同济大学计算机系

数字逻辑课程实验报告

学　　号 2353900

姓　　名 汪嘉晨

专　　业 计算机科学与技术（精英班）

授课老师 郭玉臣

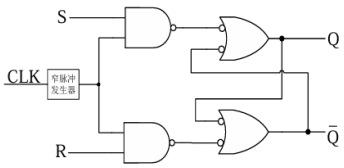
一、实验内容

使用 Verilog HDL 语言实现 D 触发器、JK 触发器以及 PC 寄存器的设计和仿真。

1）触发器

触发器是一种同步双稳态器件，用来记忆一位二进制数。所谓同步，是指触发器的记忆状态按时钟（CLK）规定的启动指示点（脉冲边沿）来改变。下面将介绍几种触发器原理:

a） SR 触发器



输入			输出		说明
S	R	CLK	Q	$\bar{Q}$	
0	0	×	Q^n	$\overline{Q^n}$	保持
0	1	↑	0	1	置 0
1	0	↑	1	0	置 1
1	1	↑	?	?	不稳

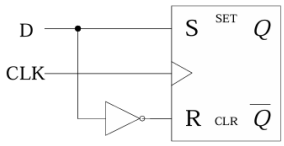
图 6.6.1 SR 触发器逻辑图及其功能表

图 6.6.1 给出了 SR 触发器的逻辑图和其功能表。其中×表示无关，↑表示时钟信号由低到高。数据输入端 S 和 R 称为同步输入，因为这个两个输入的数据只会在时钟脉冲上升沿时被传送到触发器。

- 当 S 为高 R 为低时，Q 输出在时钟脉冲上升沿时变高，触发器置 1。
- 当 S 为低 R 位高时，Q 输出在时钟脉冲上升沿时变低，触发器置 0。
- 当 S 和 R 两者都为低时，Q 输出状态不会发生变化（保持）。
- 当 S 和 R 两者都为高时，Q 输出状态是不稳定的。

b） D 触发器

D 触发器是在 SR 触发器基础上构建的。即在一个 SR 触发器上增加一个非门。图 6.6.2 给出了一个 D 触发器的逻辑图及其功能表。该触发器只有一个数据输入端 D，经反相器反相后，变成互补数据输入，送到 SR 触发器，从而避免了 SR 触发器存在的不稳态问题。当数据输入 D=1，且在时钟脉冲上升沿，触发器置位（1 状态）；当数据输入 D=0，且在时钟脉冲上升沿，触发器复位（0 状态）。

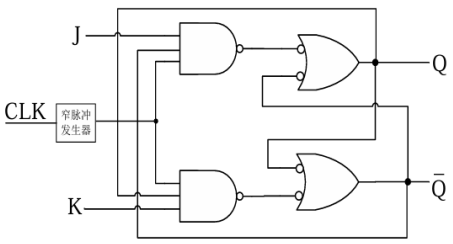


输入		输出		说明
D	CLK	Q	$\bar{Q}$	
1	↑	1	0	置位（存 1）
0	↑	0	1	复位（存 0）

图 6.6.2 D 触发器逻辑图及其功能表

c） JK 触发器

JK 触发器是一种广泛应用的触发器类型。字母 J 和 K 只表示它们是两个数据输入端符号，没有什么特别含义。JK 触发器与 SR 触发器在置位、复位方面的功能是相同的，不同之处在于，JK 触发器改进了 SR 触发器存在的不稳定状态。当 J=1，K=1 时，对每一个连续的时钟脉冲，触发器可改变成相反状态或计数状态，这种工作方式称为交替操作。图 6.6.3 所示为 JK 触发器的内部逻辑及其功能表。



输入			输出		说明
J	K	CLK	Q	$\bar{Q}$	
0	0	↑	Q^n	$\overline{Q^n}$	保持
0	1	↑	0	1	置 0
1	0	↑	1	0	置 1
1	1	↑	Q^n	$\overline{Q^n}$	交替

图 6.6.3 JK 触发器逻辑图及其功能表

2）PC 寄存器

PC 寄存器（program counter）是组成 CPU 的基本部件，用来存放当前正在执行的指令，包括指令的操作码和地址信息。我们知道，D 触发器可以用于存储比特信号。如果 D 为 1，那么在时钟的上升沿，D 触发器的输出 Q 将变为 1。如果 D 为 0，那么在时钟的上升沿，D 触发器的输出 Q 将变为 0。在实时数字系统中，通常 D 触发器的时钟输入端始终有时钟信号输入。这就意味着在每个时钟的上升沿，当前 D 值都将被锁存在 Q 中。在本实验主利用 Verilog HDL 模块实例化基于 D 触发器模块实现一个 PC 寄存器。图 6.6.4 给出了所要建模的 PC 寄存器其功能图。

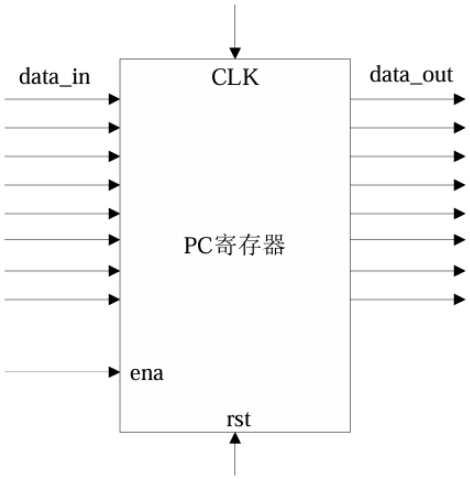


图 6.6.4 PC 寄存器功能图

二、硬件逻辑图

本实验中不做要求。

三、模块建模

1.同步复位 D 触发器

Synchronous_D_FF 模块在时间上升沿到来时根据 rst 和 D 的电平决定输出信号的电平

```
module Synchronous_D_FF(
    input CLK,
    input D,
    input RST_n,
    output reg Q1,
    output reg Q2
    );

    always @(posedge CLK)
        case(RST_n)
            0: begin Q1 = 0; Q2 = 1; end
```

```
            1:
                case(D)
                    0: begin Q1 = 0; Q2 = 1; end
                    1: begin Q1 = 1; Q2 = 0; end
                    default: begin Q1 = 1'bx; Q2 = 1'bx; end
                endcase
            default: begin Q1 = 1'bx; Q2 = 1'bx; end
        endcase
endmodule
```

2.异步复位 D 触发器

Asynchronous_D_FF 模块在时间上升沿来临时根据 D 的电平信号决定输出信号的电平。特别的，当 RST_n 信号为低电平时，输出信号立即复位

```
module Asynchronous_D_FF(
    input CLK,
    input D,
    input RST_n,
    output reg Q1,
    output reg Q2
    );

    always @(posedge CLK or negedge RST_n)
        if(RST_n == 0)
            {Q1, Q2} = 2'b01;
        else if(RST_n == 1)
            case(D)
                0: {Q1, Q2} = 2'b01;
                1: {Q1, Q2} = 2'b10;
                default: {Q1, Q2} = 2'bxx;
            endcase
        else
            {Q1, Q2} = 2'bxx;
endmodule
```

3.异步复位 JK 触发器

Asynchronous_D_FF 模块在时间上升沿来临时根据 J、K 的电平信号决定输出信号的电平。特别的，当 RST_n 信号为低电平时，输出信号立即复位

```
module JK_FF(
    input CLK,
    input J,
    input K,
    input RST_n,
    output reg Q1,
    output reg Q2
```

```
    );

    always @(posedge CLK or negedge RST_n)
        if(RST_n == 0)
            {Q1, Q2} = 2'b01;
        else if(RST_n == 1)
            case({J, K})
                2'b00: ;
                2'b01: {Q1, Q2} = 2'b01;
                2'b10: {Q1, Q2} = 2'b10;
                2'b11: {Q1, Q2} = {Q2, Q1};
                default: {Q1, Q2} = 2'bxx;
            endcase
        else
            {Q1, Q2} = 2'bxx;
endmodule
```

4.PC 寄存器

由 32 个 2 中提到的异步复位 D 触发器并联（这里不再粘出），能够实现根据 ena 使能信号在时钟上升沿到来时对寄存器进行输入、输出操作，同时能够在 rst 复位信号为低电平时对寄存器进行复位

```
module pcreg(
    input clk,
    input rst,
    input ena,
    input [31:0] data_in,
    output reg [31:0] data_out
    );

    wire [31:0] data_tmp;
    Asynchronous_D_FF b0(.CLK(clk), .D(data_in[0]), .RST_n(~rst), .Q1(data_tmp[0]));
    Asynchronous_D_FF b1(.CLK(clk), .D(data_in[1]), .RST_n(~rst), .Q1(data_tmp[1]));
    Asynchronous_D_FF b2(.CLK(clk), .D(data_in[2]), .RST_n(~rst), .Q1(data_tmp[2]));
    Asynchronous_D_FF b3(.CLK(clk), .D(data_in[3]), .RST_n(~rst), .Q1(data_tmp[3]));
    Asynchronous_D_FF b4(.CLK(clk), .D(data_in[4]), .RST_n(~rst), .Q1(data_tmp[4]));
    Asynchronous_D_FF b5(.CLK(clk), .D(data_in[5]), .RST_n(~rst), .Q1(data_tmp[5]));
    Asynchronous_D_FF b6(.CLK(clk), .D(data_in[6]), .RST_n(~rst), .Q1(data_tmp[6]));
    Asynchronous_D_FF b7(.CLK(clk), .D(data_in[7]), .RST_n(~rst), .Q1(data_tmp[7]));
    Asynchronous_D_FF b8(.CLK(clk), .D(data_in[8]), .RST_n(~rst), .Q1(data_tmp[8]));
    Asynchronous_D_FF b9(.CLK(clk), .D(data_in[9]), .RST_n(~rst), .Q1(data_tmp[9]));
    Asynchronous_D_FF b10(.CLK(clk), .D(data_in[10]), .RST_n(~rst), .Q1(data_tmp[10]));
    Asynchronous_D_FF b11(.CLK(clk), .D(data_in[11]), .RST_n(~rst), .Q1(data_tmp[11]));
    Asynchronous_D_FF b12(.CLK(clk), .D(data_in[12]), .RST_n(~rst), .Q1(data_tmp[12]));
    Asynchronous_D_FF b13(.CLK(clk), .D(data_in[13]), .RST_n(~rst), .Q1(data_tmp[13]));
```

```
    Asynchronous_D_FF b14(.CLK(clk), .D(data_in[14]), .RST_n(~rst), .Q1(data_tmp[14]));
    Asynchronous_D_FF b15(.CLK(clk), .D(data_in[15]), .RST_n(~rst), .Q1(data_tmp[15]));
    Asynchronous_D_FF b16(.CLK(clk), .D(data_in[16]), .RST_n(~rst), .Q1(data_tmp[16]));
    Asynchronous_D_FF b17(.CLK(clk), .D(data_in[17]), .RST_n(~rst), .Q1(data_tmp[17]));
    Asynchronous_D_FF b18(.CLK(clk), .D(data_in[18]), .RST_n(~rst), .Q1(data_tmp[18]));
    Asynchronous_D_FF b19(.CLK(clk), .D(data_in[19]), .RST_n(~rst), .Q1(data_tmp[19]));
    Asynchronous_D_FF b20(.CLK(clk), .D(data_in[20]), .RST_n(~rst), .Q1(data_tmp[20]));
    Asynchronous_D_FF b21(.CLK(clk), .D(data_in[21]), .RST_n(~rst), .Q1(data_tmp[21]));
    Asynchronous_D_FF b22(.CLK(clk), .D(data_in[22]), .RST_n(~rst), .Q1(data_tmp[22]));
    Asynchronous_D_FF b23(.CLK(clk), .D(data_in[23]), .RST_n(~rst), .Q1(data_tmp[23]));
    Asynchronous_D_FF b24(.CLK(clk), .D(data_in[24]), .RST_n(~rst), .Q1(data_tmp[24]));
    Asynchronous_D_FF b25(.CLK(clk), .D(data_in[25]), .RST_n(~rst), .Q1(data_tmp[25]));
    Asynchronous_D_FF b26(.CLK(clk), .D(data_in[26]), .RST_n(~rst), .Q1(data_tmp[26]));
    Asynchronous_D_FF b27(.CLK(clk), .D(data_in[27]), .RST_n(~rst), .Q1(data_tmp[27]));
    Asynchronous_D_FF b28(.CLK(clk), .D(data_in[28]), .RST_n(~rst), .Q1(data_tmp[28]));
    Asynchronous_D_FF b29(.CLK(clk), .D(data_in[29]), .RST_n(~rst), .Q1(data_tmp[29]));
    Asynchronous_D_FF b30(.CLK(clk), .D(data_in[30]), .RST_n(~rst), .Q1(data_tmp[30]));
    Asynchronous_D_FF b31(.CLK(clk), .D(data_in[31]), .RST_n(~rst), .Q1(data_tmp[31]));

    always @(*)
        if(rst == 1 && (ena == 1 || ena == 0))
            data_out = 0;
        else if(rst == 0 && ena == 1)
            data_out = data_tmp;
        else if(rst == 0 && ena == 0)
            ;
        else
            data_out = {32{1'bx}};
endmodule
```

四、测试模块建模

1.同步复位 D 触发器

```
`timescale 1ns / 1ps
module Synchronous_D_FF_tb;
    reg CLK, D, RST_n;
    wire Q1, Q2;

    Synchronous_D_FF uut(
    .CLK(CLK), .D(D), .RST_n(RST_n),
    .Q1(Q1), .Q2(Q2));

    initial
    begin
```

```
        CLK = 1;
        #50 CLK = 0;
        #50 CLK = 1;
        #50 CLK = 0;
        #50 CLK = 1;
        #50 CLK = 0;
        #50 CLK = 1;
    end

    initial
    begin
        D <= 0; RST_n <= 1;
        #60 D = 0;
        #100 D = 1;
        #100 RST_n = 0;
    end
endmodule
```

2.异步复位 D 触发器

```
module Asynchronous_D_FF_tb;
    reg CLK, D, RST_n;
    wire Q1, Q2;

    Asynchronous_D_FF uut(
    .CLK(CLK), .D(D), .RST_n(RST_n),
    .Q1(Q1), .Q2(Q2));

    initial
    begin
        CLK = 1;
        #50 CLK = 0;
        #50 CLK = 1;
        #50 CLK = 0;
        #50 CLK = 1;
        #50 CLK = 0;
        #50 CLK = 1;
    end

    initial
    begin
        D <= 0; RST_n <= 1;
        #60 D = 0;
        #100 D = 1;
        #100 RST_n = 0;
```

```
    end
endmodule
```

3.异步复位 JK 触发器

```
`timescale 1ns / 1ps
module JK_FF_tb;
    reg CLK, RST_n, J, K;
    wire Q1, Q2;

    JK_FF uut(
    .CLK(CLK), .J(J), .K(K), .RST_n(RST_n),
    .Q1(Q1), .Q2(Q2));

    initial
    begin
        CLK = 1;
        #25 CLK = 0;
        #25 CLK = 1;
        #25 CLK = 0;
        #25 CLK = 1;
        #25 CLK = 0;
        #25 CLK = 1;
        #25 CLK = 0;
        #25 CLK = 1;
        #25 CLK = 0;
        #25 CLK = 1;
    end

    initial
    begin
        RST_n <= 1; J <= 1; K <= 0;
        #60 J = 0;
        #50 K = 1;
        #50 J = 1;
        #50 RST_n = 0;
    end
endmodule
```

4.PC 寄存器

```
`timescale 1ns / 1ps
module pcreg_tb;
    reg clk, rst, ena;
    reg [31:0] data_in;
    wire [31:0] data_out;
```

```
        pcreg uut(
        .clk(clk), .rst(rst), .ena(ena),
        .data_in(data_in), .data_out(data_out));

        initial
        begin
            clk = 1;
            #25 clk = 0;
            #25 clk = 1;
            #25 clk = 0;
            #25 clk = 1;
            #25 clk = 0;
            #25 clk = 1;
        end

        initial
        begin
            data_in <= 19430407; rst <= 0; ena <= 0;
            #60 rst = 1;
            #50 rst <= 0; ena <= 1;
            #100 rst = 1;
        end
endmodule
```

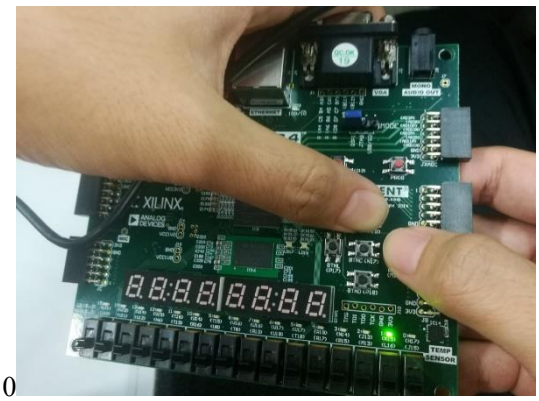
五、实验结果

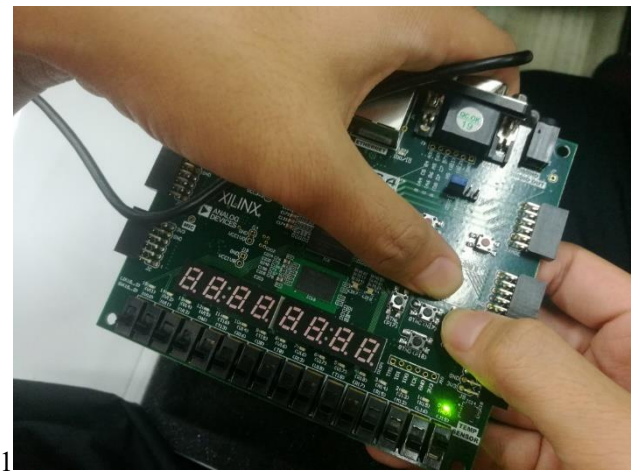
1.同步复位 D 触发器

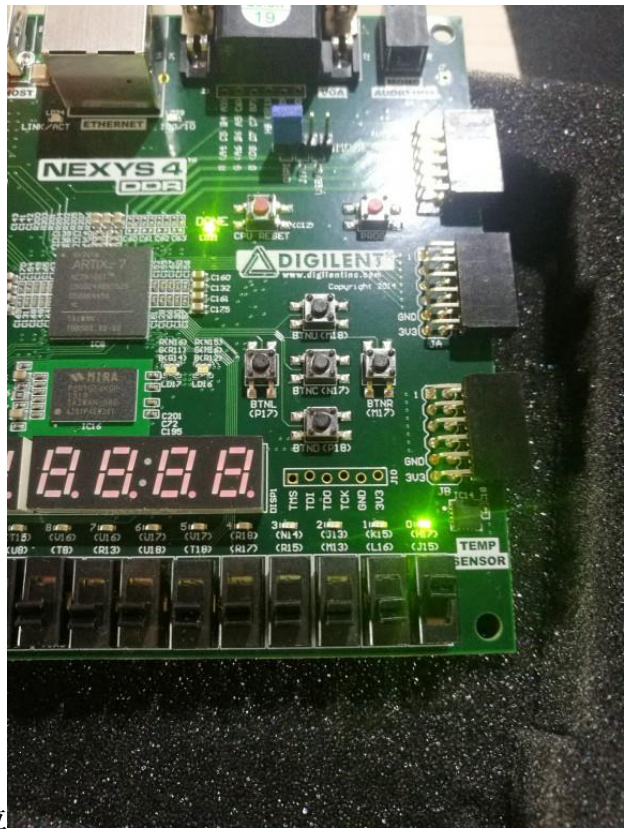
1.1 modelsim 仿真波形图

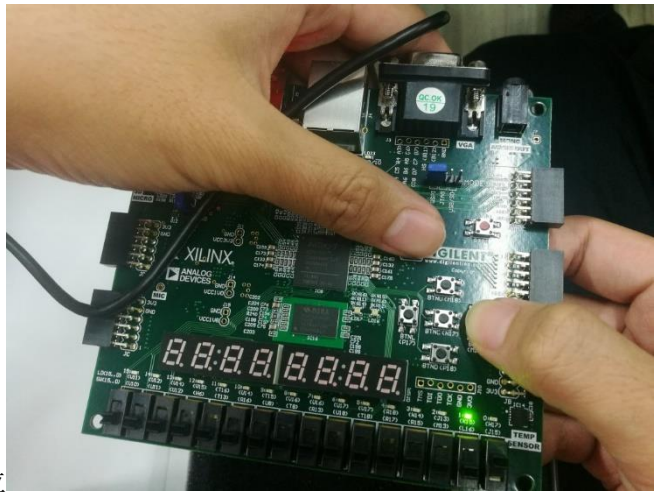
1.2 综合下板

D=0 置 0



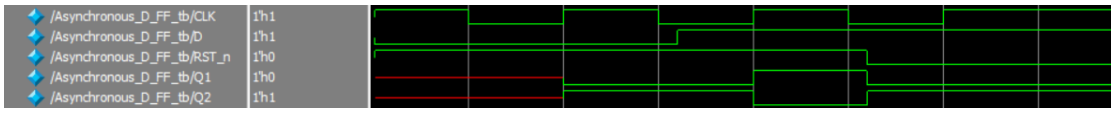
D=1 置 1 

异步不复位 

同步复位 

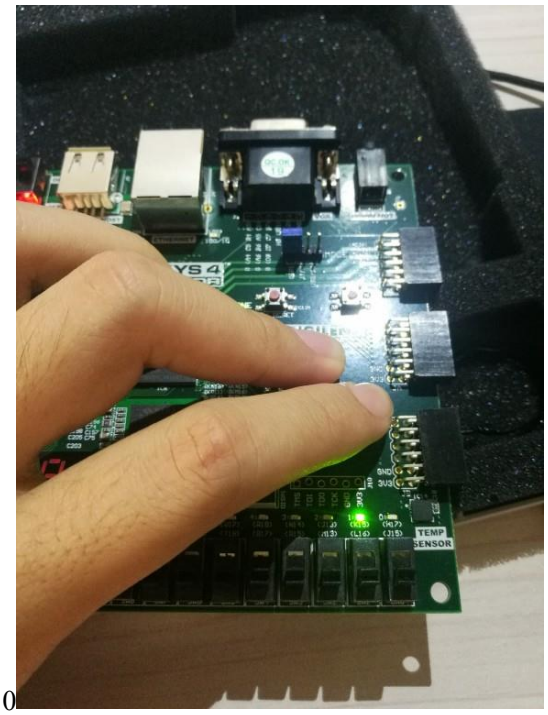
2.异步复位 D 触发器

2.1 modelsim 仿真波形图

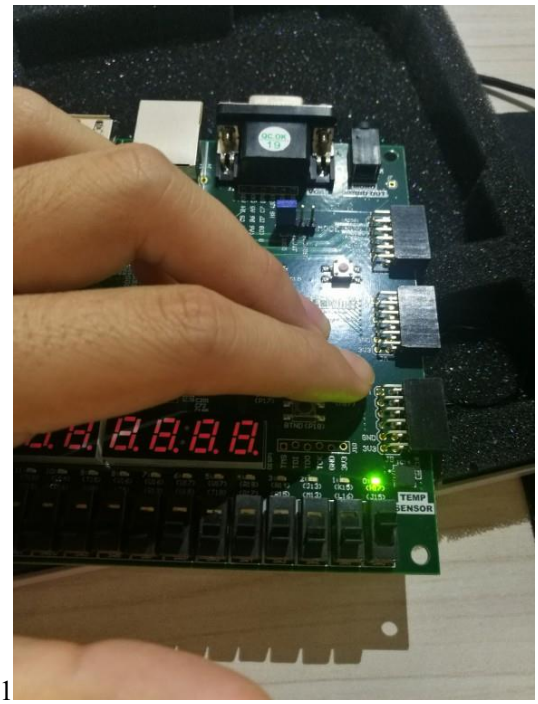


2.2 综合下板

D=0 置 0



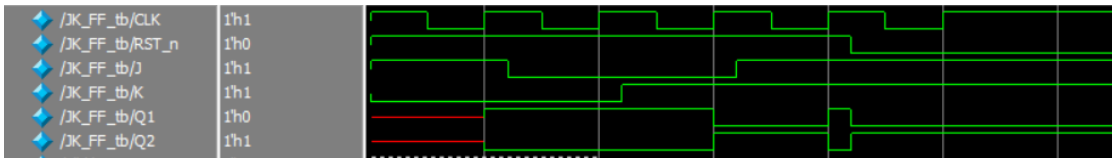
D=1 置 1



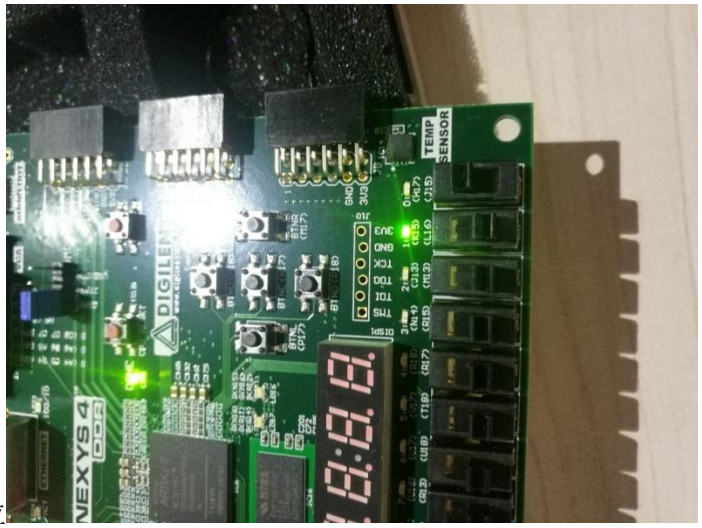
异步复位 

3.异步复位 JK 触发器

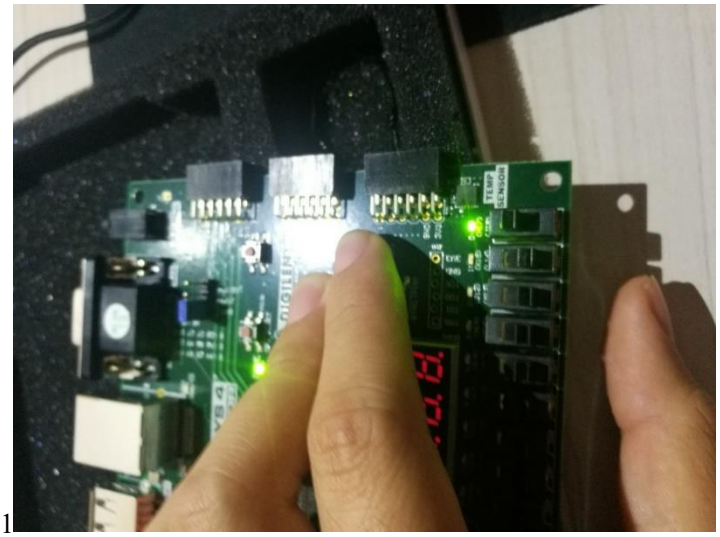
3.1 modelsim 仿真波形图



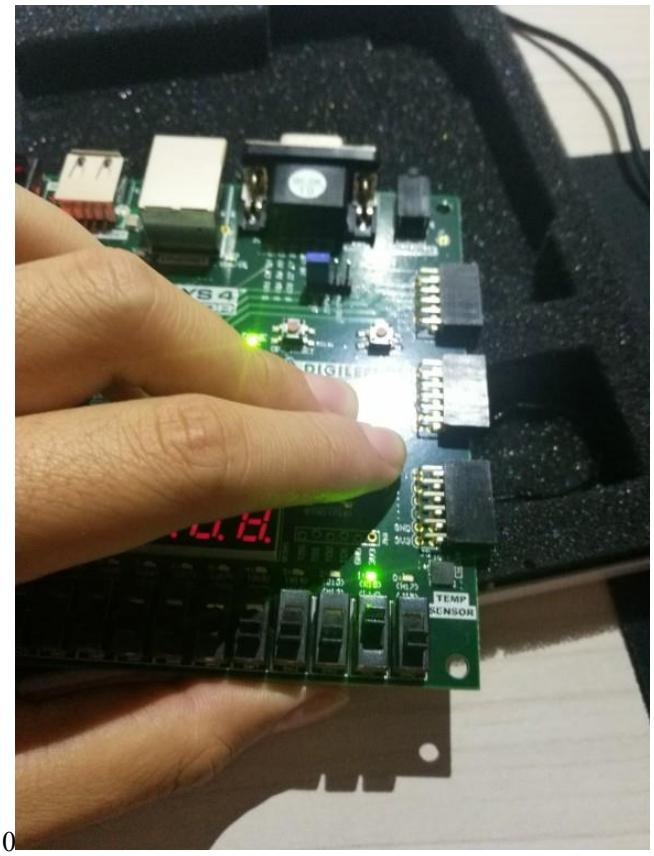
3.2 综合下板

异步复位 

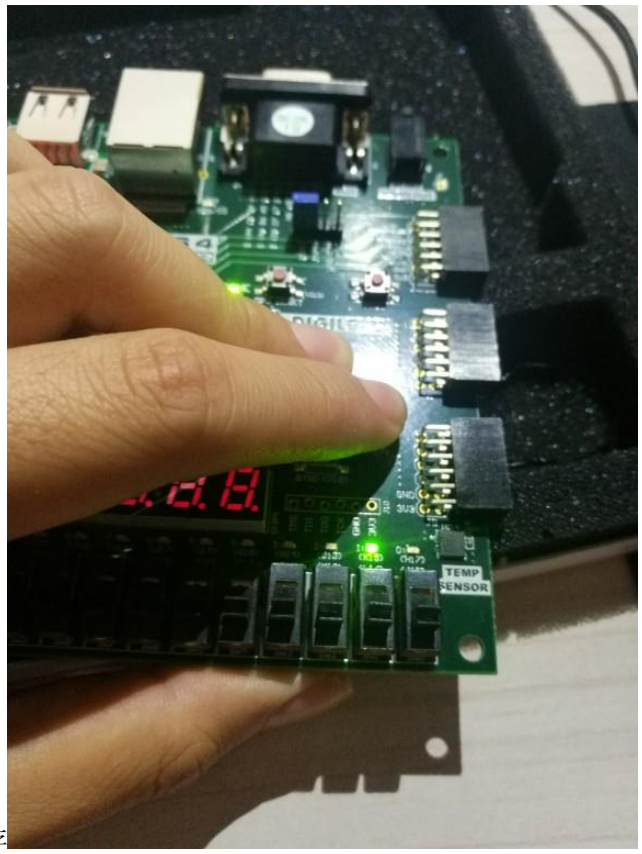
J=1 K=0 置 1



J=0 K=1 置 0



J=0 K=0 锁存



J=1 K=1 交换

4.PC 寄存器

3.1 modelsim 仿真波形图