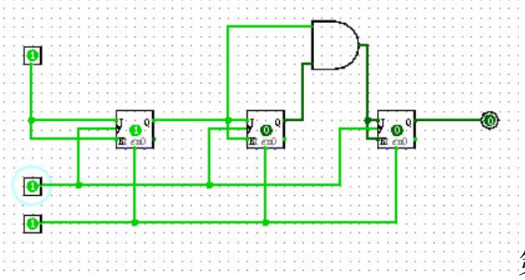
一、实验内容

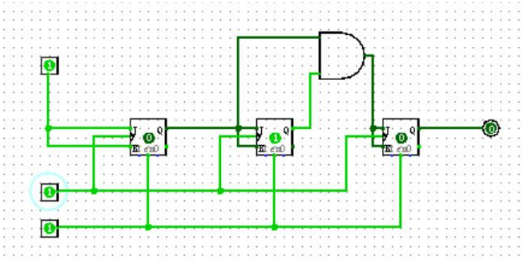
模 8 计数器实验： 使用 logisim 验证模 8 计数器电路逻辑，使用 Verilog 语言实现计数器的建模并设计。进行下板验证时输出到七位数码管中，利用分频器控制时钟频率观察数码管输出

二、硬件逻辑图

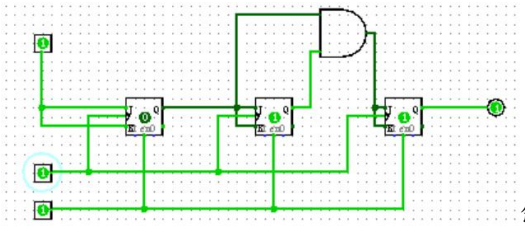
使用 Logicsim 内置的 JK 触发器模块实现对模 8 触发器的电路逻辑验证
在验证前首先利用清零输入进行清零（置初始值）
初始时钟上升沿到来前保持 000 的输出状态



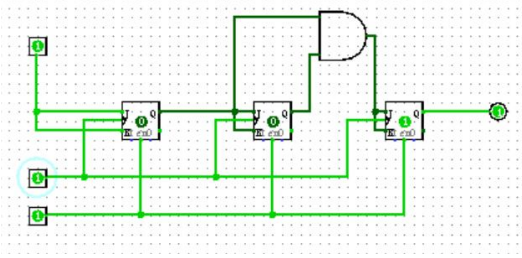
第一个时钟上升沿到来，输出变为 001



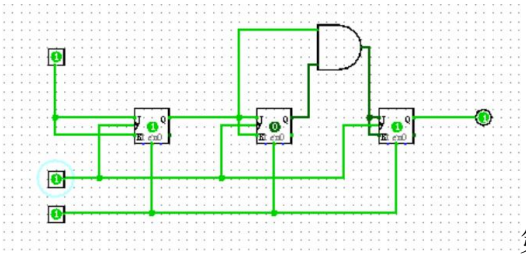
第二个时钟上升沿到来，输出变为 010



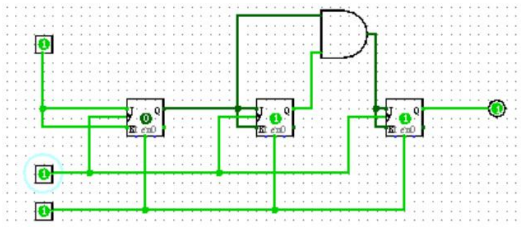
第二个时钟上升沿到来，输出变为 011



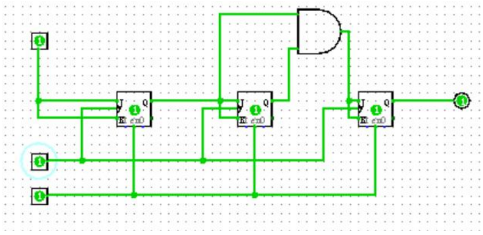
第三个时钟上升沿到来，输出变为 100



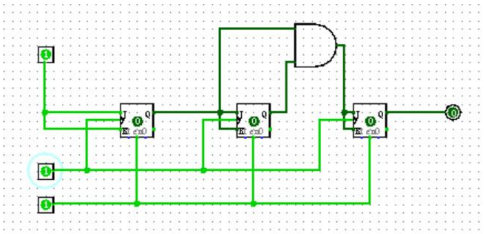
第四个时钟上升沿到来，输出变为 101



第五个时钟上升沿到来，输出变为 110



第六个时钟上升沿到来，输出变为 111



第七个时钟上升沿到来，输出变为 000，回到初始状态

三、模块建模

通过复用前置实验的七段数码管输出、JK 触发器的实现，进行 JK 触发器的实例化，进行 JK 触发器间的相应端口连接，将中间结果输出到相应的 Q 端口。通过 7 位数码管模块将结果输出到数码管显示区

```
module display7(
    input [3:0] iData,
    output [6:0] oData
    );
    reg [6:0] oData_t;
    always @(*)
    //0
    if (!iData[3] && !iData[2] && !iData[1] && !iData[0])
    begin
        oData_t[0] = 0;
```

```
        oData_t[1] = 0;
        oData_t[2] = 0;
        oData_t[3] = 0;
        oData_t[4] = 0;
        oData_t[5] = 0;
        oData_t[6] = 1;
end

//1
else if (!iData[3] && !iData[2] && !iData[1] && iData[0])
begin
        oData_t[0] = 1;
        oData_t[1] = 0;
        oData_t[2] = 0;
        oData_t[3] = 1;
        oData_t[4] = 1;
        oData_t[5] = 1;
        oData_t[6] = 1;
end

//2
else if (!iData[3] && !iData[2] && iData[1] && !iData[0])
begin
        oData_t[0] = 0;
        oData_t[1] = 0;
        oData_t[2] = 1;
        oData_t[3] = 0;
        oData_t[4] = 0;
        oData_t[5] = 1;
        oData_t[6] = 0;
end

//3
else if (!iData[3] && !iData[2] && iData[1] && iData[0])
begin
        oData_t[0] = 0;
        oData_t[1] = 0;
        oData_t[2] = 0;
        oData_t[3] = 0;
        oData_t[4] = 1;
        oData_t[5] = 1;
        oData_t[6] = 0;
end
```

```
//4
else if (!iData[3] && iData[2] && !iData[1] && !iData[0])
begin
	oData_t[0] = 1;
	oData_t[1] = 0;
	oData_t[2] = 0;
	oData_t[3] = 1;
	oData_t[4] = 1;
	oData_t[5] = 0;
	oData_t[6] = 0;
end

//5
else if (!iData[3] && iData[2] && !iData[1] && iData[0])
begin
	oData_t[0] = 0;
	oData_t[1] = 1;
	oData_t[2] = 0;
	oData_t[3] = 0;
	oData_t[4] = 1;
	oData_t[5] = 0;
	oData_t[6] = 0;
end

//6
else if (!iData[3] && iData[2] && iData[1] && !iData[0])
begin
	oData_t[0] = 0;
	oData_t[1] = 1;
	oData_t[2] = 0;
	oData_t[3] = 0;
	oData_t[4] = 0;
	oData_t[5] = 0;
	oData_t[6] = 0;
end

//7
else if (!iData[3] && iData[2] && iData[1] && iData[0])
begin
	oData_t[0] = 0;
	oData_t[1] = 0;
	oData_t[2] = 0;
	oData_t[3] = 1;
	oData_t[4] = 1;
```

```
            oData_t[5] = 1;
            oData_t[6] = 1;
        end

        //8
        else if (iData[3] && !iData[2] && !iData[1] && !iData[0])
        begin
            oData_t[0] = 0;
            oData_t[1] = 0;
            oData_t[2] = 0;
            oData_t[3] = 0;
            oData_t[4] = 0;
            oData_t[5] = 0;
            oData_t[6] = 0;
        end

        //9
        else if (iData[3] && !iData[2] && !iData[1] && iData[0])
        begin
            oData_t[0] = 0;
            oData_t[1] = 0;
            oData_t[2] = 0;
            oData_t[3] = 0;
            oData_t[4] = 1;
            oData_t[5] = 0;
            oData_t[6] = 0;
        end

        assign oData = oData_t;
endmodule


module JK_FF(
        input CLK,
        input J,
        input K,
        input RST_n,
        output reg Q1,
        output reg Q2
        );
        reg temp;

        always @(posedge CLK or negedge RST_n)
        begin
```

```
            //reset
            if (!RST_n)
            begin
                Q1 = 0;
                Q2 = 1;
            end

            //flip
            else if (J == 1 && K == 1)
            begin
                temp = Q1;
                Q1 = Q2;
                Q2 = temp;
            end

            else if (J == 0 && K == 1)
            begin
                Q1 = 0;
                Q2 = 1;
            end

            else if (J == 1 && K == 0)
            begin
                Q1 = 1;
                Q2 = 0;
            end
        end
endmodule

module Counter8(
        input CLK,
        input rst_n,
        output [2:0] oQ,
        output [6:0] oDisplay
        );

        //first jk
        reg inputJ0 = 1;
        reg inputK0 = 1;
        wire Q0;
        wire Q0_n;


        //second jk
```

```
    wire Q1;
    wire Q1_n;


    //third jk
    wire iJ2;
    wire Q2;
    wire Q2_n;

    reg [3:0] intv;

    JK_FF jk0(.CLK(CLK), .J(inputJ0), .K(inputK0), .RST_n(rst_n), .Q1(Q0), .Q2(Q0_n));

    JK_FF jk1(.CLK(CLK), .J(Q0), .K(Q0), .RST_n(rst_n), .Q1(Q1), .Q2(Q1_n));

        //and gate
    and andGate(iJ2, Q1, Q0);

    JK_FF jk2(.CLK(CLK), .J(iJ2), .K(iJ2), .RST_n(rst_n), .Q1(Q2), .Q2(Q2_n));

    display7 d7(.iData(intv), .oData(oDisplay));


    reg [2:0] temp_oQ;

    always@(posedge CLK)
    begin


        temp_oQ[0] = Q0;
        temp_oQ[1] = Q1;
        temp_oQ[2] = Q2;

        intv[3] = 0;
        intv[2] = Q2;
        intv[1] = Q1;
        intv[0] = Q0;


    end
    assign oQ = temp_oQ;


endmodule
```

四、测试模块建模

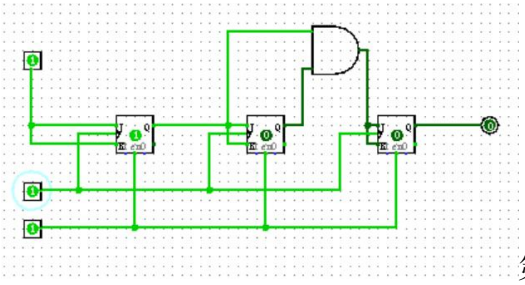
初始化时钟、初始化清零信号，进行时钟的周期性设置，在进行波形仿真时观察波形输出即可

```
module Counter8_tb();
    reg CLK;
    reg rst_n;
    wire [2:0] oQ;
    wire [6:0] oDisplay;


    //CLK
    initial
    begin
    CLK = 0;
    #1 CLK = 1;
    end
    always #2 CLK = ~CLK;


    Counter8 ct(.CLK(CLK), .rst_n(rst_n), .oQ(oQ), .oDisplay(oDisplay));


    //rst_n
    initial
    begin
        rst_n = 0;
        #1 rst_n = 1;
    end

endmodule
```

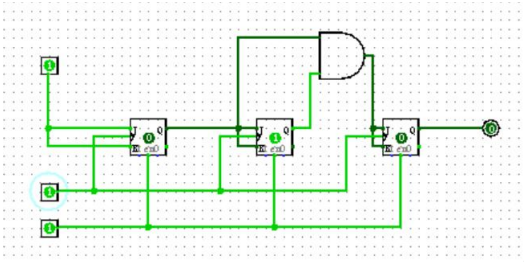
五、实验结果

1. Logicsim 逻辑验证

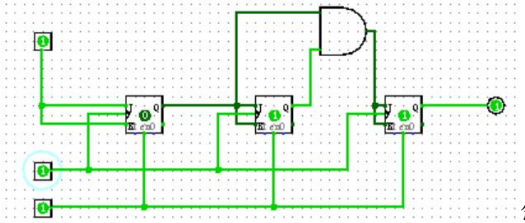
使用 Logicsim 内置的 JK 触发器模块实现对模 8 触发器的电路逻辑验证
在验证前首先利用清零输入进行清零（置初始值）
初始时钟上升沿到来前保持 000 的输出状态



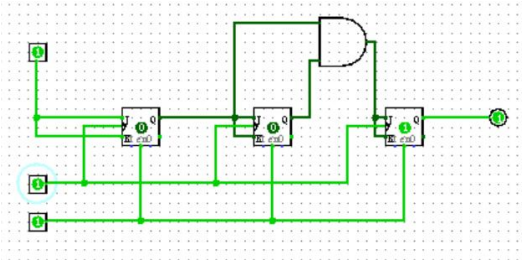
第一个时钟上升沿到来，输出变为 001



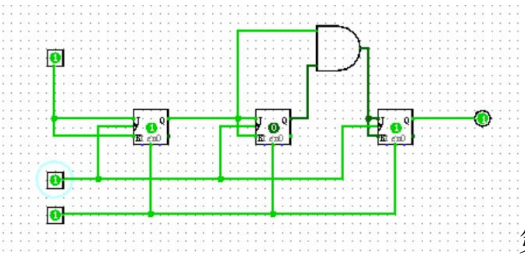
第二个时钟上升沿到来，输出变为 010



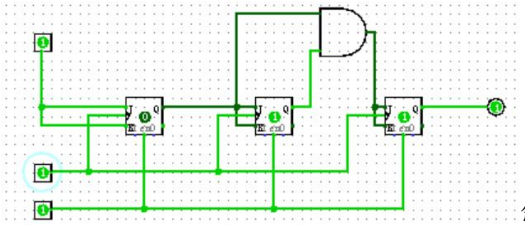
第二个时钟上升沿到来，输出变为 011



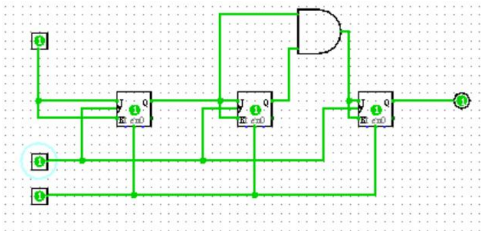
第三个时钟上升沿到来，输出变为 100



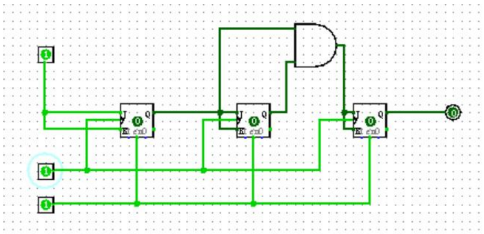
第四个时钟上升沿到来，输出变为 101



第五个时钟上升沿到来，输出变为 110

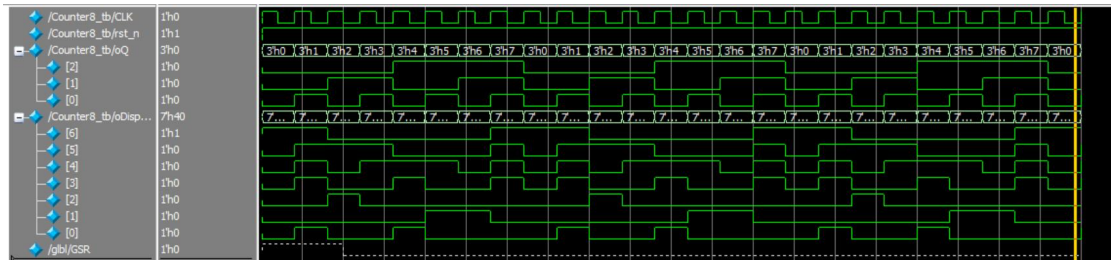


第六个时钟上升沿到来，输出变为 111



第七个时钟上升沿到来，输出变为 000，回到初始状态

2. ModelSim 波形仿真



根据相应的 8 位循环计数器输出，将结果显示到 7 位数码管，其与七位数码管的各管标号为

表 6.7　七段数码管译码驱动器逻辑功能表

输入				输出							显示字符
D_3	D_2	D_1	D_0	g	f	e	d	c	b	a	
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	1	1
0	0	1	0	0	1	0	0	1	0	0	2
0	0	1	1	0	1	1	0	0	0	0	3
0	1	0	0	0	0	1	1	0	0	1	4
0	1	0	1	0	0	1	0	0	1	0	5
0	1	1	0	0	0	0	0	0	1	0	6
0	1	1	1	1	1	1	1	0	0	0	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	1	0	0	0	0	9

每 20 个时间单位一个时钟上升沿到来，进行一次状态转换并将其输出
可以观察到 Q 的输出从二进制 0 依次变化到二进制 7，之后转换为初始状态 0
相应七位数码管的输出与二进制 0 到 7 相对应，波形验证逻辑正确

3. 下板验证

实际情况由于开发板上的时钟上升沿到来周期过小，所以先将开发板上的时钟信号进行分频器进行处理后再输入模块，分频器的分频倍数设置到 1e8 可以较为明显的观察到数码显示管的变化（间隔大约 1s）

其中需要在 Counter8.v 中添加模块 Divider
module Divider(

```
    input I_CLK,
    input rst,
    output O_CLK
    );

    parameter times = 100000000;   //分频倍数为 1e8

    integer cnt = 0;
    reg tO_CLK = 0;
    always @(posedge I_CLK or negedge rst)
    begin
        if (!rst) tO_CLK = 0;
        else
        begin

            if (cnt == times)
            begin
                tO_CLK = ~tO_CLK;
                cnt = 0;
            end
            cnt = cnt + 1;
        end

    end
    assign O_CLK = tO_CLK;
endmodule
```

其中顶层模块改为

```
module Counter8(
    input CLK,
    input rst_n,
    output [2:0] oQ,
    output [6:0] oDisplay
    );

    //first jk
    reg inputJ0 = 1;
    reg inputK0 = 1;
    wire Q0;
    wire Q0_n;


    //second jk
    wire Q1;
```

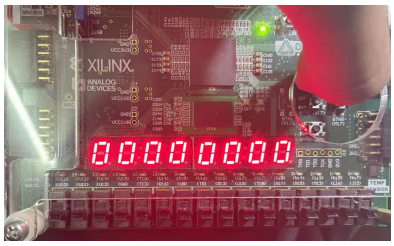
```
wire Q1_n;


//third jk
wire iJ2;
wire Q2;
wire Q2_n;

reg [3:0] intv;

wire gCLK;

Divider dv(.I_CLK(CLK), .rst(rst_n), .O_CLK(gCLK));

JK_FF jk0(.CLK(gCLK), .J(inputJ0), .K(inputK0), .RST_n(rst_n), .Q1(Q0), .Q2(Q0_n));

JK_FF jk1(.CLK(gCLK), .J(Q0), .K(Q0), .RST_n(rst_n), .Q1(Q1), .Q2(Q1_n));

	//and gate
and andGate(iJ2, Q1, Q0);

JK_FF jk2(.CLK(gCLK), .J(iJ2), .K(iJ2), .RST_n(rst_n), .Q1(Q2), .Q2(Q2_n));

display7 d7(.iData(intv), .oData(oDisplay));


reg [2:0] temp_oQ;

always@(posedge gCLK)
begin


	temp_oQ[0] = Q0;
	temp_oQ[1] = Q1;
	temp_oQ[2] = Q2;

	intv[3] = 0;
	intv[2] = Q2;
	intv[1] = Q1;
	intv[0] = Q0;


end
assign oQ = temp_oQ;
```

endmodule

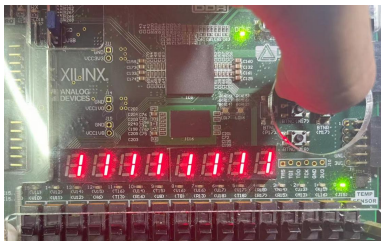
由此在下板时，首先进行清零，（清零高电平有效），在清零信号无效时（按住清零信号键）

在 J15-H13 输出为 Q，其值从 1 变化到 7 周期性变化，在数码管部分输出同样 0-7 周期性变化，其间隔大约为 1s

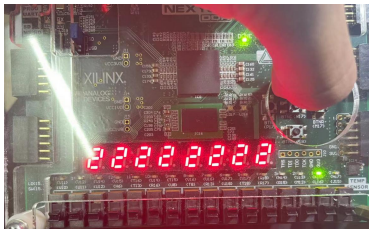
初始状态



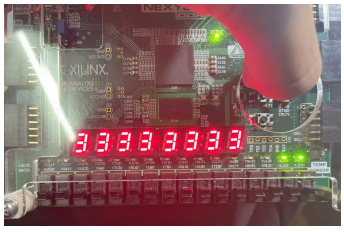
输出为 1



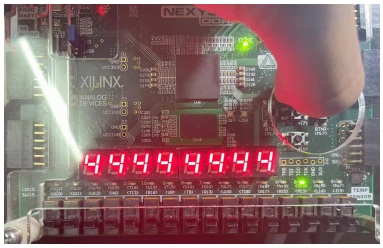
输出为 2



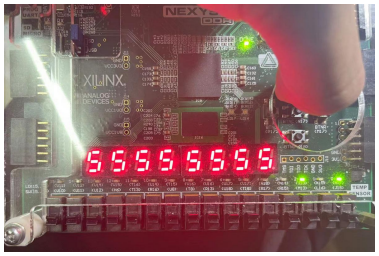
输出为 3



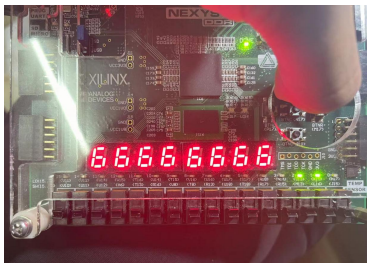
输出为 4



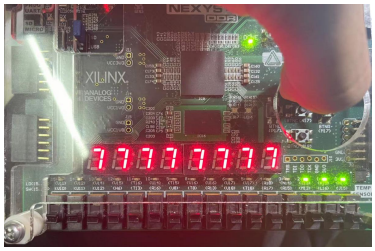
输出为 5



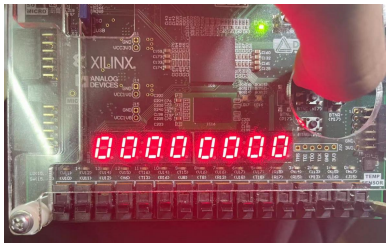
输出为 6



输出为 7



循环到状态 0



可以观察到模 8 计算器逻辑符合预期

一、 实验内容

利用计数器的脉冲输出频率可以获得一个输入时钟信号分频后的时钟信号其频率相对于原时钟频率可以成整数倍变化。利用 Verilog 进行行为描述建模，在 Modelsim 进行仿真，下板进行功能验证。

二、 硬件逻辑图

该实验未要求 Logicsim 原理图

三、 模块建模

通过手动设置计数器进行计数，当计数达到相应的分频倍数时，进行计数归零并进行过输出时钟状态转换，达到时钟分频的效果。

```
module Divider(
    input I_CLK,
    input rst,
    output O_CLK
    );

    parameter times = 100000000;

    integer cnt = 0;
    reg tO_CLK = 0;
    always @(posedge I_CLK or negedge rst)
    begin
        if (rst) tO_CLK = 0;
        else
        begin

            if (cnt == times)
            begin
                tO_CLK = ~tO_CLK;
                cnt = 0;
            end
            cnt = cnt + 1;
        end

    end
    assign O_CLK = tO_CLK;
```

```
endmodule
```

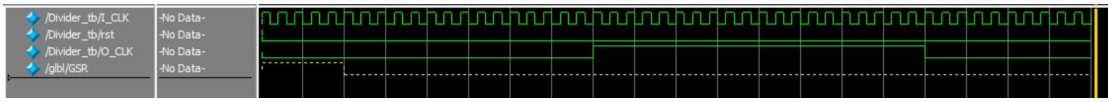
四、 测试模块建模

初始设置时钟并进行输出初始化（置零操作），合理缩短原时钟信号的到来时间，观察输出时钟信号的频率

```
module Divider_tb();
    reg I_CLK;
    reg rst;
    wire O_CLK;

    Divider dv(.I_CLK(I_CLK), .rst(rst), .O_CLK(O_CLK));

    initial
        begin
            I_CLK = 0;
            #1 I_CLK = 1;
        end
    always #10 I_CLK = ~I_CLK;

    initial
    begin
        rst = 1;
        #1 rst = 0;
    end

Endmodule
```

五、 实验结果

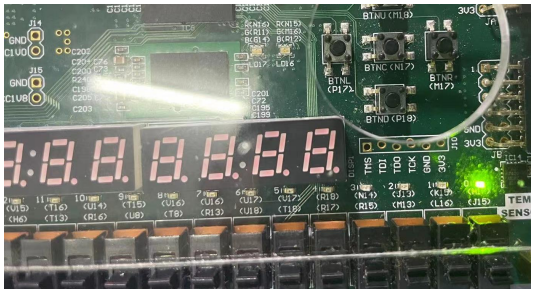
1. Modelsim 波形仿真



将原时钟周期设置为 5 个时间单位，可以在 Modelsim 波形显示区观察到大约 1.5 个时钟周期，可以看到输出时钟周期进行了有效放大，通过观察原时钟到来次数可以观察到每 20 个时间单位进行一次输出时钟的状态转换，分频器逻辑验证正确。

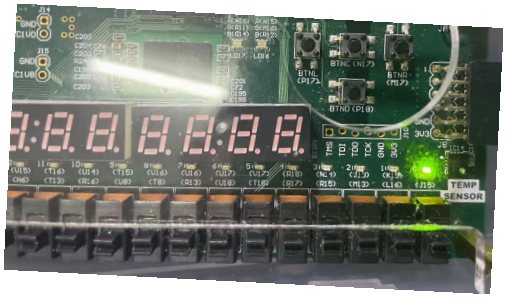
2. 下板验证

在分频倍数为 20 的情况下，下板后的频率人眼是无法观察到输出端指示灯熄灭的，因为时钟到来频率过快，表现为指示灯“一直”保持发光

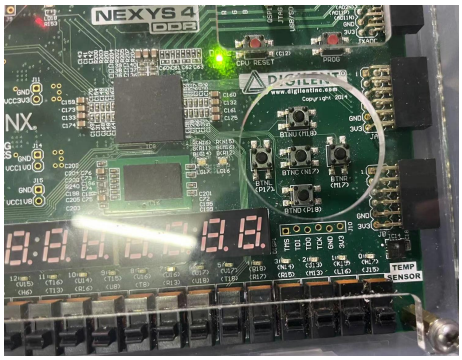


为观察实验现象，手动将放大倍数调整到 1e8，使得大概两次输出时钟信号间隔为 1s，
输出现象：每隔 2s 左右完成一次输出时钟周期，指示灯周期性发光与熄灭（注意此处清零信号是高电平有效，所以不需要按住时钟信号）

输出时钟高电平



输出时钟低电平



调整放大倍数足够大之后，输出时钟信号在人眼可辨范围内进行周期性发光、熄灭，逻辑正确。