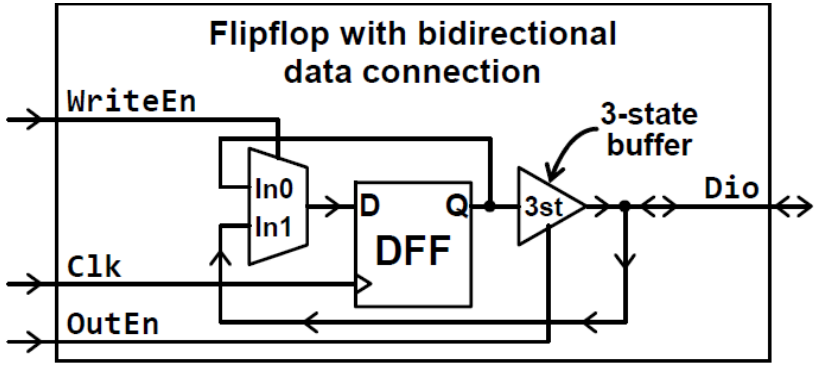
EE342 Lab-7
Instructions and Example Designs

Mehmet Çalı

Bidirectional Data Transfer

- In this experiment you will design 8 bit bidirectional register and accumulator.
- Then you will test your design using the operations listed in the lab sheet.
- Following single bit bidirectional register code is given:

```
module DFF3st(Clk, WriteEn, OutEn, Dio);
input Clk;      // clock input to trigger register storage
input WriteEn;  // enables write operation
input OutEn;    // enables 3-state output buffer for read operation
inout Dio;      // connection for data input and output

reg  FFstore;   // storage register

always @(posedge Clk)
  if (WriteEn == 1'b1)
    FFstore <= Dio;
  else
    FFstore <= FFstore;

assign Dio = (OutEn == 1'b1) ? FFstore : 1'bZ;

endmodule
```



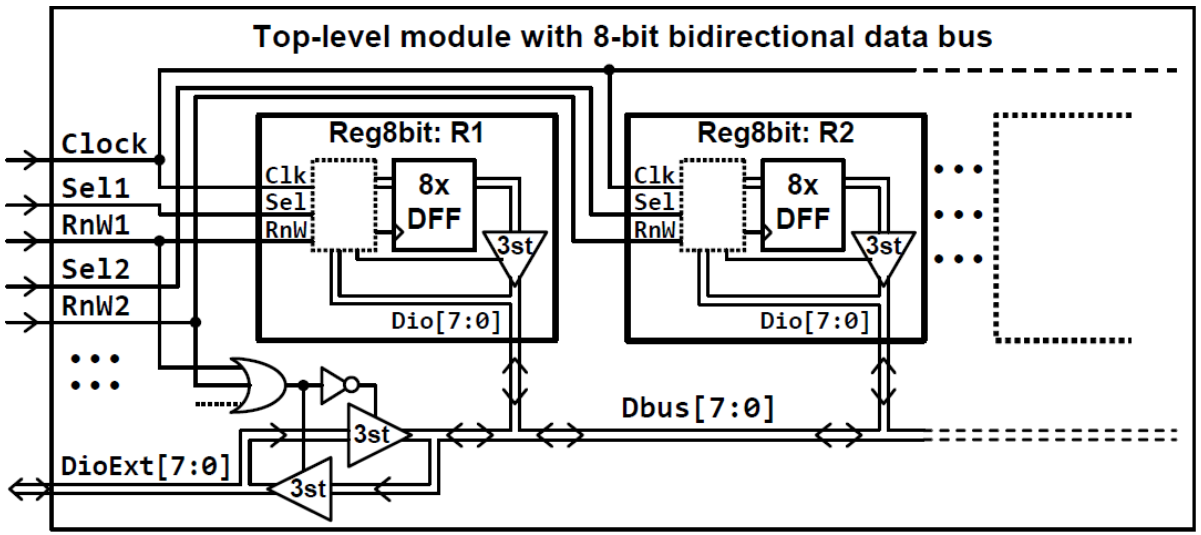
Bidirectional Data Transfer Part-1

- As you need to modify top module and the waveform file after the fifth step of the procedure (accumulator implementation), it is better to create two different projects (one for procedure 1-4, one for procedure 4-7)
- For the first part, you need to modify single bit bidirectional register code to convert it to 8 bit bidirectional register.
- In the single bit version enable signals are separated while writing to flip flop (WriteEn) and reading from flip flop (OutEn), while in the 8 bit version enable signal should be common for both reading and writing operations. Besides reading and writing operations should be determined using separate signals (RnW)

Bidirectional Data Transfer Part-1

- The top module of the first part is also given:

```
module Exp7Top(Clock, Sel1, RnW1, Sel2, RnW2, Sel3, RnW3, DioExt);
input Clock;  // clock input for all modules
input Sel1, Sel2, Sel3;  // module select inputs
input RnW1, RnW2, RnW3;  // module read/write control inputs
inout [7:0] DioExt; // I/O connection to external data pins

tri  [7:0] Dbus;  // internal data bus connecting all modules and
                  // DioExt[7:0] external data pins
Reg8bit R1(Clock, Sel1, RnW1, Dbus);  // register modules
Reg8bit R2(Clock, Sel2, RnW2, Dbus);
Reg8bit R3(Clock, Sel3, RnW3, Dbus);

// DioExt[7:0] drive Dbus[7:0] while writing to a register module.
assign Dbus[7:0] = ( (RnW1 | RnW2 | RnW3) == 1'b0 ) ?
                   DioExt[7:0] : 8'bZ;
// Dbus[7:0] drive DioExt[7:0] while reading from a register module.
assign DioExt[7:0] = ( (RnW1 | RnW2 | RnW3) == 1'b1 ) ?
                     Dbus[7:0] : 8'bZ;
endmodule
```

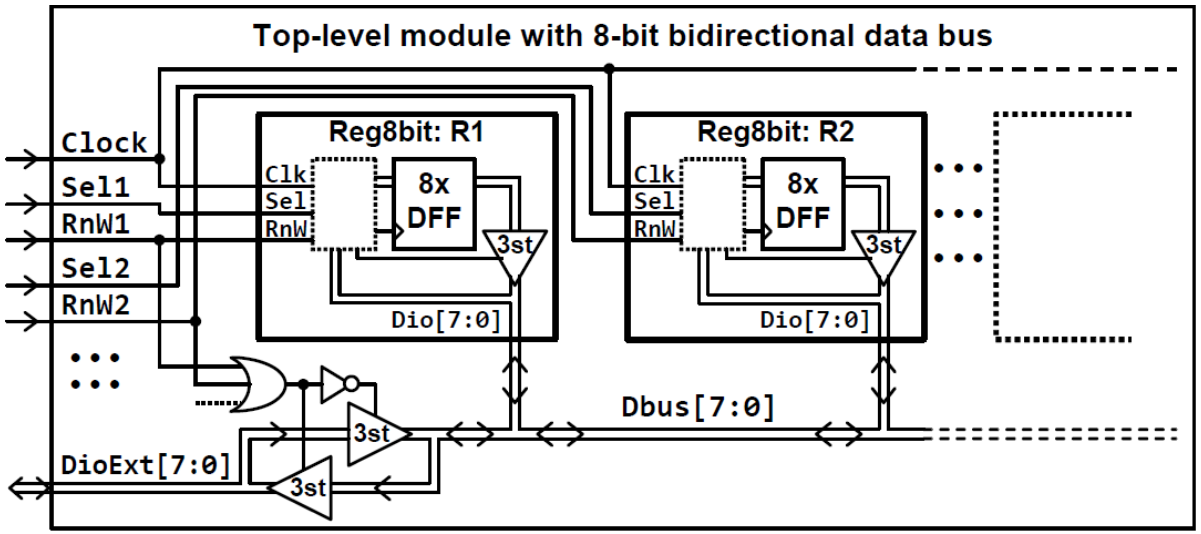


Part-1 Waveform

As the final step of the first part, you should adjust your waveform file to follow these operations precisely with the given order:

- Op1: write 0x11 DioExt to R1
- Op2: write 0x22 DioExt to R2
- Op3: write 0x33 DioExt to R3
- Op4: read the content of R1 at DioExtOut
- Op5: read the content of R2 at DioExtOut
- Op6: read the content of R3 at DioExtOut
- Op7: copy the content of R3 to R1
- Op8: copy the content of R2 to R3

You should adjust select and RnW inputs such that the operations above is realized without causing any logic contentions.



Bidirectional Data Transfer Part-2

- Create a separate project for the procedure step-5 to step-7 to make your evaluation easier.
- You should create an 8 bit Accumulator with the same inputs and outpus as the 8 bit bidirectional register.
- If the accumulator is selected
  - The accumulator should add data Dio[7:0] to accumulator register if RnW is 0
  - The accumulator content should be transferred to Dio[7:0] if RnW is 1 (After the reading the accumulator content should be cleared in the next clock cycle)
  - If the accumulator is read in one clock cycle and it is written in the next clock cycle, the data to be written should directly be transfered to accumulator register in the second clock cycle (procedure step 7)
- The new top module could be modified from the previous part such that an accumulator instance is added besides the previous three registers. The accumulator should have separate select and RnW signals. The top module of previous part should be further modified such that these signals will not couse any logic contentions.

Part-2 Waveform

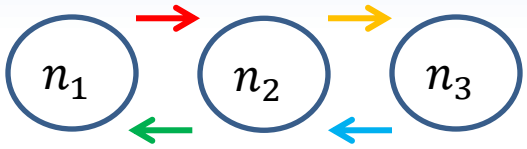
Finally, the waveforms of the second part should be adjusted to follow the below operation sequence.

- Op1: write 0x11 DioExt to R1
- Op2: write 0x22 DioExt to R2
- Op3: write 0x33 DioExt to R3
- Op4: add R2 register contents (0x22) to the accumulator
- Op5: add R3 register contents (0x33) to the accumulator
- Op6: add Dio[7:0] contents (0x44) to the accumulator
- Op7: copy the accumulator content to R1
- Op8: add Dio[7:0] contents (0x88) to the accumulator

- ➢ Dio input should be set as specified at Op6 and Op8 in the corresponding time instances.
- ➢ As Op7 involves reading and Op8 which should be placed in the next clock cycle involves writing, the contents of Dio should directly be written to accumulator. (Op7 requires reset while Op8 requires adding at the same clock cycle, so to avoid conflict this special case should be handled by directly transferring the data to accumulator)

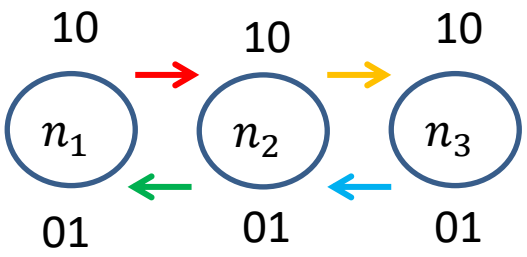
Case Study: Network Coding

- Relay channel: Channels with nodes that are used to forward incoming data to distances further than the initial node transmission range
- Two-Way Relay Channel (TWRC):
  - has only two way communication
  - consists of 3 nodes

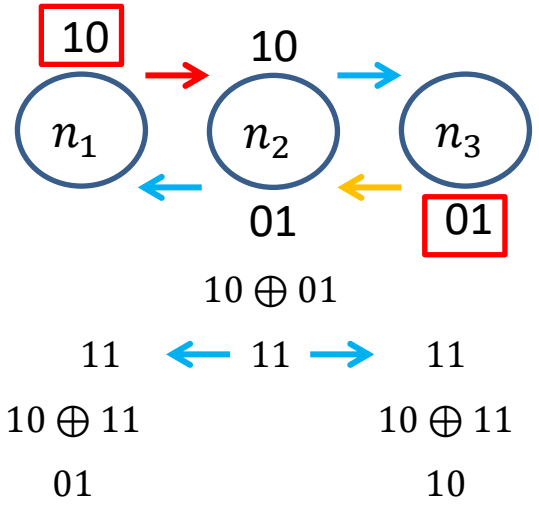


<u>Bidirectional data transfer in Network Layer:</u>

Without network coding:

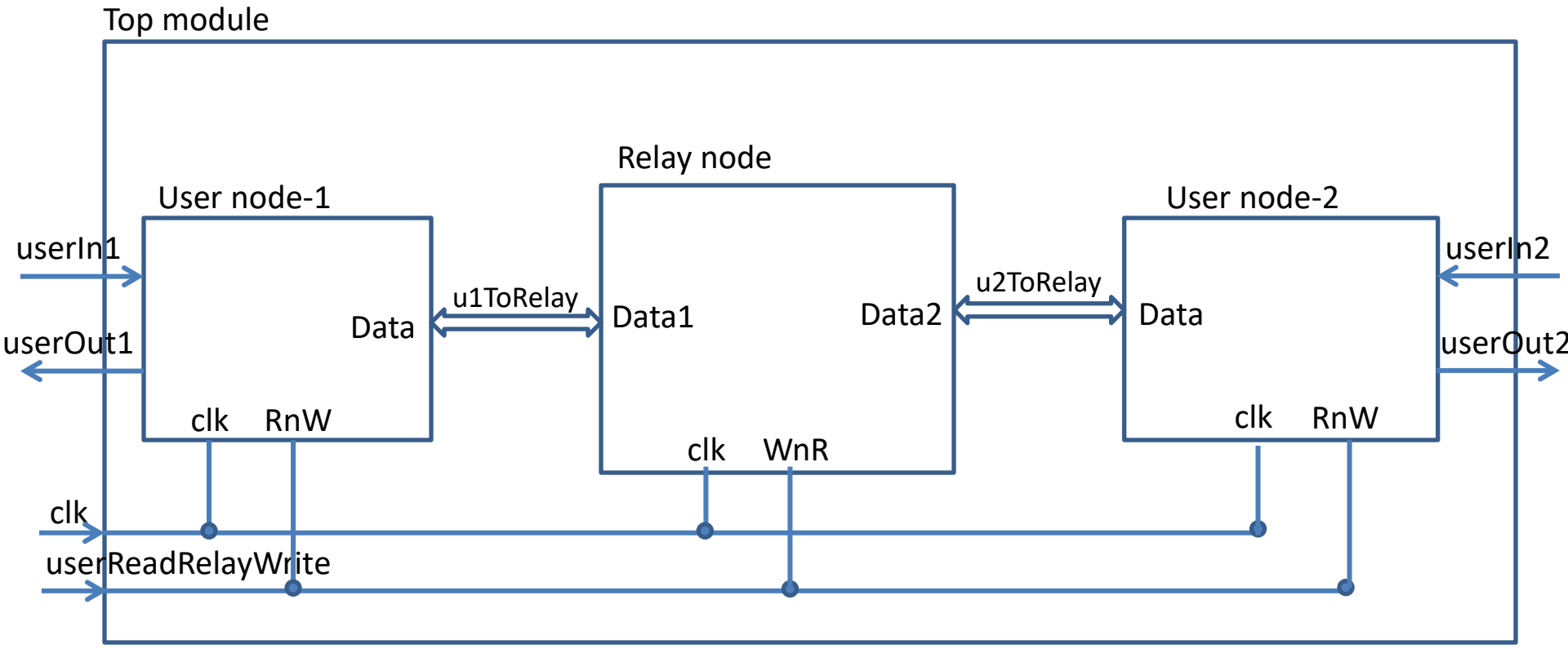


Network Coding in Network Layer:



Bidirectional Data Trasnfer in TWRC

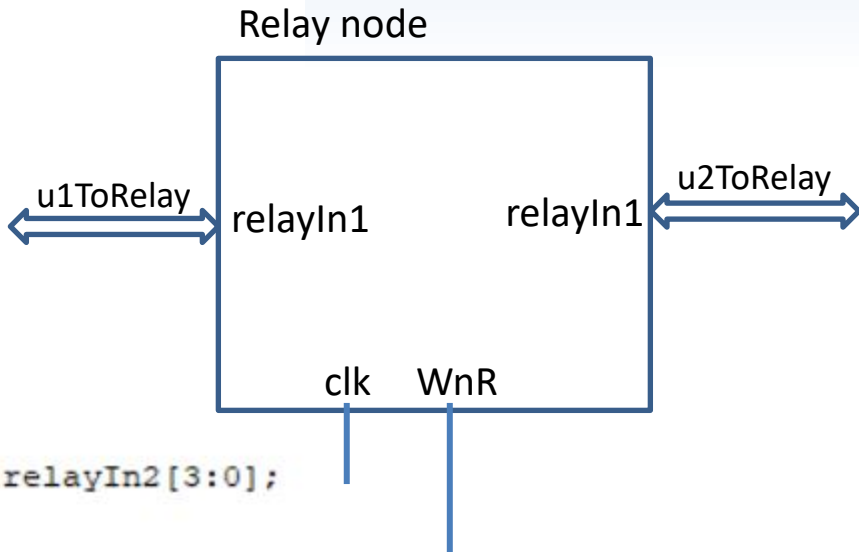
- Lets design a system using an anology to network coding basics.

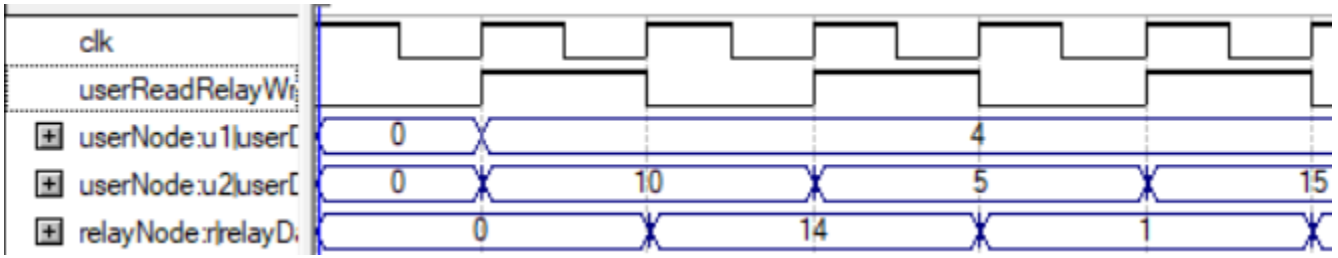


Relay Node

- Lets construct the relay node module first:

```
1  module relayNode
2  (
3      input clk,
4      input WnR,
5      inout [3:0] relayIn1,
6      inout [3:0] relayIn2
7  );
8
9  reg [3:0] relayData;
10 always @ (posedge clk)
11 begin
12     if(WnR==1'b1)
13         relayData[3:0] <= relayIn1[3:0] ^ relayIn2[3:0];
14     else
15         relayData[3:0] <= relayData[3:0];
16 end
17
18 assign relayIn1[3:0] = WnR ? 4'bz : relayData[3:0];
19 assign relayIn2[3:0] = WnR ? 4'bz : relayData[3:0];
20
21 endmodule
```

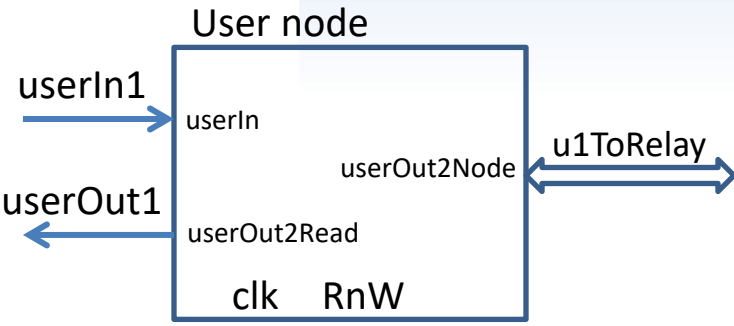


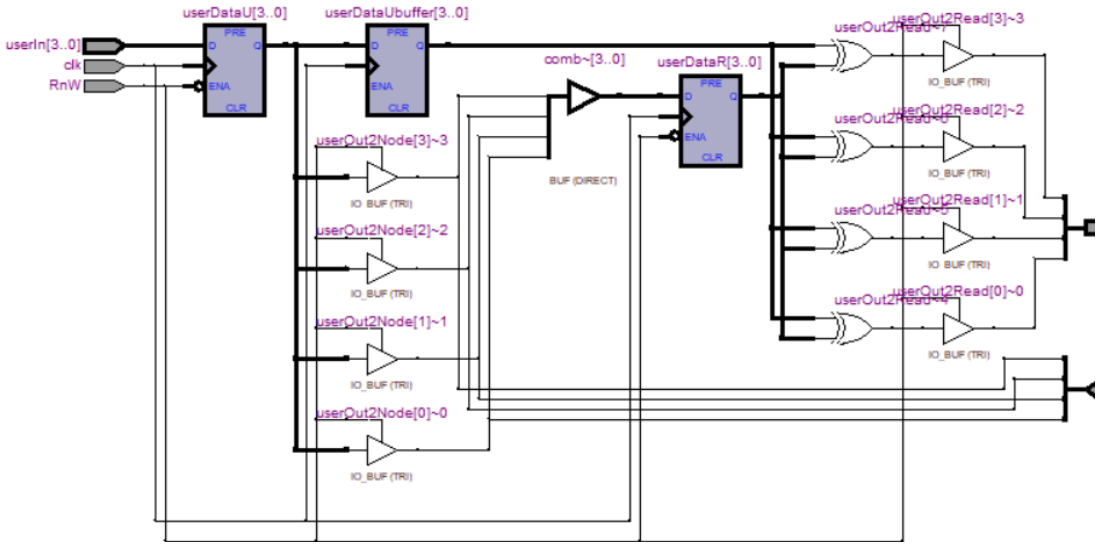


User Node

- Lets construct the user node module

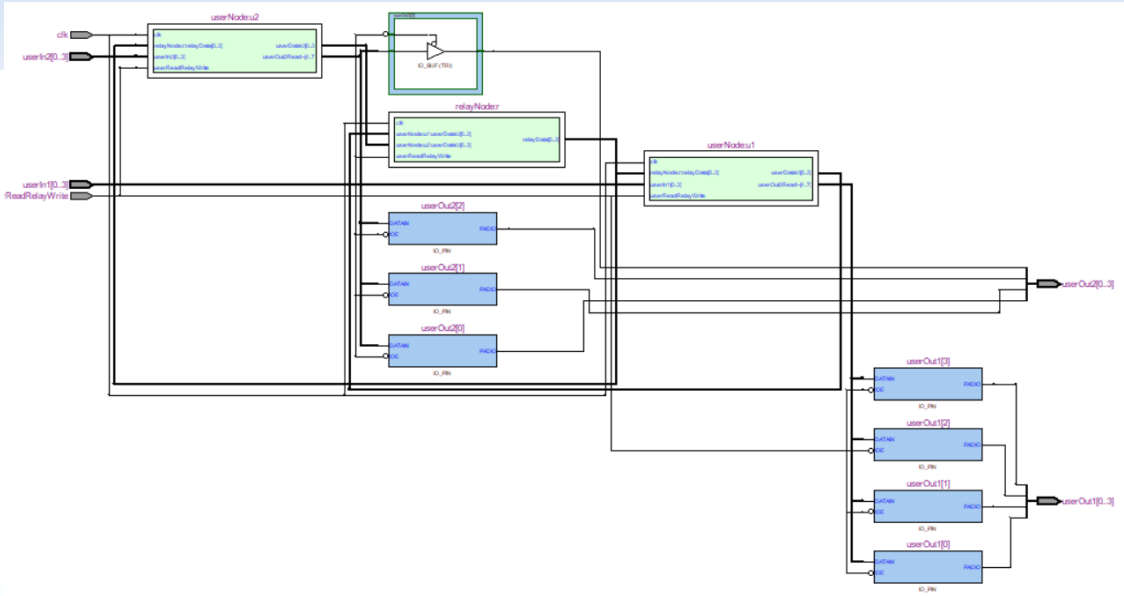
```
module userNode
(
    input clk,
    input RnW,
    input [3:0] userIn,
    inout [3:0] userOut2Node,
    output [3:0] userOut2Read
);
reg [3:0] userDataU, userDataUbuffer, userDataR;

always @ (posedge clk)
begin
    if(RnW==1'b0)
    begin
        userDataR[3:0] <= userOut2Node[3:0];
        userDataU[3:0] <= userIn[3:0];
    end
    else
    begin
        userDataR[3:0] <= userDataR[3:0];
        userDataU[3:0] <= userDataU[3:0];
    end
end
always @ (posedge clk)
    userDataUbuffer[3:0] <= userDataU[3:0];


assign userOut2Node[3:0] = RnW ? userDataU[3:0] : 4'bz;
assign userOut2Read[3:0] = RnW ? (userDataR[3:0] ^ userDataUbuffer[3:0]) : 4'bz;
endmodule
```





Top Module

```
1  module PNCTop
2  (
3       input clk,
4       input userReadRelayWrite,
5       input [3:0] userIn1,
6       input [3:0] userIn2,
7       output [3:0] userOut1,
8       output [3:0] userOut2
9  );
10   wire [3:0] u1ToRelay;
11   wire [3:0] u2ToRelay;
12
13 userNode u1(.clk(clk),
14              .RnW(userReadRelayWrite),
15              .userIn(userIn1),
16              .userOut2Node(u1ToRelay),
17              .userOut2Read(userOut1));
18 userNode u2(.clk(clk),
19              .RnW(userReadRelayWrite),
20              .userIn(userIn2),
21              .userOut2Node(u2ToRelay),
22              .userOut2Read(userOut2));
23 relayNode r(.clk(clk),
24              .WnR(userReadRelayWrite),
25              .relayIn1(u1ToRelay),
26              .relayIn2(u2ToRelay));
27
28   endmodule
```



Overall Test Results