# Assignment 3 : Structural Model

Ngo Van Canh

October 8, 2023

## Problem 1 : Write testbench for all problems in Assignment 2

### 1. Problem 1

```
1  module Problem1_tb;
2  reg[0:4] I;
3  wire F;
4  Problem1 p1(F, I[0], I[1], I[2], I[3]);
5  initial begin
6      for(I=0; I<16; I=I+1) begin
7          #5;
8          $display("%t I=%b  F=%b", $time, I, F)
9      end
10 
11 end
12 endmodule
```

Listing 1: Problem 1 testbench

Notice how the input I is defined to be 5 bits wide (line 2). This is to prevent the for loop (line 6) looping infinitely. Because if I is only 4bits, the variable itself never exceed 15. This is completely optional, alternatively, you can manually break the simulation at a desire time point, by typing the command "`run 100`" in the transcript command windows (100 is the duration you want to run, change it to suit the simulation)
The above testbench should output something like following:

```
#                        5 I=00000  F=1
#                       10 I=00001  F=1
#                       15 I=00010  F=0
#                       20 I=00011  F=0
#                       25 I=00100  F=1
#                       30 I=00101  F=1
#                       35 I=00110  F=1
#                       40 I=00111  F=1
#                       45 I=01000  F=0
#                       50 I=01001  F=0
#                       55 I=01010  F=0
#                       60 I=01011  F=0
#                       65 I=01100  F=0
#                       70 I=01101  F=0
#                       75 I=01110  F=0
#                       80 I=01111  F=0
```

Listing 2: Problems 1 Testbench Output

## 2. 4-to-1 Multiplexer

```
module mux4_to_1_tb;
reg IN0, IN1, IN2, IN3;
reg S1, S0;
wire OUTPUT;
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
initial begin
    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
    #5 $display("%t IN0= %b, IN1= %b, IN2= %b, IN3= %b", $time, IN0,IN1,IN2,
   IN3);
    S1 = 0; S0 = 0;
    #5 $display("%t S1 = %b, S0 = %b, OUTPUT = %b", $time, S1, S0, OUTPUT);
    S1 = 0; S0 = 1;
    #5 $display("%t S1 = %b, S0 = %b, OUTPUT = %b", $time, S1, S0, OUTPUT);
    S1 = 1; S0 = 0;
    #5 $display("%t S1 = %b, S0 = %b, OUTPUT = %b", $time, S1, S0, OUTPUT);
    S1 = 1; S0 = 1;
    #5 $display("%t S1 = %b, S0 = %b, OUTPUT = %b", $time, S1, S0, OUTPUT);
end
endmodule
```

Listing 3: 4-to-1 Multiplexer testbench

```
#                    5 IN0= 1, IN1= 0, IN2= 1, IN3= 0
#                   10 S1 = 0, S0 = 0, OUTPUT = 1
#                   15 S1 = 0, S0 = 1, OUTPUT = 0
#                   20 S1 = 1, S0 = 0, OUTPUT = 1
#                   25 S1 = 1, S0 = 1, OUTPUT = 0
```

Listing 4: 4-to-1 Multiplexer Testbench Output

## 3. Half Adder

```
module half_adder_tb;
reg X, Y;
wire S, C;
half_adder ha(X, Y, S, C);
initial begin
    $monitor($time," X= %b, Y=%b, S= %b, C= %b", X, Y, S, C);
end
initial begin
    #0 X=0; Y=0 ;
    #5 X=0; Y=1;
    #5 X=1; Y=0;
    #5 X=1; Y=1;
end
endmodule
```

Listing 5: Half Adder testbench

```
#                    0 X= 0, Y=0, S= 0, C= 0
#                    5 X= 0, Y=1, S= 1, C= 0
#                   10 X= 1, Y=0, S= 1, C= 0
#                   15 X= 1, Y=1, S= 0, C= 1
```

Listing 6: Half Adder Testbench Output

## 4. Full Adder

```
module full_adder_tb;
reg X, Y, Ci;
wire S, Co;
full_adder fa(X, Y, Ci, S, Co);
initial begin
    $monitor($time," X= %b, Y=%b, Ci= %b, S= %b, Co= %b", X, Y, Ci, S, Co);
end
initial begin
    #0 X=0; Y=0; Ci=0;
    #5 X=0; Y=0; Ci=1;
    #5 X=0; Y=1; Ci=0;
    #5 X=0; Y=1; Ci=1;
    #5 X=1; Y=0; Ci=0;
    #5 X=1; Y=0; Ci=1;
    #5 X=1; Y=1; Ci=0;
    #5 X=1; Y=1; Ci=1;
end
endmodule
```

Listing 7: Full Adder testbench

```
#                    0 X= 0, Y=0, Ci= 0, S= 0, Co= 0
#                    5 X= 0, Y=0, Ci= 1, S= 1, Co= 0
#                   10 X= 0, Y=1, Ci= 0, S= 1, Co= 0
#                   15 X= 0, Y=1, Ci= 1, S= 0, Co= 1
#                   20 X= 1, Y=0, Ci= 0, S= 1, Co= 0
#                   25 X= 1, Y=0, Ci= 1, S= 0, Co= 1
#                   30 X= 1, Y=1, Ci= 0, S= 0, Co= 1
#                   35 X= 1, Y=1, Ci= 1, S= 1, Co= 1
```

Listing 8: Full Adder Testbench Output

## 5. 4-bit Ripple Carry Adder

```
module Ripple_Carry_Adder_tb;
reg [3:0] X, Y;
reg Ci;
wire [3:0] S;
wire Co;
Ripple_Carry_Adder RCA(X, Y, Ci, S, Co);
initial begin
    $monitor($time," X= %b, Y=%b, Ci= %b, S= %b, Co= %b", X, Y, Ci, S, Co);
end
initial begin
    #0 X = 4'd0; Y = 4'd0; Ci = 1'b0;
    #5 X = 4'd3; Y = 4'd4;
    #5 X = 4'd2; Y = 4'd5;
    #5 X = 4'd9; Y = 4'd9;
    #5 X = 4'd10; Y = 4'd15;
    #5 X = 4'd10; Y = 4'd5; Ci = 1'b1;
end
endmodule
```

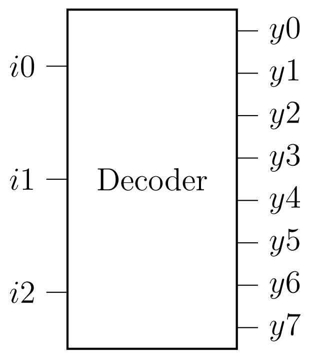
Listing 9: 4-bit Ripple Carry Adder testbench

```
#                    0 X= 0000, Y=0000, Ci= 0, S= 0000, Co= 0
#                    5 X= 0011, Y=0100, Ci= 0, S= 0111, Co= 0
#                   10 X= 0010, Y=0101, Ci= 0, S= 0111, Co= 0
#                   15 X= 1001, Y=1001, Ci= 0, S= 0010, Co= 1
#                   20 X= 1010, Y=1111, Ci= 0, S= 1001, Co= 1
#                   25 X= 1010, Y=0101, Ci= 1, S= 0000, Co= 1
```

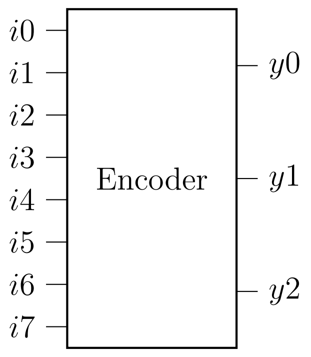
Listing 10: Full Adder Testbench Output

## Problem 2 : 8-to-3 Encoder and 3-to-8 Decoder

1. Construct truth table.
2. Determine output function.
3. Write Verilog code and testbench for that circuit.



(a) 3-to-8 Decoder



(b) 8-to-3 Encoder

Figure 1: Encoder and Decoder

### a. Decoder

| $i0$ | $i1$ | $i2$ | $y0$ | $y1$ | $y2$ | $y3$ | $y4$ | $y5$ | $y6$ | $y7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Table 1: Decoder 3-to-8 Truth Table

$$\Rightarrow \begin{cases} y0 = i0'i1'i2' \\ y1 = i0'i1'i2 \\ y2 = i0'i1i2' \\ y3 = i0'i1i2 \\ y4 = i0i1'i2' \\ y5 = i0i1'i2 \\ y6 = i0i1i2' \\ y7 = i0i1i2 \end{cases}$$

```
module Decoder_3_to_8 (
    input[0:2] i,
    output[0:7] y
);


endmodule
```

Listing 11: Decoder 3-to-8 and testbench

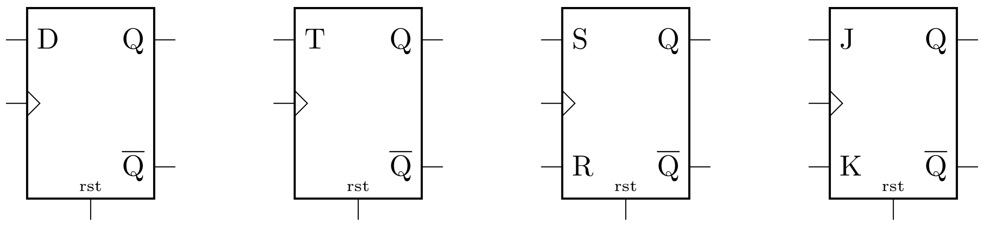
```
    #
    #
    #
    #
    #
    #

```

Listing 12: Full Adder Testbench Output

## b. Encoder

## Problem 3 : Flip-flops : D-FF, T-FF, JK-FF, SR-FF

1. Construct truth table.
2. Determine output function.
3. Write Verilog code and testbench for that circuit.



(a) D Flip-flop (b) T Flip-flop (c) SR Flip-flop (d) JK Flip-flop

Figure 2: Flip-flops

*NOTE : Using structural model. Flip flops include reset signal.*