Lab12

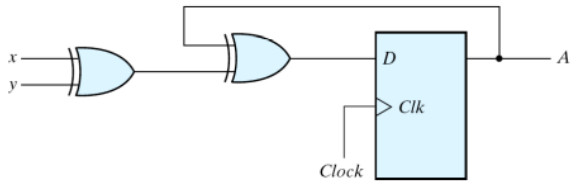
實驗主題: Analysis of Synchronous Sequential Logic

實驗日期:B103040009 尹信淳

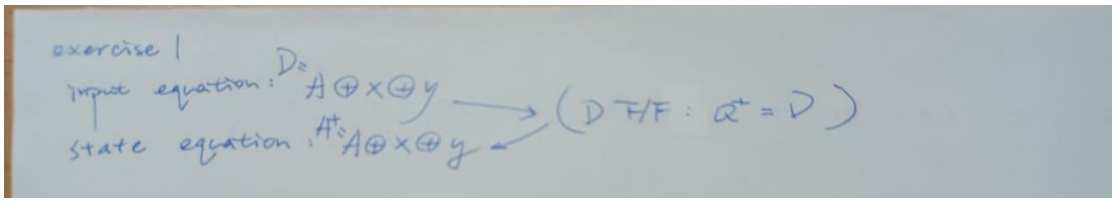
實驗內容: Analyze and Verify the circuits using Verilog HDL

Exercise1:

Simulate the sequential circuit shown in Fig. 5.17

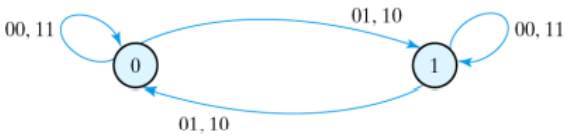



Input equation & state equation:




State table & state diagram:

| Present state | Inputs | | Next state |
|---|---|---|---|
| *A* | *x* | *y* | *A* |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |




Code:

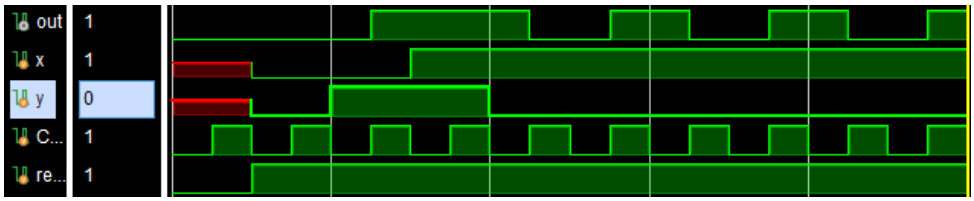
Behavioral modeling:

```
module exercise1(output out,input x,y,Clock,reset);
    reg A;
    parameter S0=1'b0, S1=1'b1;
    always @ (posedge Clock,negedge reset)
        if(reset==0)    A <= S0;
        else case(A)
           S0:if(x==0&&y==0 || x==1&&y==1)  A<=A;else   A<=S1;
           S1:if(x==0&&y==0 || x==1&&y==1)  A<=A;else   A<=S0;
        endcase
        assign out = A;
endmodule
```

Testbench:

```
module tb;
    wire out;
    reg x,y,Clock,reset;
    exercise1 UUT (.out(out),.x(x),.y(y),.Clock(Clock),.reset(reset));
    initial #100 $finish;
    initial begin Clock = 0; forever #5 Clock = ~Clock; end
    initial fork
    reset=0;
    #10 reset=1;
    #10 {x,y}=2'b00;
    #20 {x,y}=2'b01;
    #30 {x,y}=2'b11;
    #40 {x,y}=2'b10;
    join
endmodule
```

Result:



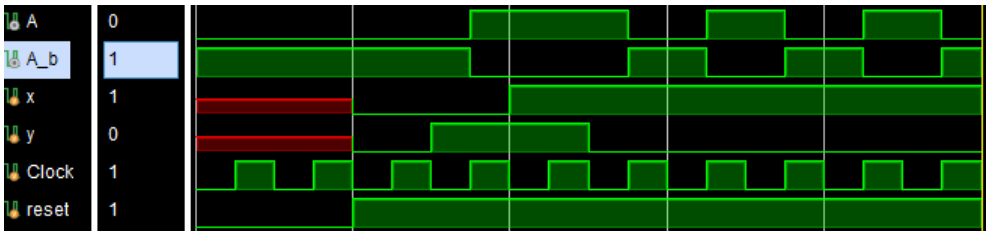

Structural modeling:

```
module DFF(Q,Q_b,D,Clk,reset);
    output Q,Q_b;
    input D,Clk,reset;
    wire n1,n2,n3,n4;
    nand(n1,n4,n2);
    nand(n2,Clk,n1,reset);
    nand(n3,n2,Clk,n4);
    nand(n4,n3,D,reset);
    nand(Q,n2,Q_b);
    nand(Q_b,n3,Q,reset);
endmodule
module e1(A,A_b,x,y,Clock,reset);
    input x,y,Clock,reset;
    output A,A_b;
    wire n1;
    xor (n1,x,y,A);
    DFF d1(A,A_b,n1,Clock,reset);
endmodule
```

Testbench:

```
module tb;
wire A,A_b;
reg x,y, Clock, reset;
e1 UUT (.A(A),.A_b(A_b), .x(x),.y(y), .Clock(Clock), .reset(reset));
initial #100 $finish;
initial begin Clock = 0; forever #5 Clock = ~Clock; end
    initial fork
        reset = 0;
        #20 reset = 1;
        #20 {x,y} = 2'b00;
        #30 {x,y} = 2'b01;
        #40 {x,y} = 2'b11;
        #50 {x,y} = 2'b10;
        #60 reset = 0;
        #70 {x,y} = 2'b00;
        #80 {x,y} = 2'b01;
        #90 {x,y} = 2'b11;
        #100 {x,y} = 2'b10;
    join
endmodule
```

Result:



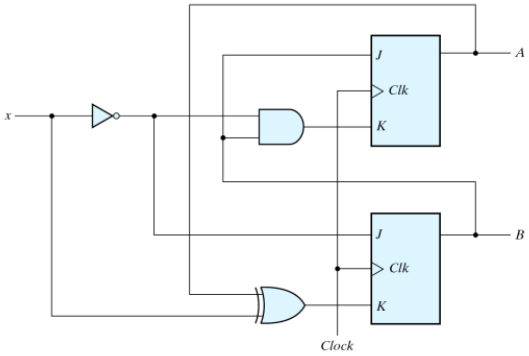

Exercise1 實驗結果與分析:

功能正確，且兩種寫法輸出相同。

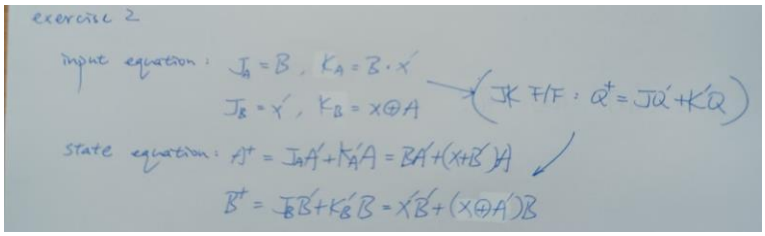
---

Exercise2:

Simulate the sequential circuit shown in Fig. 5.18



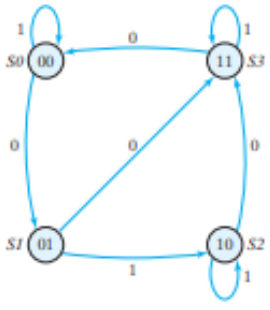

Input equation & state equation:




State table & state diagram:

**Table 5.4**
*State Table for Sequential Circuit with JK Flip-Flops*

| Present State | | Input | Next State | | Flip-Flop Inputs | | | |
|---|---|---|---|---|---|---|---|---|
| A | B | x | A | B | $J_A$ | $K_A$ | $J_B$ | $K_B$ |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |




Code:

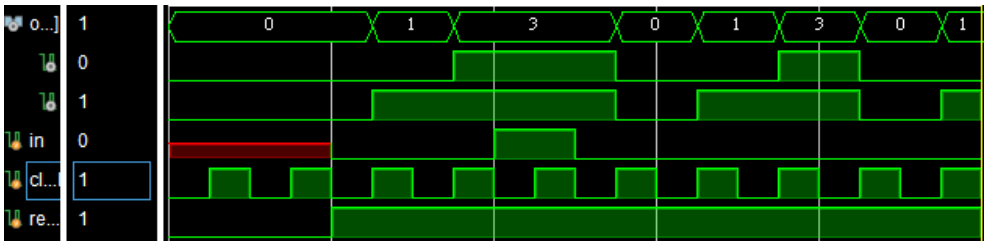
Behavioral modeling:

```
module e2(output[1:0] out,input in,clock,reset);
    reg[1:0] state;
    parameter S0 = 2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
    always @ (posedge clock,negedge reset)
    if(reset==0)    state<=S0;
    else case(state)
        S0:if(!in)  state<=S1; else state<=S0;
        S1:if(in)   state<=S2; else state<=S3;
        S2:if(!in)  state<=S3; else state<=S2;
        S3:if(!in)  state<=S0; else state<=S3;
    endcase
    assign out = state;
endmodule
```

## Testbench:

```
module tb;
    wire [1:0] out;
    reg in,clock, reset;
    e2 UUT (.out(out),.in(in), .clock(clock), .reset(reset));
     initial #100 $finish;
    initial begin clock = 0; forever #5 clock = ~clock; end
    initial fork
    reset=0;
    #20 reset = 1;
    #20 in = 0;
    #40 in = 1;
    #50 in= 0;
    #60 in= 0;
    join
endmodule
```

## Result:



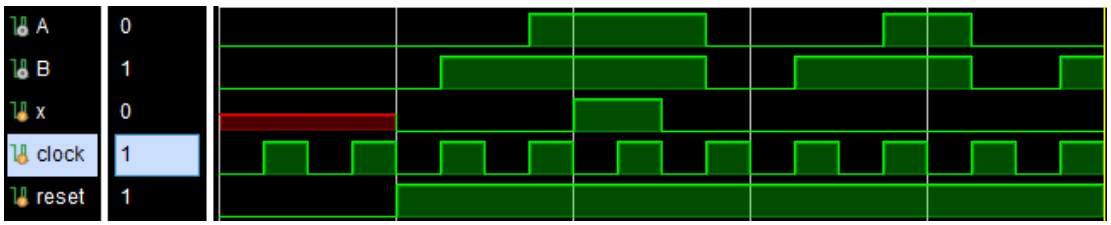

## Structural modeling:

```
module DFF(Q,Q_b,D,Clk,reset);
    output Q,Q_b; input D,Clk,reset;
    wire n1,n2,n3,n4;
    nand(n1,n4,n2);
    nand(n2,Clk,n1,reset);
    nand(n3,n2,Clk,n4);
    nand(n4,n3,D,reset);
    nand(Q,n2,Q_b);
    nand(Q_b,n3,Q,reset);   endmodule
module JKFF(Q,Q_b,J,K,Clk,reset);
    output Q,Q_b; input J,K,Clk,reset;
    wire n1,n2,n3,n4;
    and(n1,J,Q_b);
    not(n2,K);
    and(n3,n2,Q);
    or(n4,n1,n3);
    DFF d1(Q,Q_b,n4,Clk,reset); endmodule
module e2(A,B,x,clock,reset);
    output A,B; input x,clock,reset;
    wire x_not,B_not,n1,n2,n3,n4;
    not(x_not,x);
    and(n1,B,x_not);
    xor(n2,A,x);
    JKFF jk1(A,n3,B,n1,clock,reset);
    JKFF jk2(B,n4,x_not,n2,clock,reset);
endmodule
```

Testbench:

```
module tb;
    wire A,B;
    reg x,clock, reset;
    e2 UUT (.A(A),.B(B),.x(x), .clock(clock), .reset(reset));
     initial #100 $finish;
    initial begin clock = 0; forever #5 clock = ~clock; end
    initial fork
    reset=0;
    #20 reset = 1;
    #20 x = 0;
    #40 x = 1;
    #50 x= 0;
    #60 x= 0;
    join
endmodule
```

Result:




Exercise2 實驗結果與分析:

功能正常，且兩種寫法輸出相同

---

實驗心得

這次實驗開始練習較為複雜的序向電路，兩個 exercise 做的事情是一樣的。拿到電路圖之後，我們要先進行分析: 先得出該電路中 F/F(s)的 input equation 及 next-state equation，隨後再依照該電路使用的 F/F 種類來得出 state diagram 或 state table。如此一來就能依照 state diagram(table)來進行 Behavioral modeling。除此之外，每題還需要用 Strucutral modeling 來驗證兩種 modeling 會得到相同結果。這兩題訓練了序向電路分析步驟，以及如何使用分析出來的 state diagram 來進行 behavioral modeling 的模擬。是個寶貴的實作經驗。