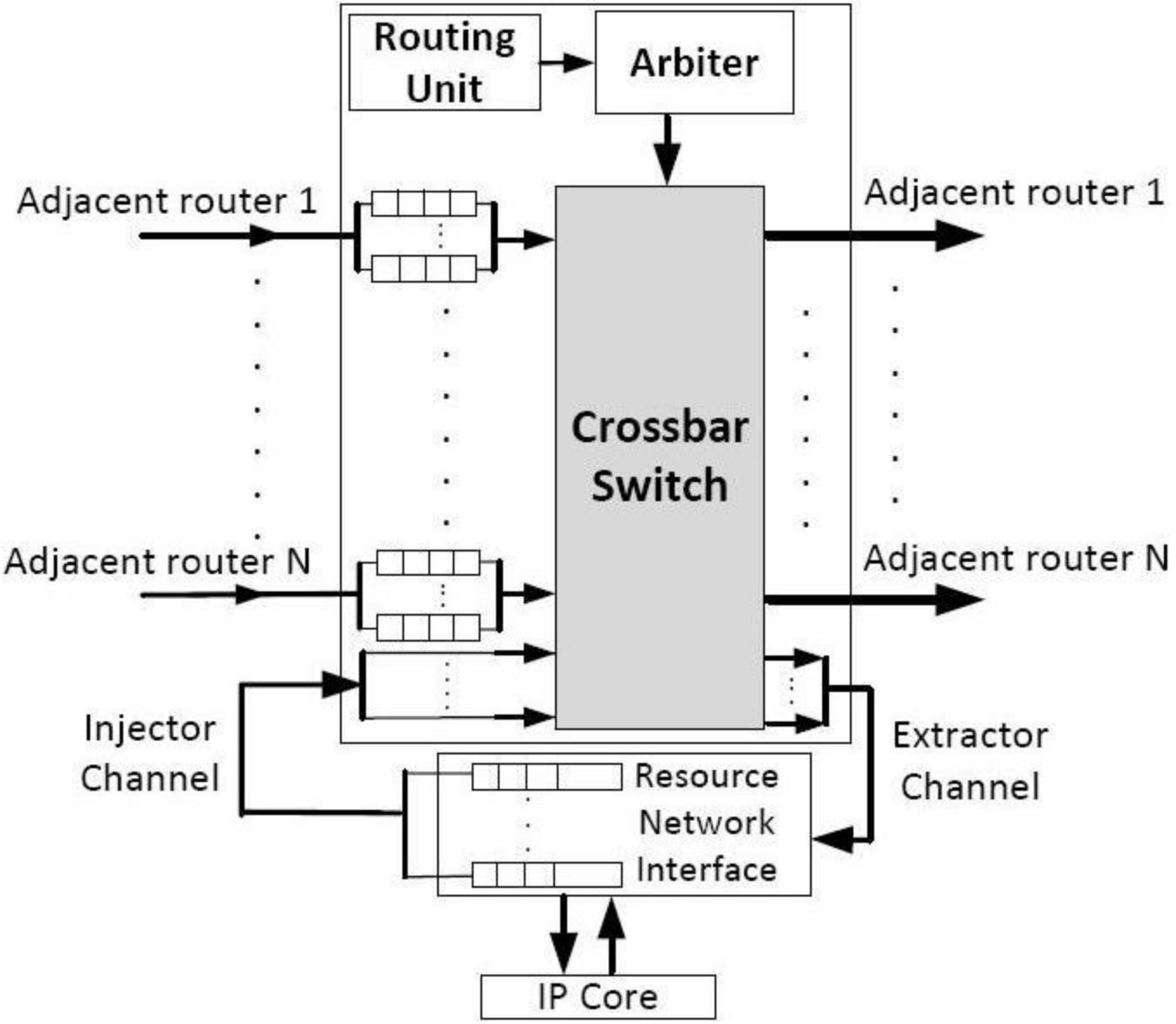
Asynchronous Communication





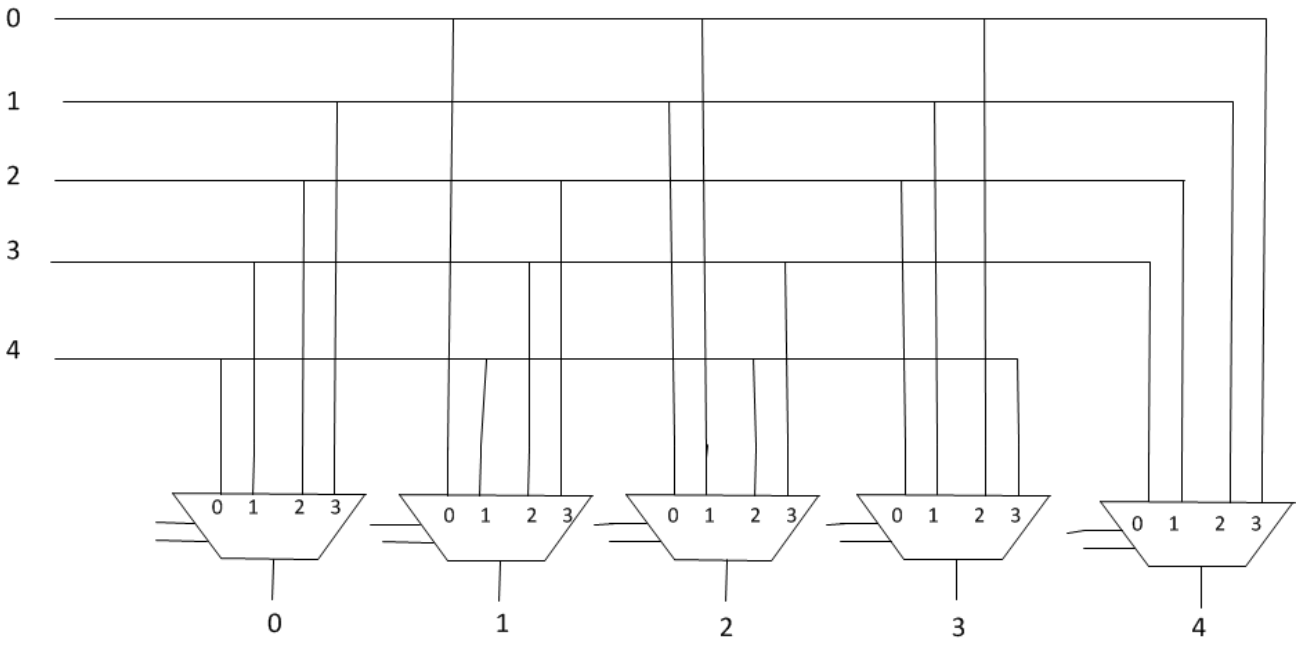
Asynchronous Communication

- At event #1, at positive edge of sender clock, clk_s a data is transferred out of sender. Then at half clock later, event #2, a credit_out is transferred out of sender too.
- Then at event #3 or at positive edge of receiver clock module, clk_r, the credit from sender (credit_out) is detected, and then the data is stored in the receiver. The receiver issues a credit signal (credit_in) back to the sender.
- Then at negative edge of sender clock or event #4, when credit_in is high, it leads to deactivate the credit_out signal. Then at positive edge of receiver clock or event #5, when credit_out is detected as a low signal, the receiver deactivate credit_in signal.
- After event #5, the sender is free to send new data. If the receiver is not ready to receive data (e.g. it is full), it does not deactivate credit_in signal at event #5. Therefore, the sender does not start sending new data.

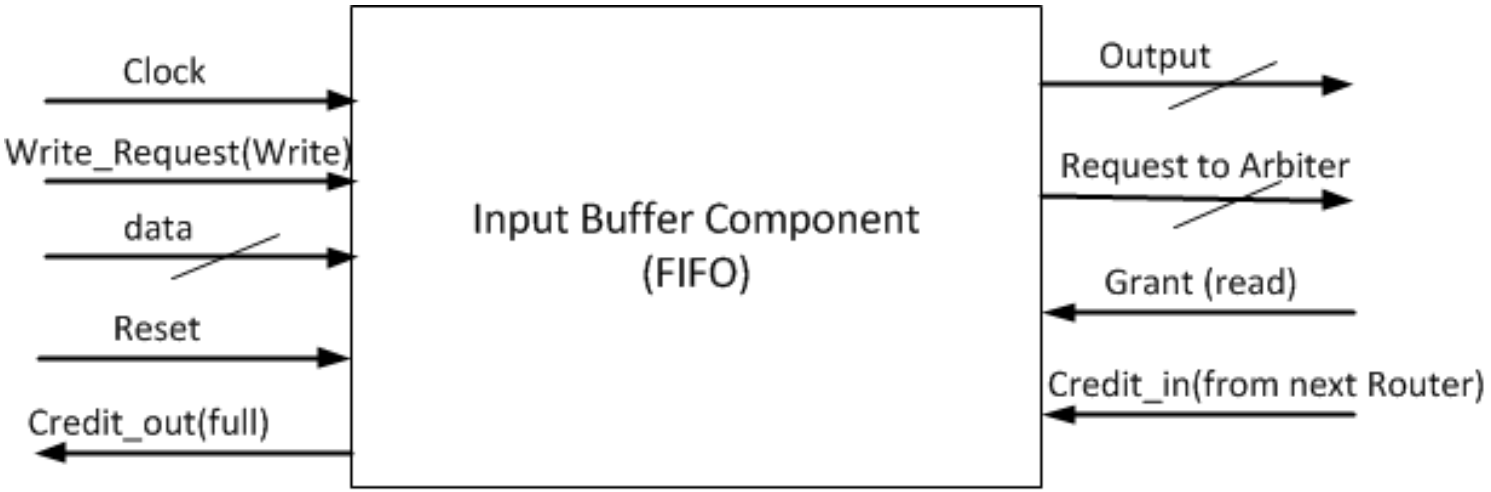

Routing Unit
Arbiter
Adjacent router 1
Adjacent router 1
Crossbar Switch
Adjacent router N
Adjacent router N
Injector Channel
Extractor Channel
Resource Network Interface
IP Core

Crossbar Switch

- The crossbar switch is the final stage of the router. It maps the packets coming from the input ports to the assigned output ports.



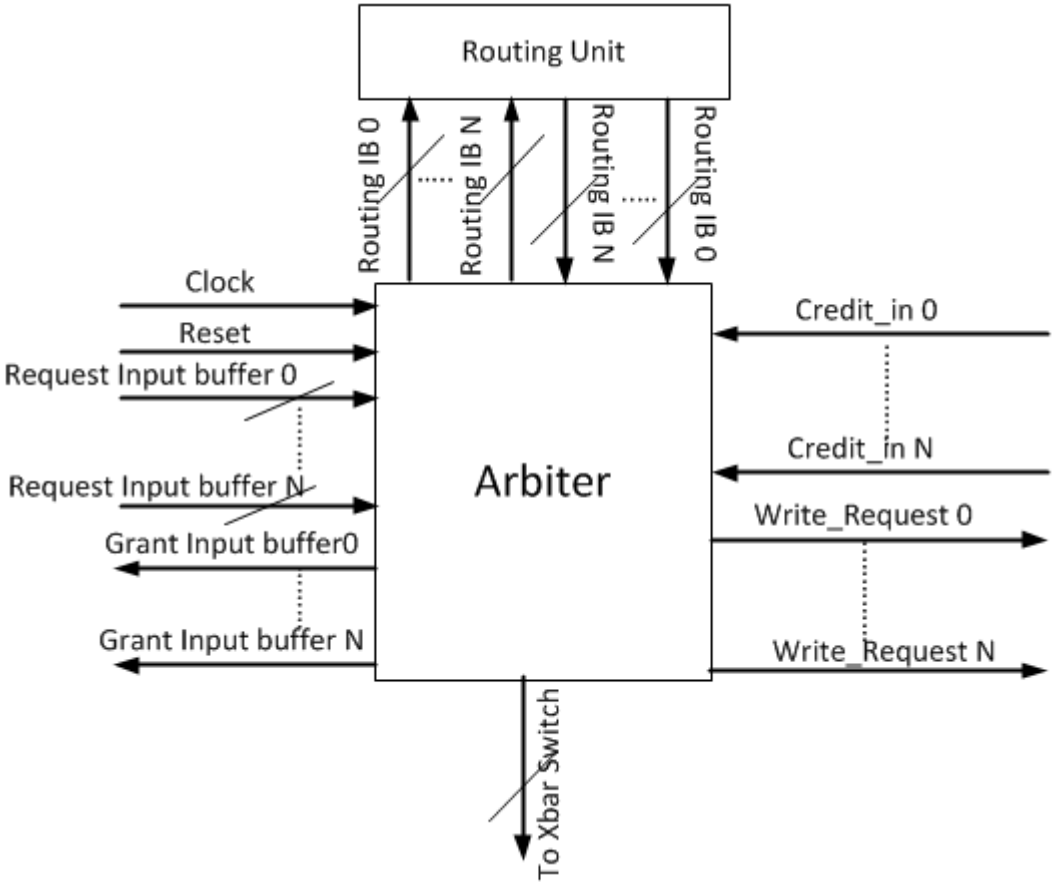
FIFO Buffer


Clock
Write_Request(Write)
data
Reset
Credit_out(full)
Input Buffer Component
(FIFO)
Output
Request to Arbiter
Grant (read)
Credit_in(from next Router)

FIFO Buffer

- Associated with each input port, an input buffer is used for temporarily storing the incoming flits. The input buffer works on the basis of first in first out (FIFO) mechanism. The input buffer component includes 2 parts, one part which handles the controlling signals and the second part for storing the incoming packets.
- The reading operation stops when the control signal **Credit in** coming from Arbiter which indicates that no more space is available in the adjacent.
- However, the storage in the current buffer will continue get new flits until it becomes full.
- The control signal "**Credit out**" represents as a back pressure to the adjacent source node, when FIFO Buffer becomes full.
- As soon as the header it reaches to the FIFO, the packet information included in this header it will notify the Arbiter about the arrival of the packet in this port, and provide the destination address to the Routing Unit to calculate the direction.

Arbiter


Routing Unit
Routing IB 0
Routing IB N
Routing IB N
Routing IB 0
Clock
Reset
Request Input buffer 0
Request Input buffer N
Grant Input buffer0
Grant Input buffer N
Arbiter
Credit_in 0
Credit_in N
Write_Request 0
Write_Request N
To Xbar Switch

Arbiter

- In order to move the packets through the switch, an arbiter is used in every router. The arbiter generates arbitration signals to provide synchronous connections between any pair of input ports and output ports of crossbar switches.
- Once it receives a request from an input port, it sends out the routing information to the routing unit and receives the direction information.
- Then, the availability of free buffer locations in the neighboring destination router is checked through examining the validity of the signal "Credit in". If it is available, it generates a grant signal to the proper input buffer component and as well as crossbar switch. Is also generates "Write request" signal to the input port of next router.

Routing Unit

- In order to find the destination output port for a packet in a switch, a routing unit is used.
- This unit is responsible for obtaining the direction that packet should take based on the address provided by the head it. It receives the address from arbiter unit.
- Once it receives an address, it decodes the address and generates destination port and send it back to the arbiter unit.
- It is constructed by some multiplexers to provide the associated output port for the packet.
- We will implement the XY routing algorithm.

Simulation

MUST be Simultaneously

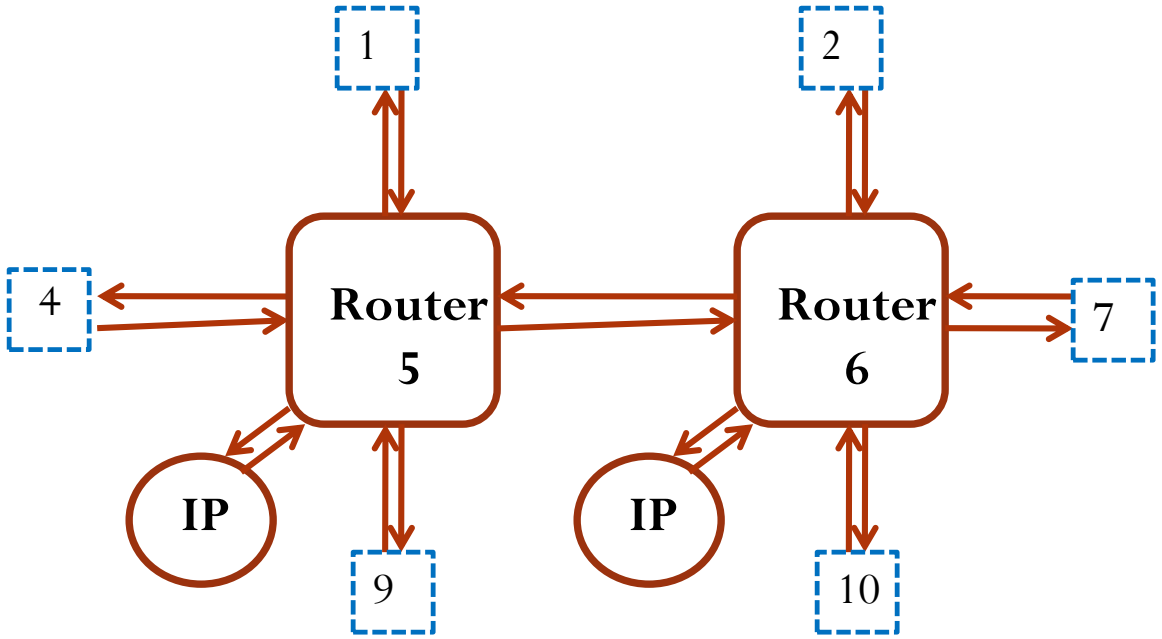
- 1:
  - 4->10
  - 7->1
  - 5->6
- 2:
  - 6->9
  - 10->1

80%

40%

120%!!!

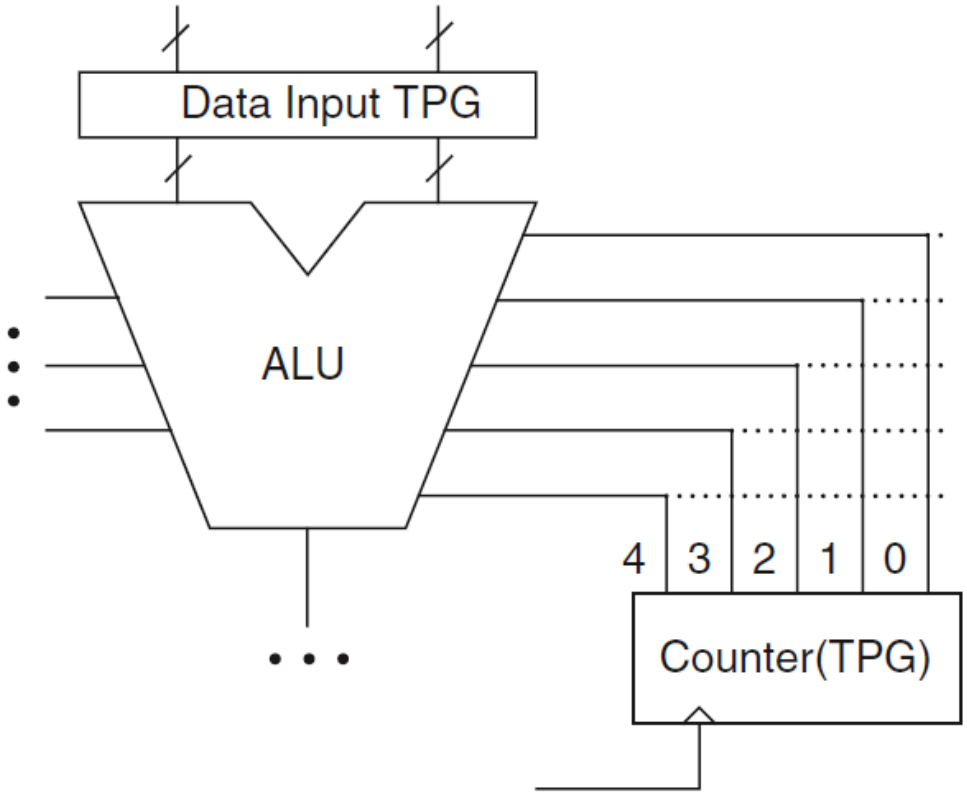
Interconnection Network



Input Generation (Stimuli)

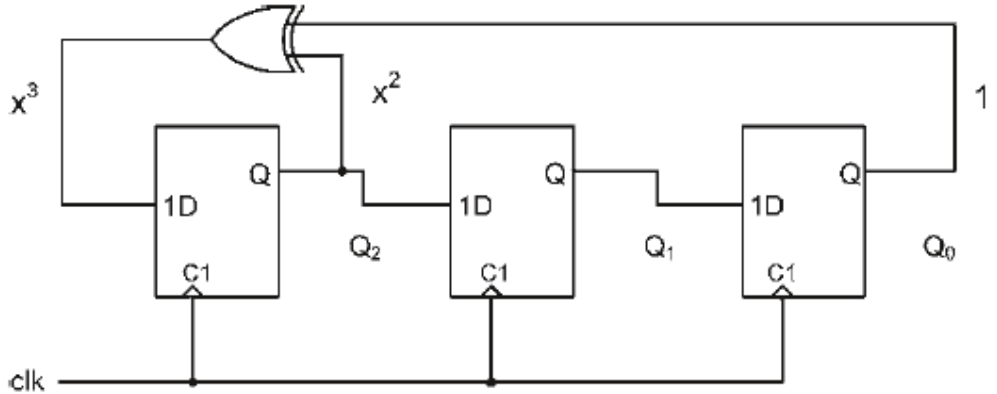
- Exhaustive Counters
  - It covers all the possible combinations, good for ALU verification!
  - In an *n*-input component, there are 2^n combinations
- Random
  - It can cover a good number of combinations!

Exhaustive Counters



LFSR (Linear Feedback Shift Register)

- A sifter register with a special feedback circuit to generate the serial input
- The feedback circuit performs XOR operation over specific bits
- Can circulate through 2^n-1 states for an **n**-bit register
- 3-bit LFSR:



$P(x) = x^3 + x^2 + 1$

Q_2	Q_1	Q_0
1	**0**	**0**
1	1	0
1	1	1
0	1	1
1	0	1
0	1	0
0	0	1
1	**0**	**0**
1	1	0
...	...	...

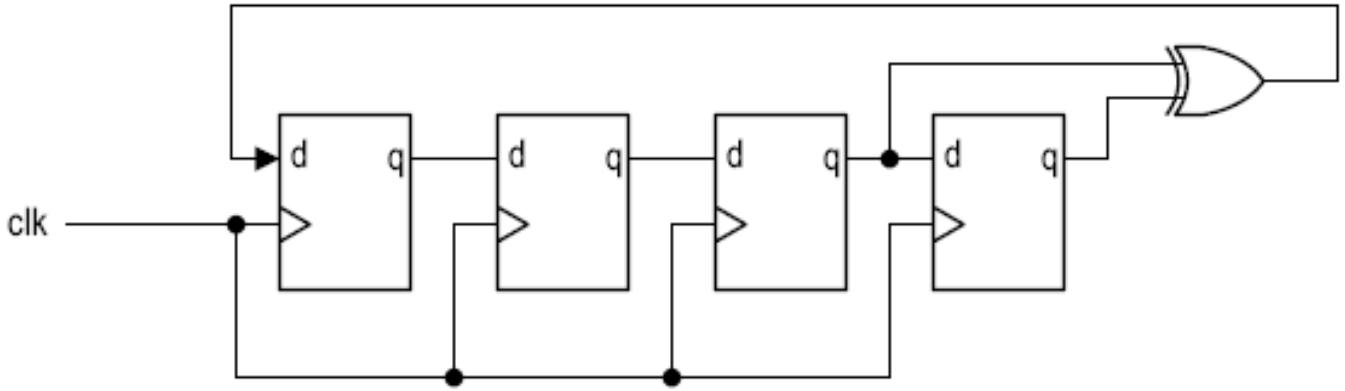
LFSR (Linear Feedback Shift Reg)

- Property of LFSR
  - n-bit LFSR can cycle through 2^n-1 states
  - The feedback circuit always exists
  - The sequence is ***pseudo-random***

- Application of LFSR
  - Pseudorandom: used in **verification**, **testing**, data encryption/decryption

LFSR (Linear Feedback Shift Reg)

- 4-bit LFSR



"1000", "0100", "0010", "1001", "1100", "0110", "1011", "0101", "1010", "1101", "1110", "1111", "0111", "0011", "0001".

```
use ieee.std_logic_1164.all;
entity lfsr4 is
   port(
      clk, reset: in std_logic;
      q: out std_logic_vector(3 downto 0));
end lfsr4;

architecture no_zero_arch of lfsr4 is
   signal r_reg, r_next: std_logic_vector(3 downto 0);
   signal fb: std_logic;
   constant SEED: std_logic_vector(3 downto 0):="0001";
   begin
      -- register
      process (clk,reset)
      begin
         if (reset='1') then
            r_reg <= SEED;
         elsif (clk'event and clk='1') then
            r_reg <= r_next;
         end if;
      end process;
      -- next-state logic
      fb <= r_reg(1) xor r_reg(0);
      r_next <= fb & r_reg(3 downto 1) ;
      -- output logic
      q <= r_reg;
   end no_zero_arch;
```