作业要求：（1）**必须提供可综合的电路行为描述的HDL代码，测试代码和仿真波形。**

（2）作业必须打印上交。

1. **（40分）用Verilog设计一个5/9分频电路，即每9个时钟周期内，输出5个周期脉冲信号。**

解：首先分析题目，题目要求实现在9个时钟周期内输出5个周期的脉冲信号，也就是分频，属于小数分频。如果只用HDL代码不借助外部功能模块的情况下， 采用拼接法所以计算解方程得,也就是说在九个时钟周期内生成一个1分频和4个2分频即可实现在9个时钟周期内输出5个周期脉冲信号，或者可以选择“敲除”9个时钟周期中的4个。这里采用拼接法：

HDL代码如下

module fdiv\_9\_5 (

input clk,

input clr,

output reg clk\_div

);

reg [4:0] cnt0;

always@(posedge clk or posedge clr)begin//计数器

if(clr)

cnt0 <= 0;

else

begin

if(cnt0 < 8)

cnt0 <= cnt0 + 1'b1;

else

cnt0 <= 0;

end

end

always@(\*)//分频

begin

if(clr)

clk\_div = 0;

else

case (cnt0)

5'b00000:clk\_div = clk;

5'b00010,5'b00100,5'b00110,5'b01000 : clk\_div = 1'b0;

5'b00001,5'b00011,5'b00101,5'b00111 : clk\_div = 1'b1;

default: clk\_div = 0;

endcase

end

endmodule

测试程序：

`timescale 1ns/1ps

module fdiv\_9\_5\_tb();

reg clk;

reg clr;

wire clk\_div;

fdiv\_9\_5 F1(

    .clk(clk),

    .clr(clr),

    .clk\_div(clk\_div)

);

parameter ClockPeriod=10;

initial

    begin

        clk=0;

        forever

        #(ClockPeriod/2) clk=~clk;

end

initial begin

    clr = 0;

    #15;

    clr = 1;

    #10;

    clr = 0;

    #90;

    clr = 1;

    #10;

    clr = 0;

    #1000;

    $stop;

end

endmodule

仿真结果如下：

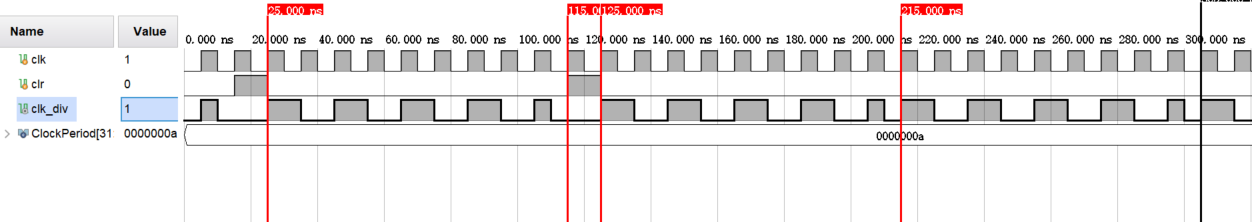


图1分频器仿真结果

如图1所示前两条线之间clk为9个时钟周期，clk\_div为5个周期脉冲信号。当clr信号到来时分频器归零从新开始分频，首先产生4个二分频，再产生一个1分频。

2、（60分）**设计一个数字记录与回放电路，电路的信号与电路的功能描述如下：**

**功能描述：**

**待设计的电路内部有一个二位十六进制计数器，计数频率为1KHZ，通过输入启停信号S\_P可以实现计数器的启动和停止功能，当计数器停止时，当前的计数结果将会被寄存起来。不断的输入S\_P，就有不同的计数结果被存储起来。脉冲输入信号Rev用来显示储存的数据，每输入一个Rev脉冲，按照存储时间的先后，将S\_P信号先存储的计数结果先显示出来。S\_P和Rev脉冲高、低电平时间远大于时钟信号周期。**

**回放存储电路最多只能存储8个数据，如果有新的数据存储，且存储电路已经存储满了，则将最先存储的数据替换成新输入的数据。**

**回放时，如果已经回放了所有存储的电路，含空的存储单元，则重新回放。**

**除清零允许异步外，Rev和S\_P的触发必须为同步触发。**

|  |  |  |
| --- | --- | --- |
| **信号名** | **位宽** | **功能描述** |
| **Clk** | **1** | **脉冲输入。1Khz时钟信号，上升沿有效。** |
| **Rst** | **1** | **电平输入。异步清零信号，高电平清零。** |
| **S\_P** | **1** | **脉冲输入。计数启停，上升沿有效，每输入一个S\_P，计数或停止状态就发生改变。** |
| **Rev** | **1** | **脉冲输入。存储的计数结果回放信号，上升沿有效，按照先存储先播放原则回放存储的数据。输入一个Rev脉冲，回放一个数据。** |
| **Dis\_LED** | **8** | **电平输出。2位十六进制LED输出，在S\_P有效时播放计数结果，在Rev有效时显示存储的数据。** |
| **R\_S** | **1** | **电平输出。高电平表示回放，低电平表示存储。** |

解：首先分析题目可知：

S\_P信号为脉冲输入，同步时序逻辑，并且每次点击起停状态发生改变。对S\_P信号进行二分频得到信号S\_P\_in，这样S\_P\_in的高电平都是开始计数，S\_P\_in下降沿都是停止计数，只需要对S\_P\_in进行下降沿检测生成写使能信号adr\_en；

Rev信号也做这样的处理，每次检测到其上升沿则回放一个数据。

由于当8位存满时应该让新输入的值代替一开始输入的值，且优先输出先输入的值。所以检测寄存器是否存满，如果存满，则继续按照写指针进行读取。如果没有存满则从零开始回放。如果回放完了，则循环回放。

写指针为4位信号adr[3:0]，从零开始每次写入数据指针加一，当指针等于8时输入存储到第0位，同时指针赋值为1，且寄存器状态信号full为高，否则为低。

回放时使用两个指针readr2[3:0]和readr[2:0]，如果寄存器没有满readr则从零开始读，如果满了readr2则从adr（adr≠7）当前值开始读，这样可以保正读的一定是先输入的数据。

HDL代码如下：

module S\_R (

input clk,

input rst,

input S\_P,

input Rev,

output reg [7:0] Dis\_LED,

output reg R\_S

);

reg [7:0] cnt\_16;

reg [1:0] test\_Rev;

reg [1:0] adr\_add;

reg [7:0] sram\_16[7:0];

reg [3:0] adr;

reg [2:0] readr;

wire full;

integer i;

reg [3:0]readr2;

wire adr\_en;

reg S\_P\_in;

wire Rev\_in;

//---------------------------------------------分频器--------------------------------------------------------//

always@( posedge S\_P)begin

S\_P\_in = ~S\_P\_in;

end

//----------------------------------------------边沿检测------------------------------------------------------//

always @(posedge clk,posedge rst) begin

if(rst) begin

test\_Rev <= 0;

S\_P\_in <= 0;

adr\_add <= 0;

end

else begin

test\_Rev[0] <= Rev;

test\_Rev[1] <= test\_Rev[0];//回放上升沿

adr\_add[0] <= S\_P\_in;

adr\_add[1] <= adr\_add[0];//存储计数上升沿

end

end

assign Rev\_in = rst ? 0:((test\_Rev==2'b01)? 1:0);//回放输出

assign adr\_en = rst ? 0:((adr\_add==2'b10)? 1:0);//存储地址加一使能

//-------------------------------------------------计数器------------------------------------------------------//

always@(posedge clk ,posedge rst)begin

if(rst)begin

cnt\_16 <= 0;

Dis\_LED <= 0;

end

else begin

if(S\_P\_in) begin

cnt\_16 <= cnt\_16 + 1'b1;

Dis\_LED <= cnt\_16;//显示计数器的数

end

else begin

cnt\_16 <= cnt\_16;

end

end

end

//----------------------------------------------存入寄存器-------------------------------------------------//

always@(posedge clk ,posedge rst)begin

if(rst)begin

adr <= 0;

R\_S <= 0;//存储状态

for(i=0;i<=7;i=i+1)sram\_16[i] <= 0;

end

else begin

if( adr\_en ) begin

R\_S <= 0;

if( adr <= 7)begin

sram\_16[adr] <= cnt\_16;

adr <= adr+1'b1;

Dis\_LED <= cnt\_16;

readr2 <= adr+1'b1;

end

else begin

sram\_16[0] <= cnt\_16;

adr <= 1;

Dis\_LED <= cnt\_16;

readr2 <= 1;

end

end

end

end

//-------------------------------------------------是否存满-------------------------------------------------//

assign full = rst ? 0:((adr==8)? 1:full);//写地址只要有一次等于8则存满，且锁存状态。

//----------------------------------------------------回放-------------------------------------------------------//

always@(posedge clk ,posedge rst) begin

if(rst) begin

readr <= 0;

readr2 <= 0;

end

else begin

if(Rev\_in) begin

R\_S <= 1;

if(full)begin//存满了

if( readr2 <= 7)begin//写地址小于等于7

Dis\_LED <= sram\_16[readr2];

readr2 <= readr2 + 1'b1;

end

else begin//写地址大于7

Dis\_LED <= sram\_16[0];

readr2 <=1'b1;

end

end

else begin//没存满从头开始读

Dis\_LED <= sram\_16[readr];//显示存储的数

readr <= readr + 1'b1;

end

end

end

end

endmodule

测试代码如下：

`timescale 100ns/1ns

module S\_R\_tb ();

reg clk;

reg rst;

reg S\_P;

reg Rev;

wire [7:0]Dis\_LED;

wire R\_S;

integer i;

integer k;

S\_R S1(

.clk(clk),

.rst(rst),

.S\_P(S\_P),

.Rev(Rev),

.Dis\_LED(Dis\_LED),

.R\_S(R\_S)

);

parameter ClockPeriod=10;

initial begin

clk=0;

forever

#(ClockPeriod/2) clk=~clk;

end

initial begin

Rev = 0;

S\_P = 0;

#5;

rst = 0;

#10;

rst = 1;

#10;

rst = 0;

for(i = 0;i <= 9;i=i+1)begin

S\_P = 1;

#50;

S\_P = 0;

#50;

end

#900;

for(k = 0;k <= 15;k=k+1)begin

Rev = 1;

#50;

Rev = 0;

#50;

end

rst = 0;

#10;

rst = 1;

#10;

rst = 0;

for(i = 0;i <= 17;i=i+1)begin

S\_P = 1;

#50;

S\_P = 0;

#50;

end

#900;

for(k = 0;k <= 15;k=k+1)begin

Rev = 1;

#50;

Rev = 0;

#50;

end

$stop;

end

endmodule

仿真图像如下：

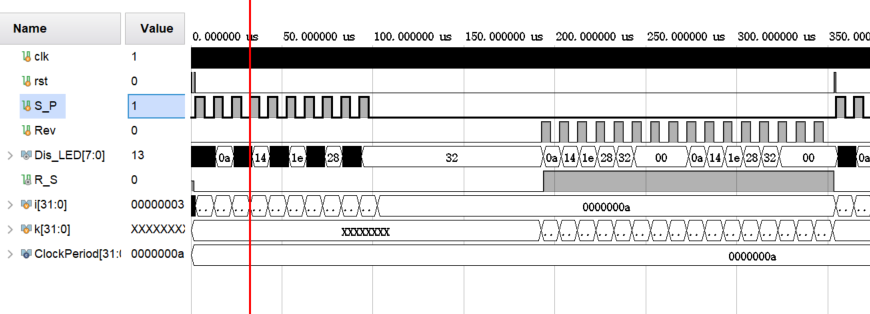


图2 回放电路，未存满

如图2所示，在进行计数开始和停止以及存储与实际输入的S\_P信号存在延迟，延迟时间为两个周期周期，消耗在了边沿检测上。S\_P信号为高后开始计数，第二次S\_P为高后计数暂停，并将计数器结果存储到寄存器中，回放时同样存在两个时钟周期延迟，当回放完包括为空的单元，则从头开始回放。

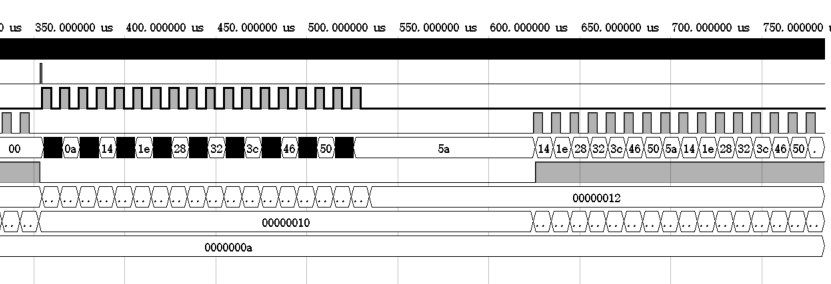


图3 回放仿真，已经存满并持续输入

从图三中可以发现，当存满时继续存入的数会替换掉一开始存入的数，在回放时优先输出先存的数。