Hardware Design Lab 1 Report

Group Number: 30

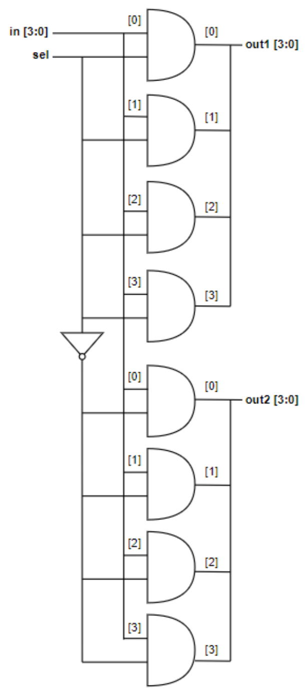
Members: 112062130 侯佑勳
112062326 孔祥光

Contents:

Gate-Level Designs and Explanation **1**
Verification & Testbench **4**
FPGA **6**
What have we learned? **6**
Contribution **7**

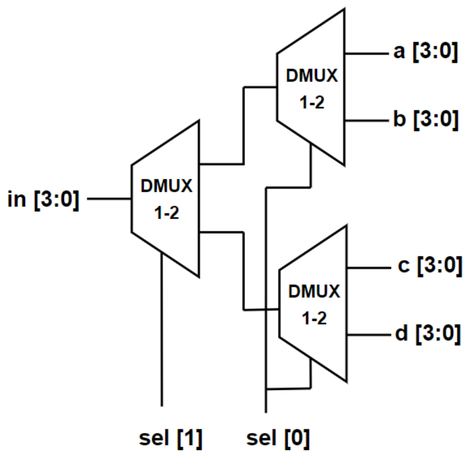
Gate-Level Designs and Explanation

(1) 4-bit 1-to-4 DMUX



∆ **Figure. 1** 1-to-2 DMUX in Gate-Level Design

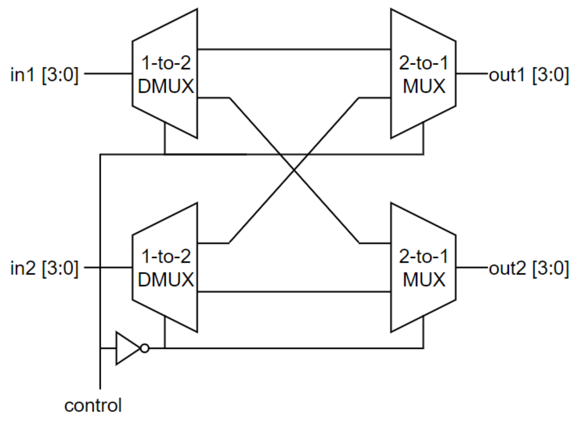
To determine which output should show input data, and which output should be zero, we use one NOT gate and eight AND gates to implement. While the “sel” is set to 1, “out1” will be “in” and “out2” will be 0. Similarly, when the “sel” is set to 0, "out1" will be 0 and "out2" will be "in".



∆ **Figure. 2** 1-to-4 DMUX by Using 3 1-to-2 DMUXes

By using one 1-to-2 DMUX, we can use sel[1] to determine (a,b) or (c,d) to be the “in”. For example, if the sel[1] is 0, the upper DMUX’s input will be “in”, and the others’ will be 0. Continuing the example, the upper DMUX will determine whether a or b to be the “in” based on sel[0].

(2) 4-bit 2x2 Crossbar



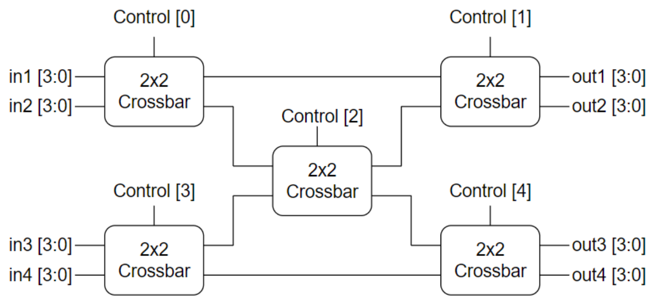
∆ **Figure. 3** 4-bit 2x2 Crossbar

When the control is 0, the upper-left DMUX will choose the upper output to transmit data. The upper-right MUX will receive the data (which is the same as “in1”) by the upper input and transmit it to “out1”. Similarly, The “in2” will transmit to “out2” in the lower part.

When the control is 1, the upper-left DMUX will choose lower output to transmit data, and the lower-right MUX will receive the data (which is same as “in1”) by the upper input and transmit it to “out2”.Similarly, The “in2” will transmit to “out1”.

In simple terms, While the control is 0, we can imagine the data transmits from the crossbar’s upper left to lower right or lower left to upper right. While the control is 1, we can imagine the data transmits from the crossbar’s upper left to upper right or upper left to upper right.

(3) 4-bit 4x4crossbar



∆ **Figure. 4** 4-bit 4x4 Crossbar

We combine 5 crossbars and each crossbar has 1-bit control. Through giving different control, we can determine where each input should transmit to the output. However, some different controls have the same result. Note that 4 out of the 24(4!) results can't be achieved.

[(in1, out3), (in2, out4), (in3, out1), (in4, out2)])

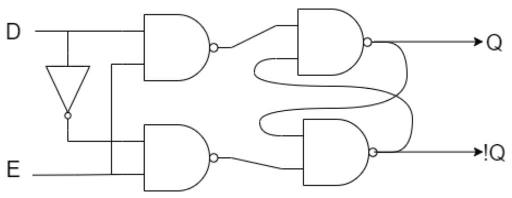
[(in1, out3), (in2, out4), (in3, out2), (in4, out1)])

[(in1, out4), (in2, out3), (in3, out1), (in4, out2)])

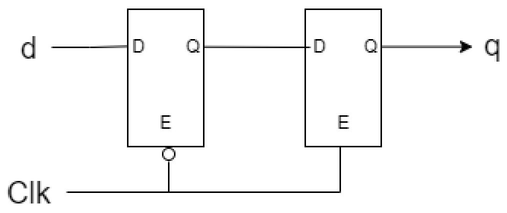
[(in1, out4), (in2, out3), (in3, out2), (in4, out1)])

(4) 1-bit toggle flip flop (TFF)

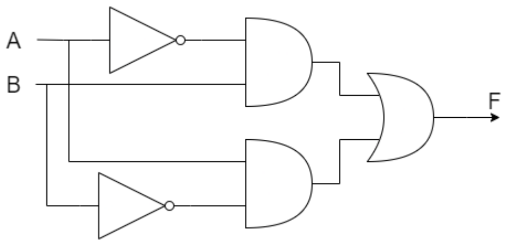
We first construct a clock-positive D Flip-Flop with 2 D-Latches, then make a Toggle Flip-Flop out of it, using an AND gate and a XOR gate (not directly using the primitive one).



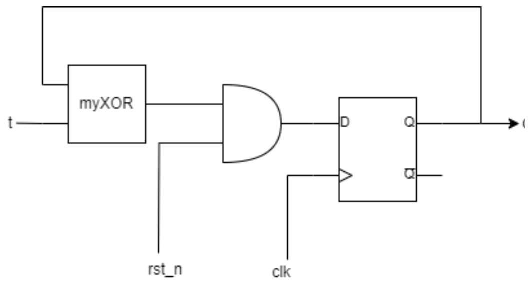
∆ **Figure. 5** D-Latch



∆ **Figure. 6** D Flip-Flop



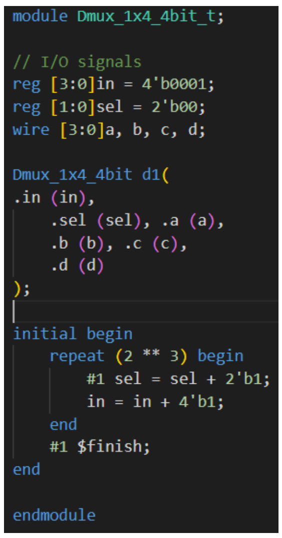
∆ **Figure. 7** non-primitive XOR



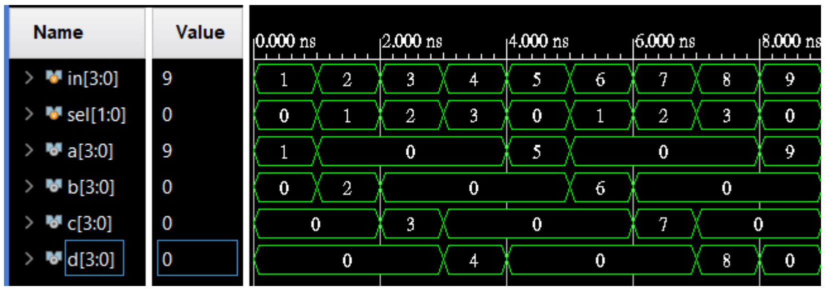
∆ **Figure. 8** T Flip-Flop

Verification & Testbench

(1) 4-bit 1-to-4 DMUX



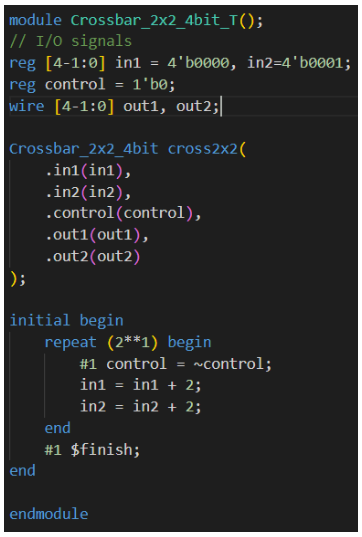
```
module Dmux_1x4_4bit_t;

// I/O signals
reg [3:0]in = 4'b0001;
reg [1:0]sel = 2'b00;
wire [3:0]a, b, c, d;

Dmux_1x4_4bit d1(
.in (in),
    .sel (sel), .a (a),
    .b (b), .c (c),
    .d (d)
);

initial begin
    repeat (2 ** 3) begin
        #1 sel = sel + 2'b1;
        in = in + 4'b1;
    end
    #1 $finish;
end

endmodule
```



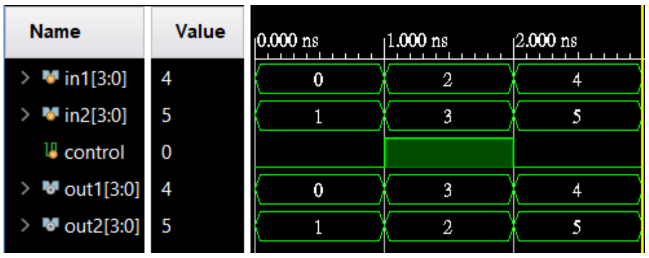
∆ **Figure. 9** 1-4 DMUX Testbench & Waveform

We set all the output's initial value to zero, and let input increase 1 every cycle. We also let sel increase 1 every cycle to determine which outputs will receive the data from input. From the Waveform, we can clearly know the relation between sel and output(0→a, 1→b, 2→c, 3→d).

(2) 4-bit 2x2 Crossbar



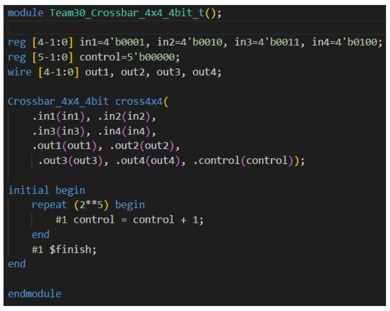
```
module Crossbar_2x2_4bit_T();
// I/O signals
reg [4-1:0] in1 = 4'b0000, in2=4'b0001;
reg control = 1'b0;
wire [4-1:0] out1, out2;

Crossbar_2x2_4bit cross2x2(
    .in1(in1),
    .in2(in2),
    .control(control),
    .out1(out1),
    .out2(out2)
);

initial begin
    repeat (2**1) begin
        #1 control = ~control;
        in1 = in1 + 2;
        in2 = in2 + 2;
    end
    #1 $finish;
end

endmodule
```



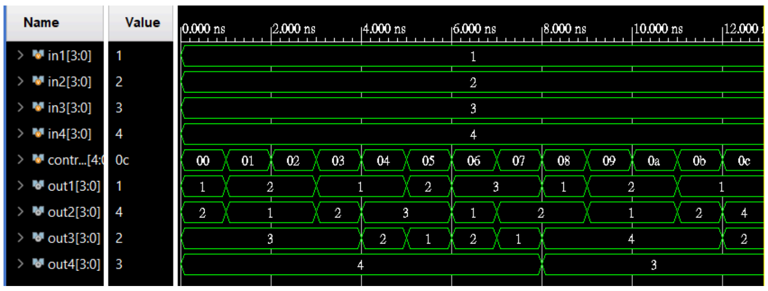
△ **Figure. 10** 2x2 Crossbar Testbench & Waveform

We ensure that the value of in1 and in2 are always different, then check if the correspondence between outputs and inputs are as described by control.

(3) 4-bit 4x4 Crossbar



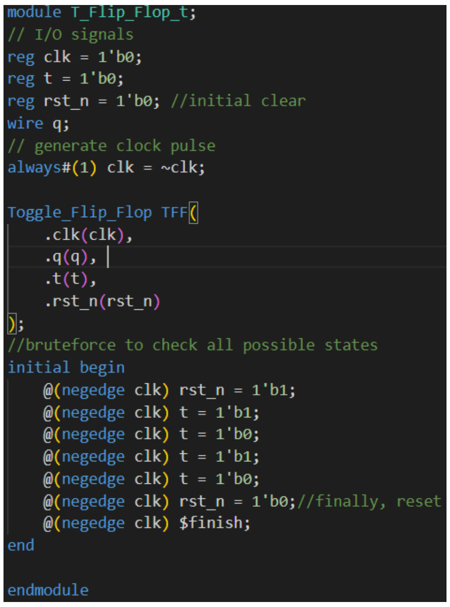
```
module Team30_Crossbar_4x4_4bit_t();

reg [4-1:0] in1=4'b0001, in2=4'b0010, in3=4'b0011, in4=4'b0100;
reg [5-1:0] control=5'b00000;
wire [4-1:0] out1, out2, out3, out4;

Crossbar_4x4_4bit cross4x4(
    .in1(in1), .in2(in2),
    .in3(in3), .in4(in4),
    .out1(out1), .out2(out2),
     .out3(out3), .out4(out4), .control(control));

initial begin
    repeat (2**5) begin
        #1 control = control + 1;
    end
    #1 $finish;
end

endmodule
```



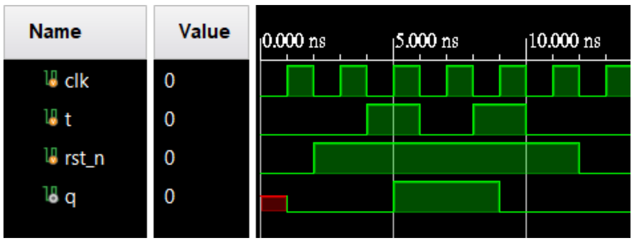
△ **Figure. 11** 4x4 Crossbar Testbench & Waveform

We set the value of in1~in4 to 1~4, respectively, to easily see which output is connected to which input. Then exhaust all the 32 possible settings of control to verify (since there are 5 bits).

(4) 1-bit Toggle Flip Flop (TFF)



```verilog
module T_Flip_Flop_t;
// I/O signals
reg clk = 1'b0;
reg t = 1'b0;
reg rst_n = 1'b0; //initial clear
wire q;
// generate clock pulse
always#(1) clk = ~clk;

Toggle_Flip_Flop TFF(
    .clk(clk),
    .q(q),
    .t(t),
    .rst_n(rst_n)
);
//bruteforce to check all possible states
initial begin
    @(negedge clk) rst_n = 1'b1;
    @(negedge clk) t = 1'b1;
    @(negedge clk) t = 1'b0;
    @(negedge clk) t = 1'b1;
    @(negedge clk) t = 1'b0;
    @(negedge clk) rst_n = 1'b0;//finally, reset
    @(negedge clk) $finish;
end

endmodule
```



∆ **Figure. 12** TFF Testbench & Waveform

What have we learned?

112062130 侯佑勳 :

In this LAB, I got familiar with how to implement MUX and DMUX and learned how to utilize those two to create a crossbar. I also have gained the ability to create an appropriate testbench to test my module. I think a great testbench is not only comprehensive but can also generate a good waveform to observe.

112062326 孔祥光 :

Through this LAB, we have started by constructing the fundamental building blocks of hardware design, such as MUXes and Flip-Flops, using nothing but gate-level design. And we gradually assemble those to form much more complex modules. We’ve also familiarized ourselves with the basic workflow of Vivado and the simulation framework of Verilog.

Contribution

112062130 侯佑勳:
1-to-4 DMUX (+ Testbench)
2x2 Crossbar
4x4 Crossbar
CAD simulation

112062326 孔祥光:
2x2 Crossbar’s Testbench
4x4 Crossbar’s Testbench
TFF (+ Testbench)
2x2 Crossbar on FPGA