# 实验四 模型机时序部件的实现

# 班级 计科1903 姓名 陈旭 学号 201914020128

一、实验目的

1. 熟悉计数器、寄存器和 RAM 的工作原理。
2. 了解模型机中 SM 的作用。
3. 学会使用 VHDL 语言设计时序电路。

二、实验内容

1. 用 VHDL 语言设计SM。
2. 用 VHDL 语言设计PC(指令计数器)。
3. 用 VHDL 语言设计通用寄存器组。
4. 采用Quartus中已有的参数化模块来定制RAM功能。

三、实验方法

1. 实验方法
   * 采用基于FPGA进行数字逻辑电路设计的方法。
   * 采用基本逻辑门电路和组合逻辑电路实现SM、PC(指令计数器)、通用寄存器组。
   * 采用Quartus中已有的参数化模块来定制RAM
   * 采用的软件工具是Quartus II。
2. 实验软件操作步骤
3. SM
4. 新建，编写源代码。

(1)选择保存项和芯片类型：【File】-【new project wizard】-【next】（设置文件路径为C:\Users\86150\Desktop\mylearn\vscodec++\logicandcomputerdesignfundamentals\experiment4\SM设置project name为SM）-【next】（设置文件名SM）-【next】（设置芯片类型为【cyclone-EP1CT144C8】）-【finish】

(2).新建：【file】-【new】（第二个VHDL File）-【OK】

1. 写好源代码，保存文件（SM.vhd）。
2. 编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译结果有警告，编译成功。
3. 波形仿真及验证。新建一个vector waveform file。按照程序所述插入clk, Sm\_en,z

）。(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all；【list】）-【>>】-【ok】-【ok】)。任意设置clk, Sm\_en的输入波形…点击保存按钮保存。（操作为：点击name（如：enable））-右击-【value】-【count】（如设置binary；start value=0; count value=5.0ns），同理设置name b（如0,1,5），保存）。然后【start simulation】，出name C的输出图。

1. 时序仿真和功能仿真。
2. 查看RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。
3. **PC(指令计数器)**
4. 新建，编写源代码。

(1)选择保存项和芯片类型：【File】-【new project wizard】-【next】（设置文件路径为C:\Users\86150\Desktop\mylearn\vscodec++\logicandcomputerdesignfundamentals\experiment4\program\_counter设置project name为program\_counter）-【next】（设置文件名program\_counter.vhd）-【next】（设置芯片类型为【cyclone-EP1CT144C8】）-【finish】

(2).新建：【file】-【new】（VHDL File）-【OK】

1. 写好源代码，保存文件（program\_counter.vhd）。
2. 编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译结果有警告，编译成功。
3. 波形仿真及验证。新建一个vector waveform file。按照程序所述插入ldpc, inpc, clk）。(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all；【list】）-【>>】-【ok】-【ok】)。任意设置ldpc, inpc, clk

的输入波形…点击保存按钮保存。（操作为：点击name（如：enable））-右击-【value】-【count】（如设置binary；start value=0; count value=5.0ns），同理设置name b（如0,1,5），保存）。然后【start simulation】，出name C的输出图。

1. 时序仿真和功能仿真。
2. 查看RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。
3. **通用寄存器组**
4. 新建，编写源代码。

(1)选择保存项和芯片类型：【File】-【new project wizard】-【next】（设置文件路径为C:\Users\86150\Desktop\mylearn\vscodec++\logicandcomputerdesignfundamentals\experiment4\general\_registers设置project name为general\_registers）-【next】（设置文件名general\_registers）-【next】（设置芯片类型为【cyclone-EP1CT144C8】）-【finish】

(2).新建：【file】-【new】（VHDL File）-【OK】

1. 写好源代码，保存文件（general\_registers.vhd）。
2. 编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译结果有警告，编译成功。
3. 波形仿真及验证。新建一个vector waveform file。按照程序所述插入 we, clk, raa, rwba, i, s。(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all；【list】）-【>>】-【ok】-【ok】)。任意设置结点的输入波形…点击保存按钮保存。（操作为：点击name（如：enable））-右击-【value】-【count】（如设置binary；start value=0; count value=5.0ns），同理设置name b（如0,1,5），保存）。然后【start simulation】，出name C的输出图。
4. 时序仿真和功能仿真。
5. 查看RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。
6. **采用Quartus中已有的参数化模块来定制RAM**
7. 新建，编写源代码。

(1).选择保存项和芯片类型：【File】-【new project wizard】-【next】（设置文件路径+设置project name为LPM\_RAM）-【next】-【next】（family=CycloneⅡ；name=EP2C5T144C8）-【next】-【finish】

(2).新建：【file】-【new】（Block Diagram/Schematic File）-【OK】

1. 空白处双击呼出【Symbol】窗口，在【Symbol】元件库的【megafunctions】|【storage】中选择LPM\_RAM\_IO，将元件图添加到合适位置，并双击参数框设置参数，设置好对应input和output保证功能实现，保存文件(LPM\_RAM.bdf)
2. 创建初始化数据文件：【File】->【New】->【Memory Initialization File】，保存为对应文件名与路径，并设置好初始化数据
3. 编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译成功。

波形仿真及验证。新建一个vector waveform file。添加输入输出节点。(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all；【list】）-【>>】-【ok】-【ok】)。对输入输出进行排序后，选中部分节点右击选择Grouping对其进行整合。设置endtime和grid size（点击edit-选择end time/grid time-输入合适的时间）。设置各段输入点击保存按钮保存。（操作为：点击snap to grid -选中一个时间段-双击-Numeric or named value设置输入值）。然后Assignment-settings-simulator settings-mode先后设置为Functional/Timing进行功能仿真和时序仿真(功能仿真时需点击Processing->Generate

四、实验过程

1. 编译过程

* **源代码（VHDL设计）如下**

1. SM

library ieee;

use ieee.std\_logic\_1164.all;

entity sm is

    port(

        clk, Sm\_en: in std\_logic;

        z: out std\_logic

    );

end sm;

architecture sm of sm is

    signal sm:std\_logic:='0';

    begin

        process(clk, Sm\_en)

        begin

            if(clk'event and clk='0') then

                if(Sm\_en='1') then

                    z<=not sm;

                    sm<=not sm;

                else

                    z<=sm;

                end if;

            else

                sm<=sm;

            end if;

        end process;

    end architecture sm;

1. **PC(指令计数器)**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity program\_counter is

    port(

        ldpc, inpc, clk:in std\_logic;

        a:in std\_logic\_vector(7 downto 0);

        c:out std\_logic\_vector(7 downto 0)

    );

end program\_counter;

architecture program\_counter of program\_counter is

    signal adress: std\_logic\_vector(7 downto 0):="00000000";

    begin

        process(ldpc, inpc, clk, a)

        begin

            if(clk'event and clk='0') then

                if inpc='1' and ldpc='0' then

                    adress<=adress+"00000001";

                elsif(inpc='0' and ldpc='1') then

                    adress<=a;

                else

                end if;

            else

            end if;

        end process;

        c<=adress;

    end architecture program\_counter;

1. **通用寄存器组**

library ieee;

use ieee.std\_logic\_1164.all;

entity general\_registers is

    port(

        we, clk:in std\_logic;

        raa, rwba:in std\_logic\_vector(1 downto 0);

        i:in std\_logic\_vector(7 downto 0);

        s, d:out std\_logic\_vector(7 downto 0)

    );

end general\_registers;

architecture general\_registers of general\_registers is

    signal a:std\_logic\_vector(7 downto 0);

    signal b:std\_logic\_vector(7 downto 0);

    signal c:std\_logic\_vector(7 downto 0);

    signal temp:std\_logic\_vector(7 downto 0);

    begin

        process(we, clk, raa, rwba, i)

        begin

            if(we='0') then

                if(clk'event and clk='0') then

                    if(rwba="00") then

                        a<=i;

                    elsif(rwba="01") then

                        b<=i;

                    elsif(rwba="10") then

                        c<=i;

                    else

                        c<=i;

                    end if;

                else

                end if;

            else

            end if;

            if(raa="00") then

                s<=a;

            elsif(raa="01") then

                s<=b;

            elsif(raa="10") then

                s<=c;

            else

                s<=c;

            end if;

            if(rwba="00") then

                d<=a;

            elsif (rwba="01") then

                d<=b;

            elsif (rwba="10") then

                d<=c;

            else

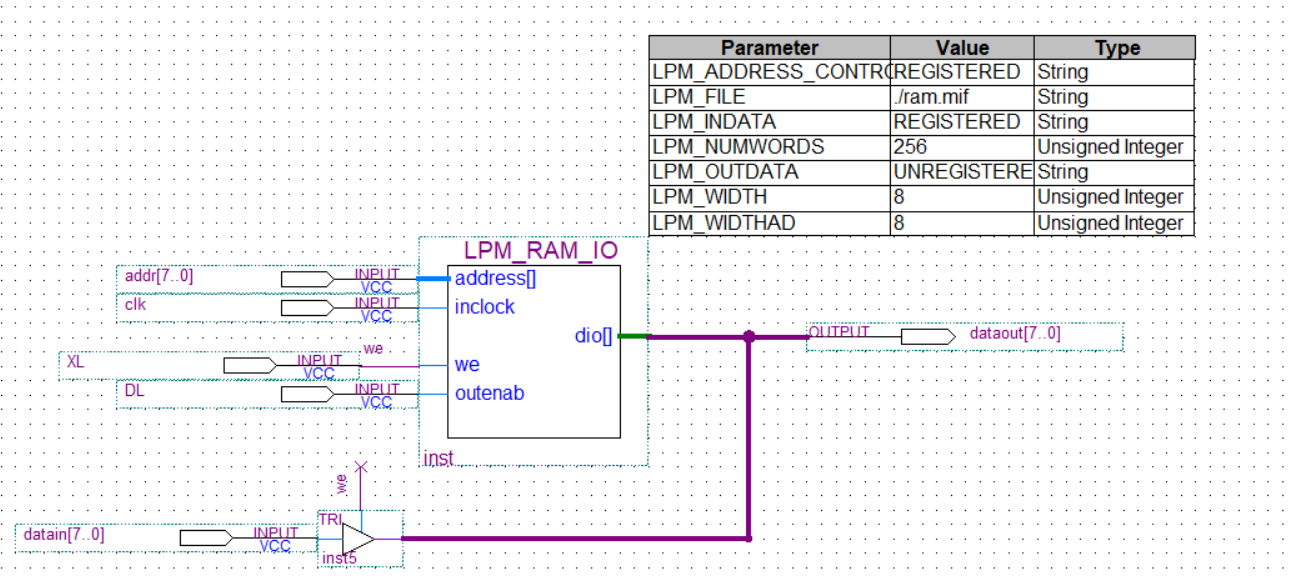
                d<=c;

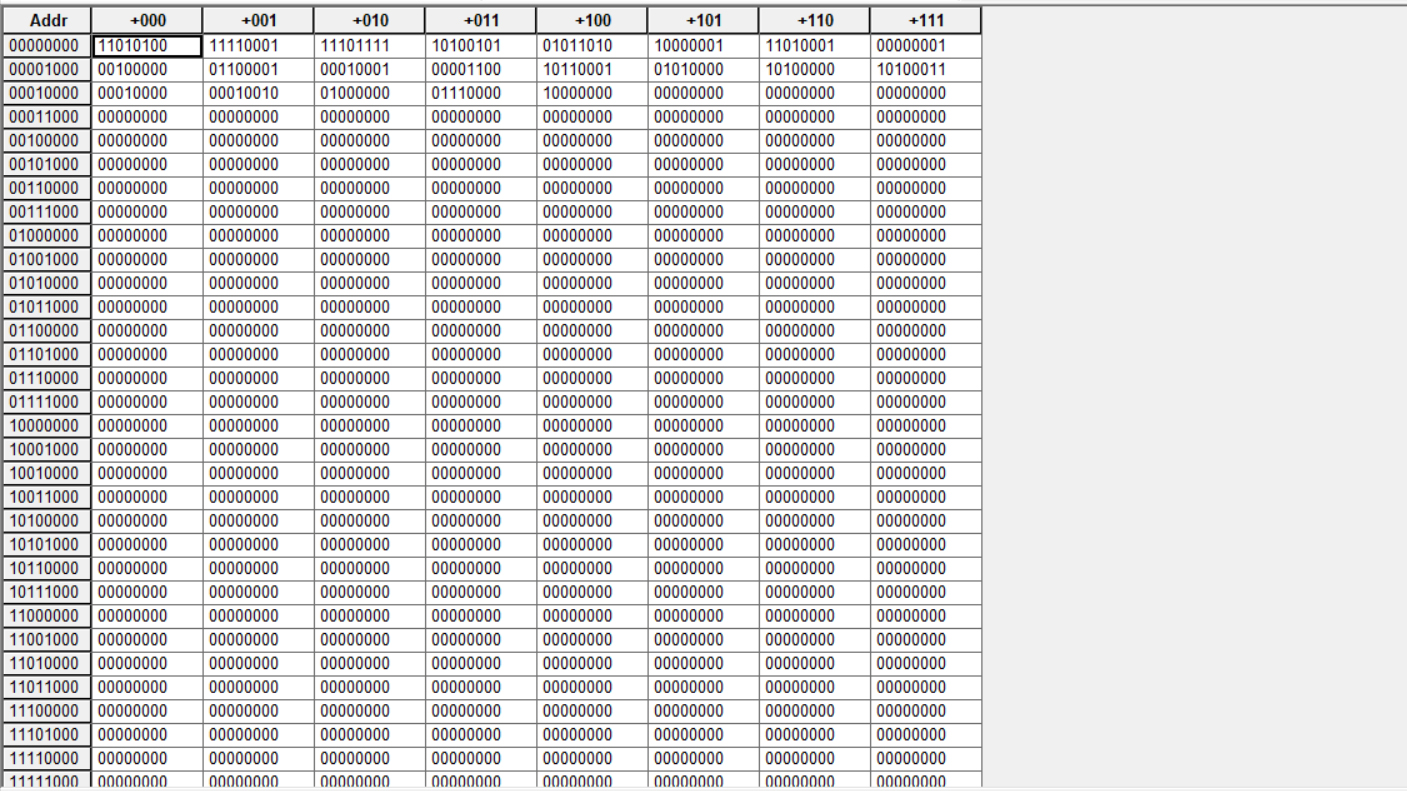
            end if;

        end process;

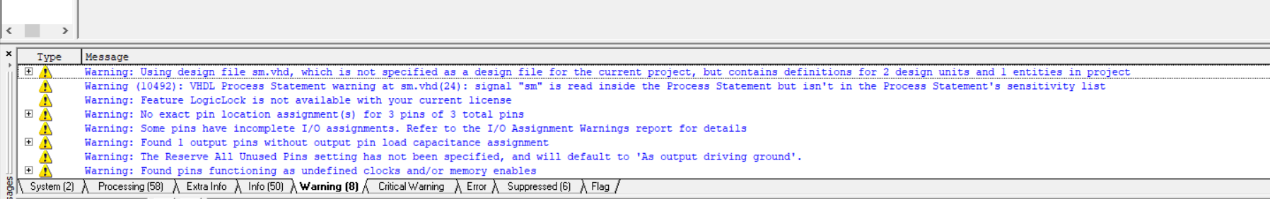
    end general\_registers;

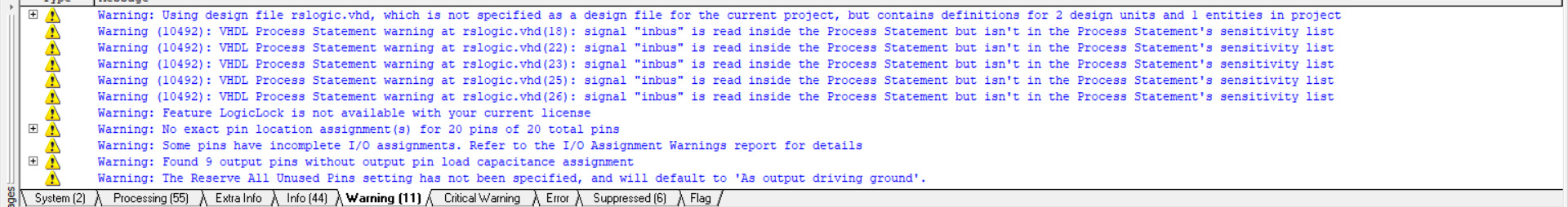
1. **RAM**

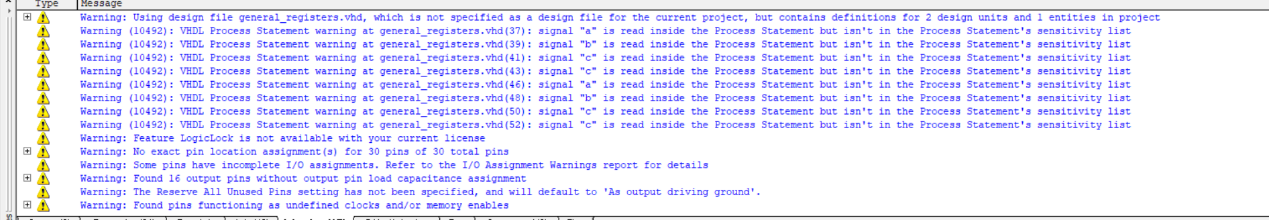


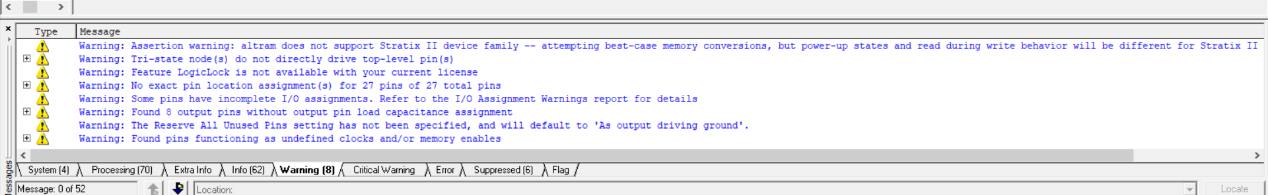
****

**b)编译、调试过程**





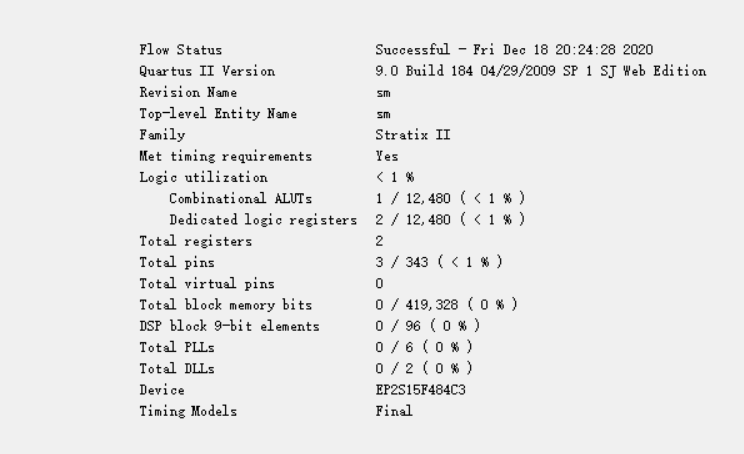


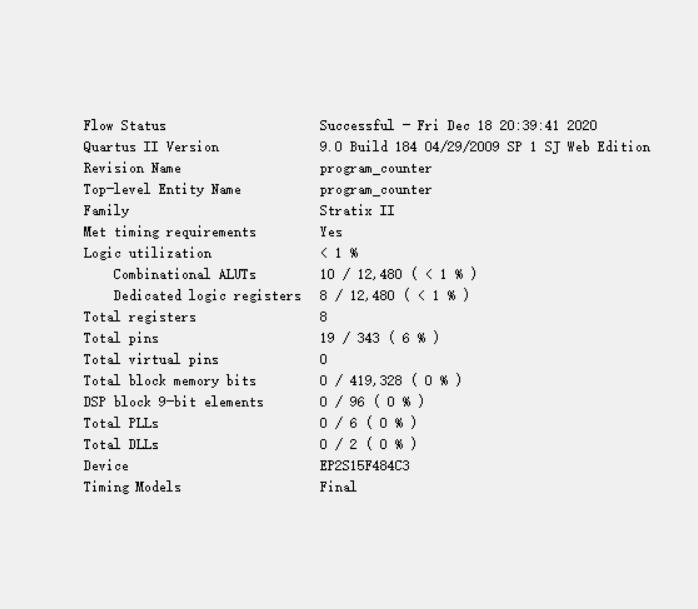


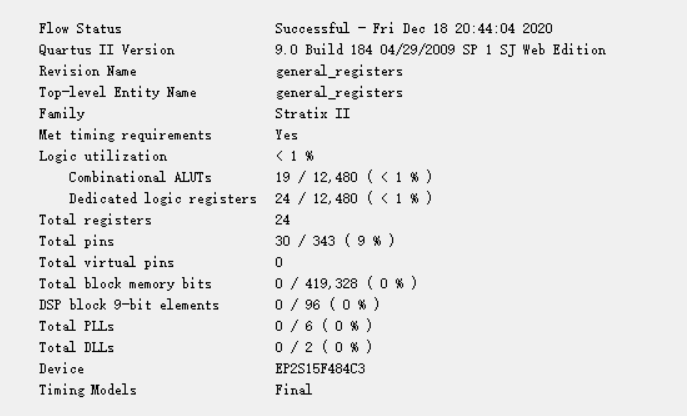
自上至下依次为SM、PC(指令计数器)、通用寄存器组、RAM的编译器提示信息。

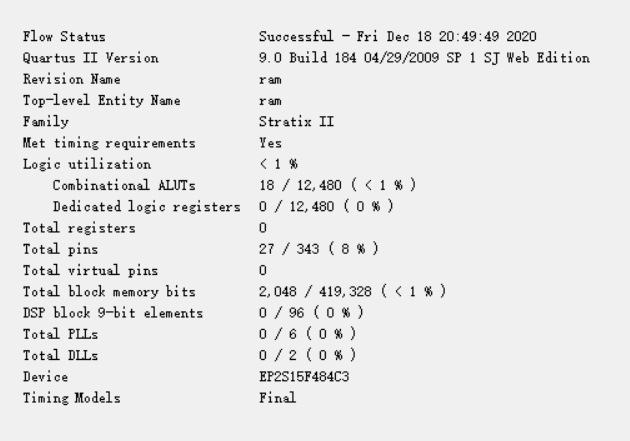
以上编译器均给出警告，但均能够编译通过。

* **资源消耗**





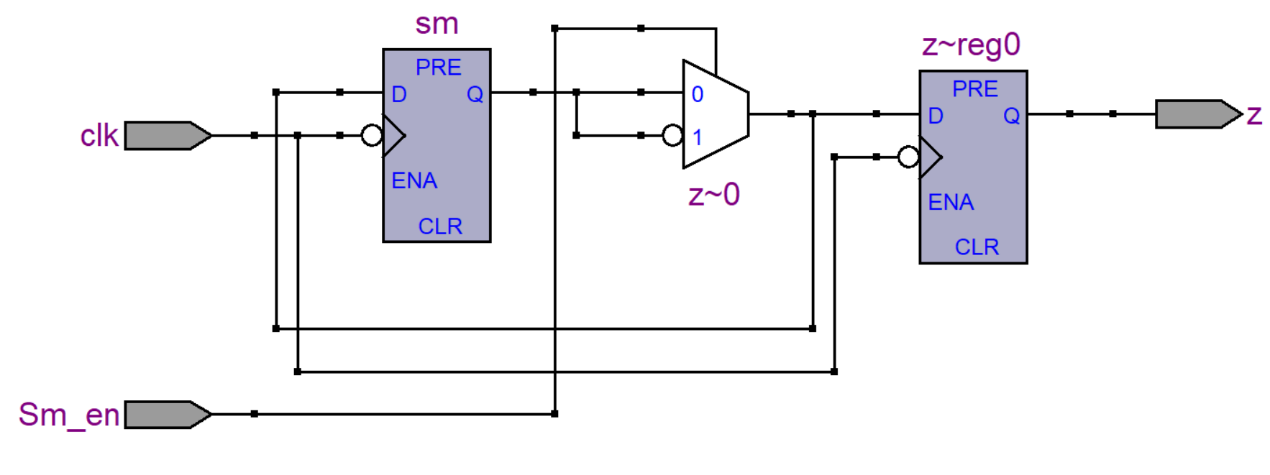




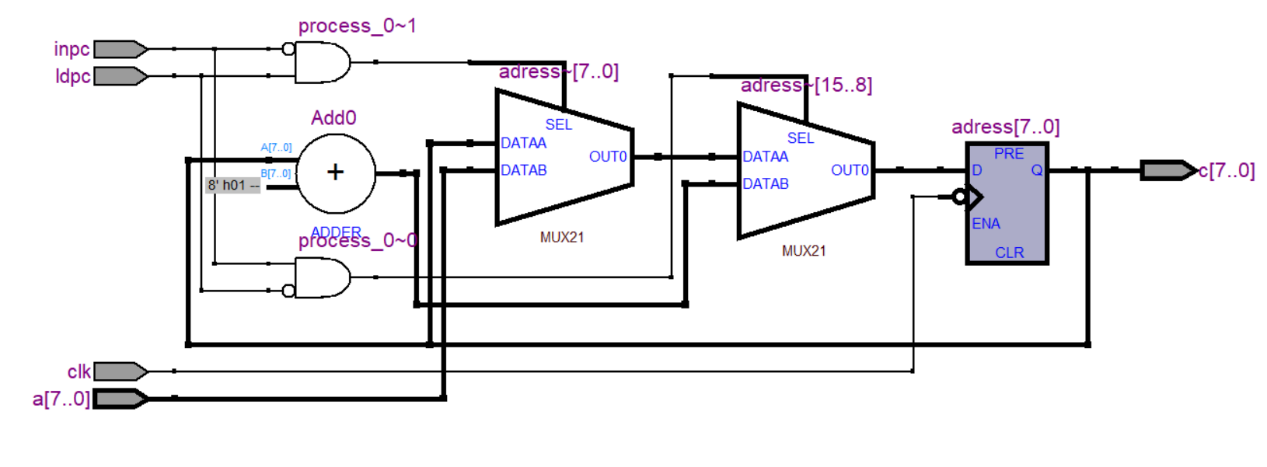
以上资源消耗基本都趋近于0.

**c) RTL视图**

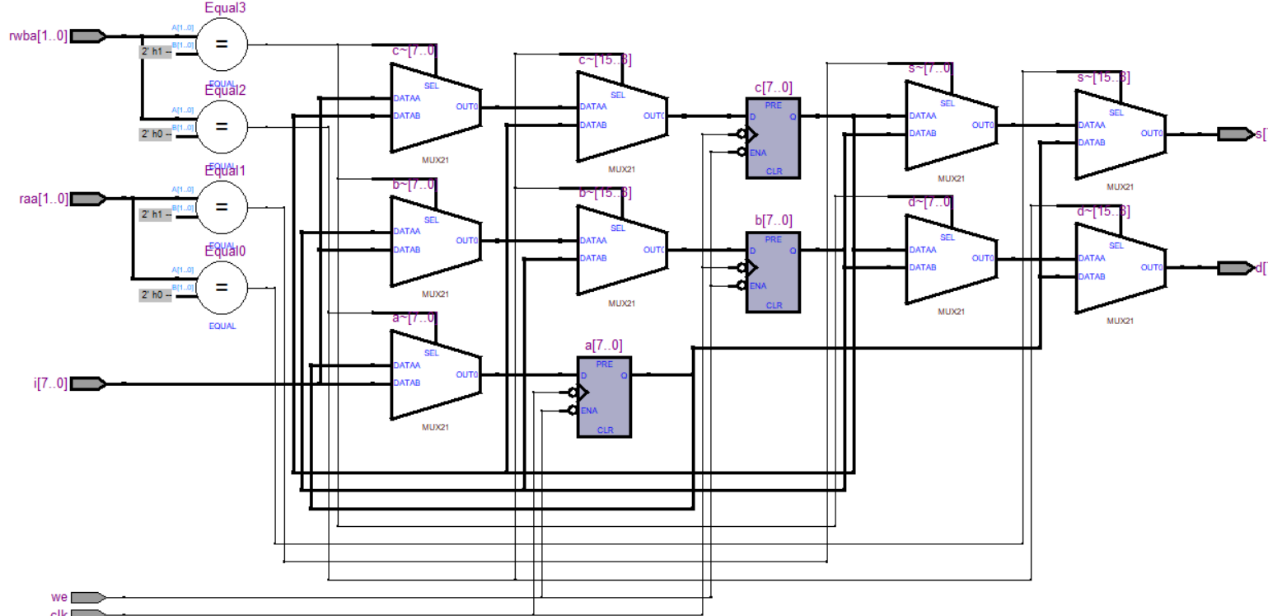
1. SM



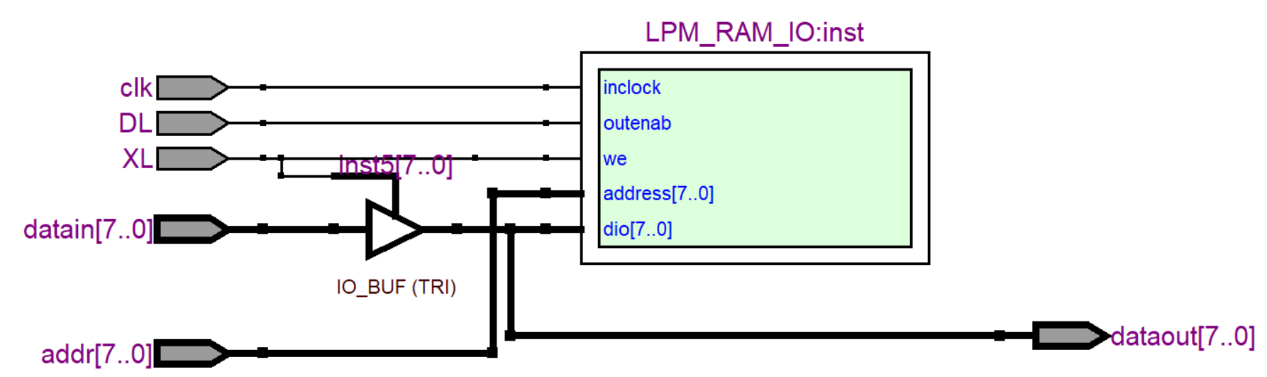
1. 指令计数器PC



1. 通用寄存器组



1. RAM



**d)结果分析及结论**

* SM

RTL显示该VHDL为带有使能信号的D触发器

无语法错误，资源占用正常

* 指令计数器PC

RTL显示该VHDL由加法器，一个触发器与多个多路复用器组成

无语法错误，资源占用正常

* 通用寄存器组

RTL显示该VHDL结构描述的结果为三个通用寄存器结合大量多路选择器的电路

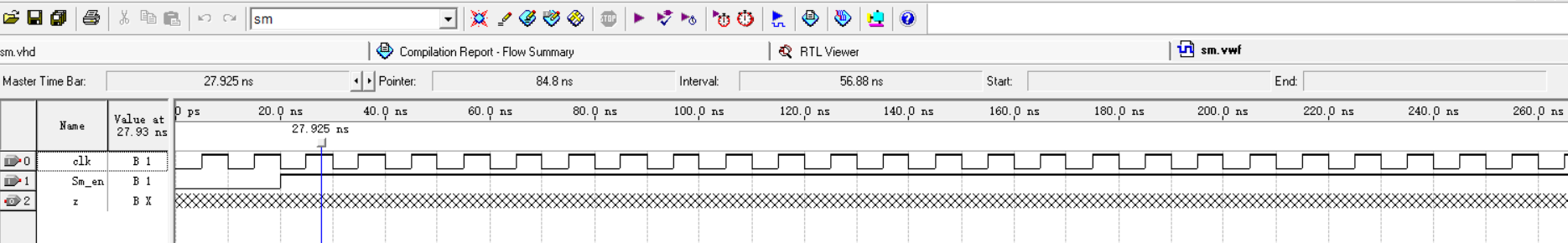
无语法错误，资源占用正常

* RAM

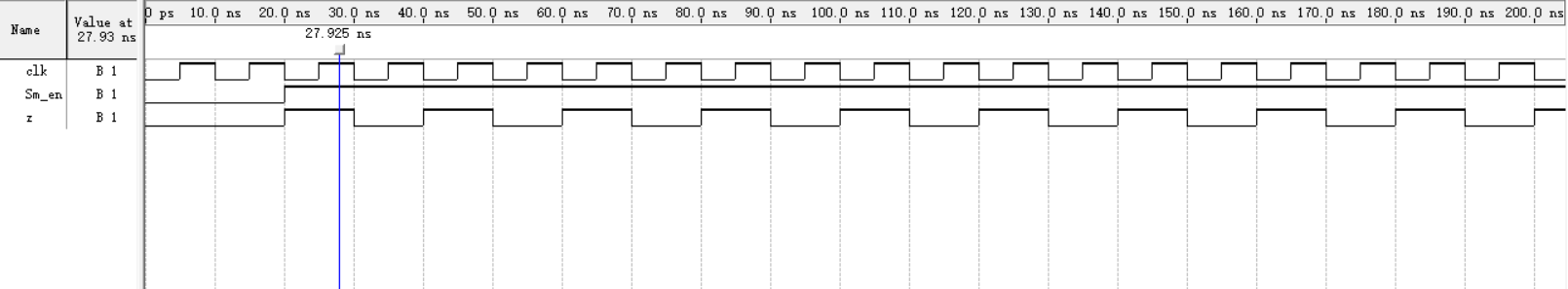
RTL显示与该电路原理图显示一致

无语法错误，资源占用正常

1. 波形仿真
2. **SM**
3. 波形仿真过程（过程详见实验步骤）



1. 波形仿真波形图



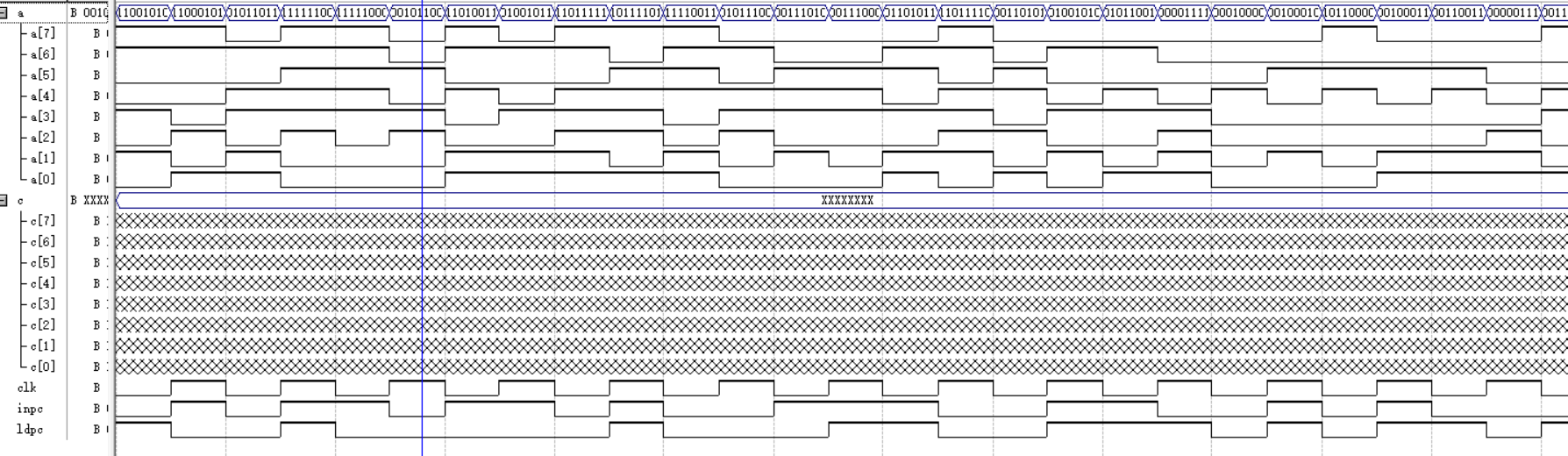
1. 结果分析及结论

以第一个周期为例

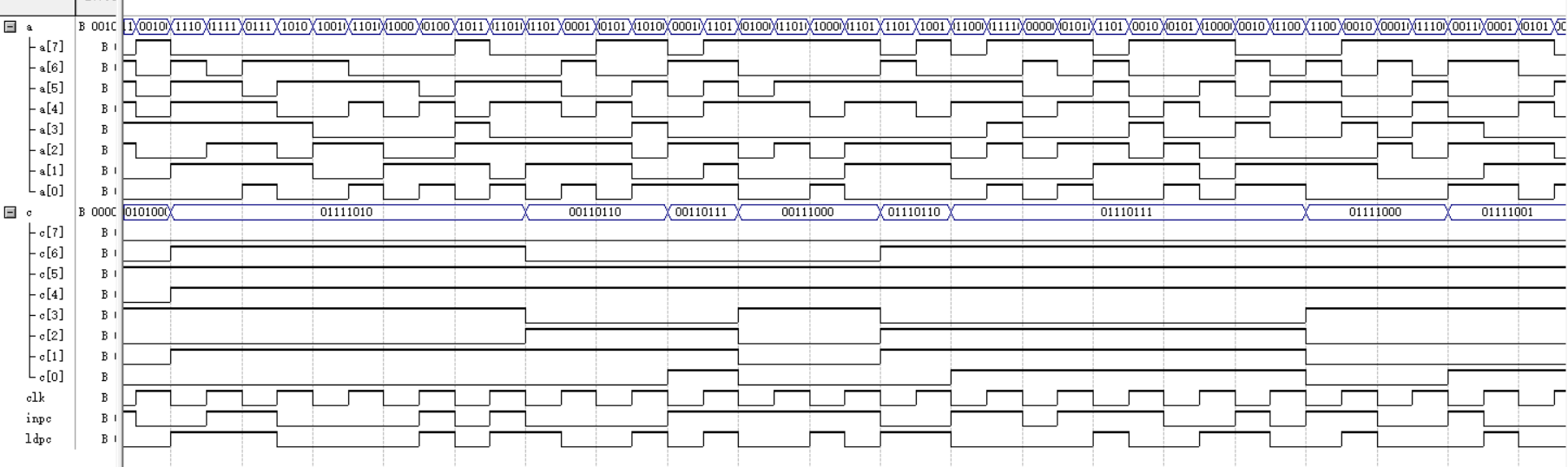
clk输入为下降沿:Sm\_en为1,z取反，否则Sm\_en为0.z输出不变,如图20.0ns

结果正确

1. **指令计数器PC**
2. 波形仿真过程（过程详见实验步骤）



1. 波形仿真波形图



1. 结果分析及结论

0-75ns：加载使能ldpc为0，自增使能inpc为0，输入地址为00000000，此时既不加载也不自增，输出地址为00000000

75-125ns：加载使能ldpc为1，自增使能inpc为1，输入地址为00000000，此时既不加载也不自增，输出地址为00000000

125-175ns：加载使能ldpc为0，自增使能inpc为0，输入地址为00000000，此时既不加载也不自增，输出地址为00000000

175-200ns：加载使能ldpc为0，自增使能inpc为1，输入地址为00000000，此时未到下降沿，输出地址为00000000

200-250ns：加载使能ldpc为0，自增使能inpc为1，输入地址为00000000，此时地址自增作为下一次输出，本次输出地址为0000000

250-300ns：加载使能ldpc为0，自增使能inpc为1，输入地址为00000000，此时地址自增作为下一次输出，本次输出地址为0000001

300-325ns：加载使能ldpc为0，自增使能inpc为1，输入地址为00000000，此时地址自增作为下一次输出，本次输出地址为0000010

325-350ns：加载使能ldpc为1，自增使能inpc为0，输入地址为11111110，此时未到下降沿，输出地址为00000010

350-375ns：加载使能ldpc为1，自增使能inpc为0，输入地址为11111110，此时地址跳转为输入地址，并使下一地址为输入地址+1，输出地址为11111110

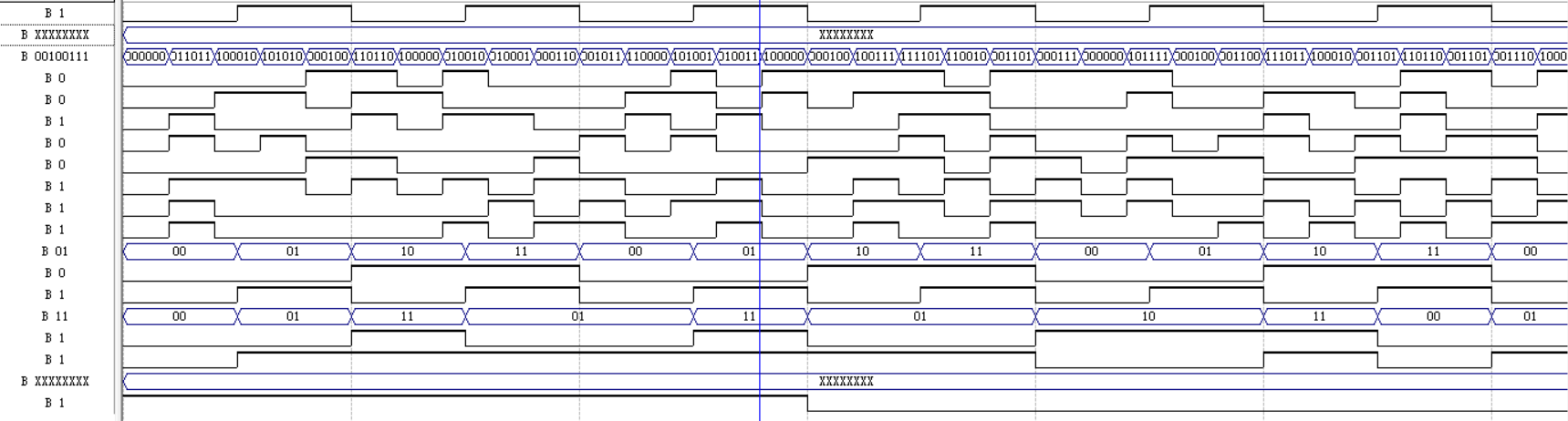
375-400ns：加载使能ldpc为0，自增使能inpc为1，输入地址为00000000，此时未到下降沿，输出地址为11111110

400-450ns：加载使能ldpc为0，自增使能inpc为1，输入地址为00000000，此时地址自增作为下一次输出，本次输出地址为11111111

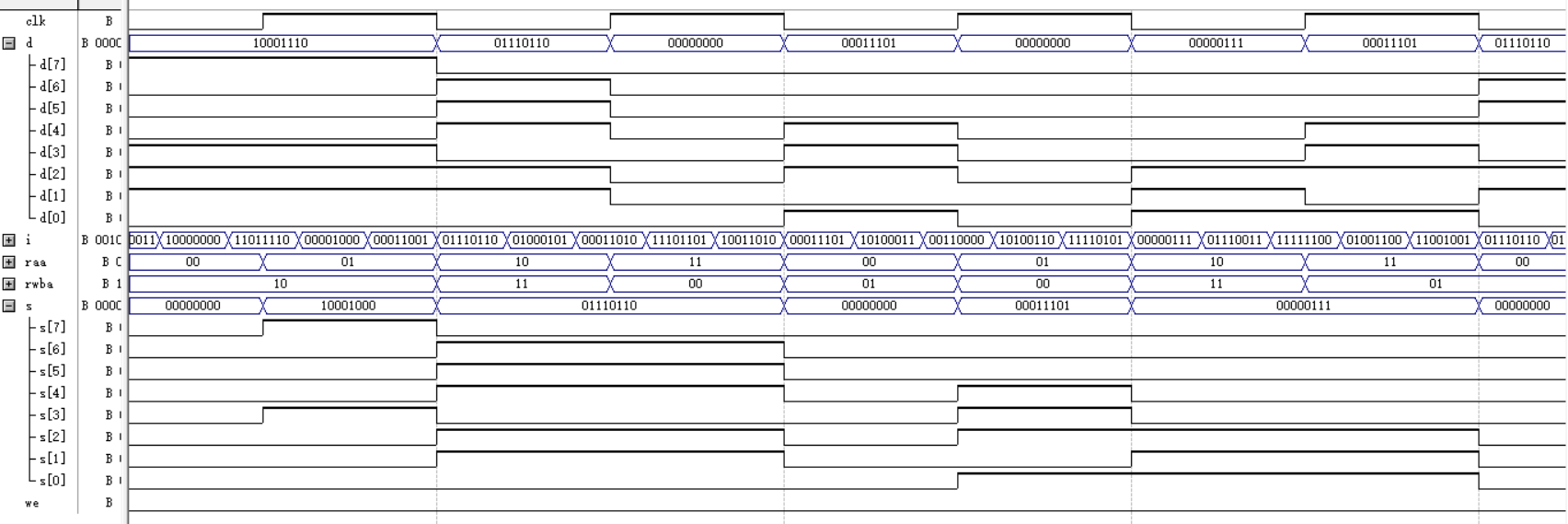
450-500ns：加载使能ldpc为0，自增使能inpc为1，输入地址为00000000，此时地址自增作为下一次输出，由于已到11111111，再自增后本次输出地址为0000000

结果正确

1. **控制信号产生逻辑**
2. 波形仿真过程（过程详见实验步骤）



1. 波形仿真波形图



1. 结果分析及结论

0-50ns：WE为0，RA为11，WA为00，data为10110001，输出AO与BO均为高阻态。此时为写入数据data：10110001到寄存器A中

50-100ns：WE为0，RA为11，WA为01，data为11110001，输出AO与BO均为高阻态。此时为写入数据data：11110001到寄存器B中

100-150ns：WE为0，RA为11，WA为10，data为11010001，输出AO与BO均为高阻态。 此时为写入数据data：11010001到寄存器C中

150-200ns：WE为1，RA为00，WA为01，data为00000000，输出AO为10110001，BO为11110001。此时从寄存器A中读取数据10110001输出到AO，从寄存器B中读取数据11110001输出到BO

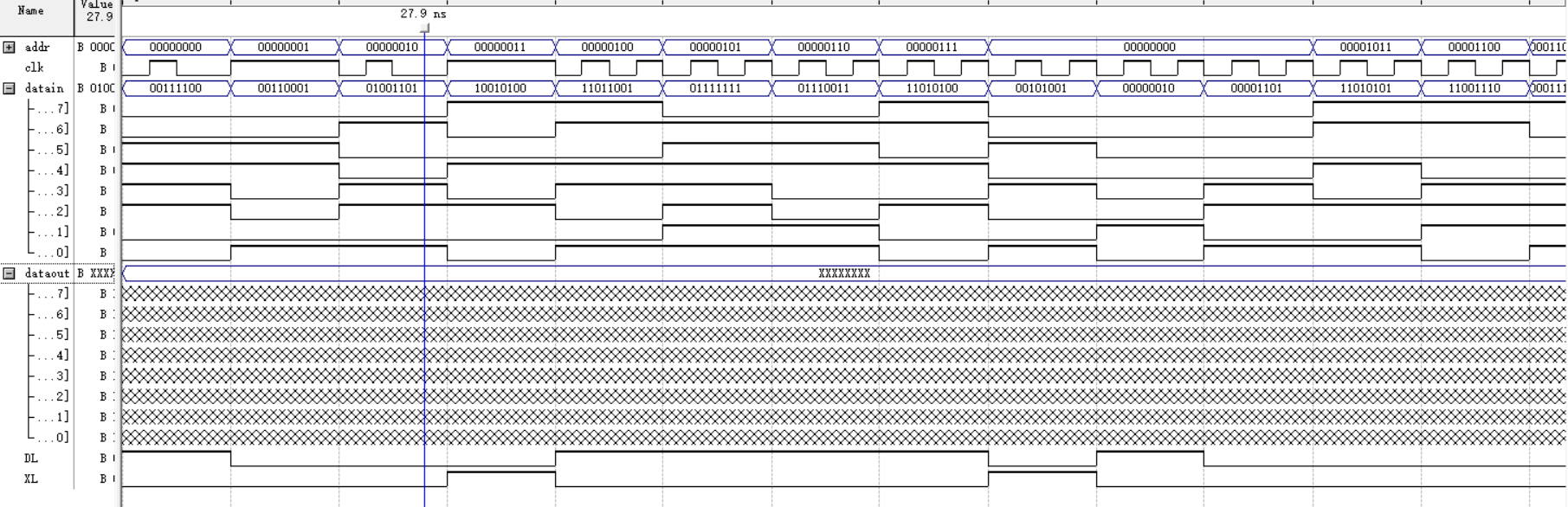
200-250ns：WE为1，RA为01，WA为10，data为00000000，输出AO为11110001，BO为11010001。此时从寄存器B中读取数据11110001输出到AO，从寄存器C中读取数据11010001输出到BO

250-300ns：WE为1，RA为10，WA为00，data为00000000，输出AO为11010001，BO为10110001。此时从寄存器C中读取数据11010001输出到AO，从寄存器A中读取数据10110001输出到BO

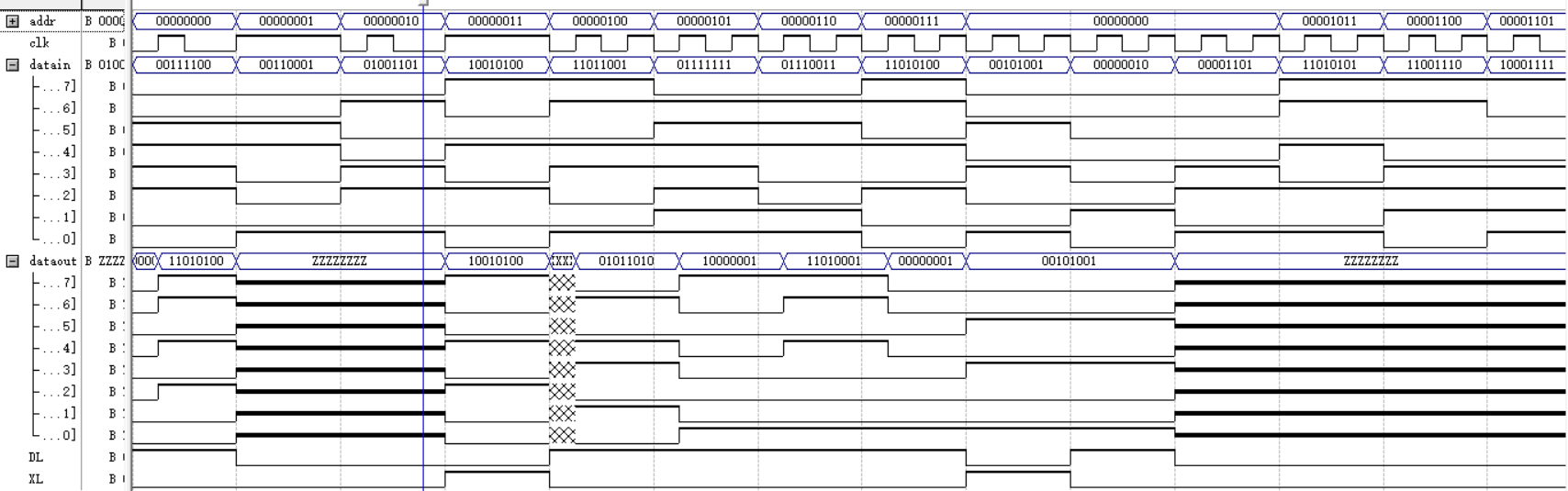
300-350ns：WE为1，RA为11，WA为11，data为00000000，输出AO为11010001，BO为11010001。此时从寄存器C中读取数据11010001输出到AO，从寄存器C中读取数据11010001输出到BO

350ns-之后：WE为0，AO与BO恒为高阻态

1. **RAM**
2. 波形仿真过程（过程详见实验步骤）



1. 波形仿真波形图



1. 结果分析及结论

0-25ns：读信号DL为1，写信号XL为0，输入地址00000000，输入数据00000000，未到 时钟上升沿，输出为00000000

25-75ns:读信号DL为1，写信号XL为0，输入地址00000000，输入数据00000000，输出原

本初始化在00000000地址上的数据11010100

75-125ns：读信号DL为1，写信号XL为0，输入地址00000001，输入数据00000000，输 出原本初始化在00000001地址上的数据11110001

125-175ns：读信号DL为1，写信号XL为0，输入地址00000010，输入数据00000000，输出原本初始化在00000010地址上的数据11101111

175-225ns：读信号DL为1，写信号XL为0，输入地址00000011，输入数据00000000，输出原本初始化在00000011地址上的数据10100101

225-275ns：读信号DL为1，写信号XL为0，输入地址00000100，输入数据00000000，输出原本初始化在00000100地址上的数据01011010

275-325ns：读信号DL为1，写信号XL为0，输入地址00000101，输入数据00000000，输出原本初始化在00000101地址上的数据11110001

325-375ns：读信号DL为1，写信号XL为0，输入地址00000110，输入数据00000000，输出原本初始化在00000110地址上的数据10000001

375-400ns：读信号DL为1，写信号XL为0，输入地址00000111，输入数据00000000，输出原本初始化在00000111地址上的数据00000001

400-450ns：读信号DL为0，写信号XL为1，输入地址00000000，输入数据10101010，将地址00000000上的数据修改为10101010，并输出输入数据

450-500ns：读信号DL为1，写信号XL为0，输入地址00000000，输入数据00000000，输出经过修改后在00000000地址上的数据10101010

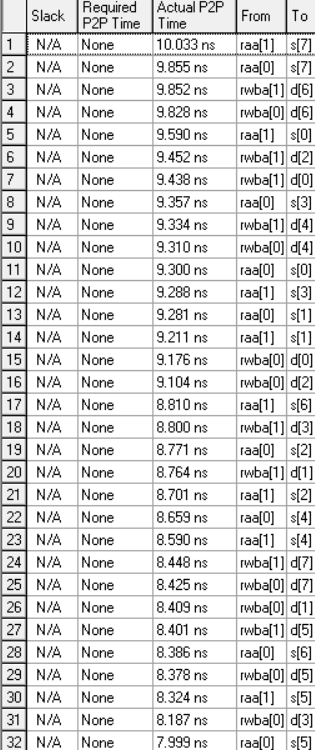
1. 时序仿真

* SM和指令计数器PC没有tpd
* 通用寄存器组

1. 时序仿真过程

做好上述步骤后，编译【classic timing analysis】-在compilation report中选择【timing analysis】-【tpd】（引脚到引脚的延时）

1. 时序仿真图



1. 结果分析及结论

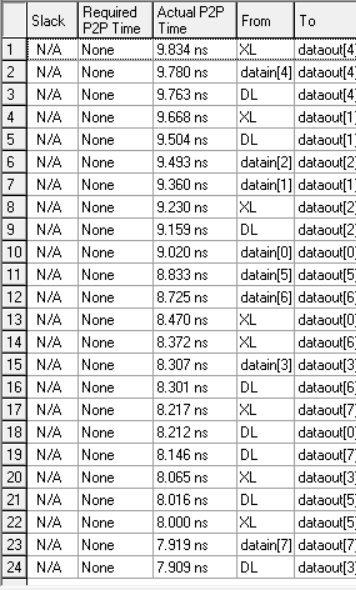
每个引脚只间相互传递产生的延时各不相同，挑选其中p2p时间的最大值，为raa [1]传递给s[7]。为10.033ns，故整体延时为10.033ns。

* RAM

1. 时序仿真过程

做好上述步骤后，编译【classic timing analysis】-在compilation report中选择【timing analysis】-【tpd】（引脚到引脚的延时）

1. 时序仿真图



1. 结果分析及结论

每个引脚只间相互传递产生的延时各不相同，挑选其中p2p时间的最大值，为XL传递给dataout[4]。为9.834ns，故整体延时为9.834ns。

tpd (引脚到引脚的延时)

五、实验结论

1、思考题

1. ①防止PC，SM，寄存器中的操作超前进行，快于RAM输出数据。会导致对RAM中

数据的处理会少一位，而导致某些行为结果错误或根本无法执行导致CPU崩溃

②该CPU需要一个周期取指令，一个周期执行指令，首先在第一个周期中的下降沿时

在PC中取指令，然后在上升沿由RAM读取到指令寄存器中，与此同时，PC中的地

址自动+1;然后在下降沿SM指令执行，所以部分为上升沿执行，部分为下降沿执行。

1. 库引入、实体声明、端口方向、结构体、库，程序包的调用、进程语句

Process语句: 进程语句，用以包装顺序语句.使整个顺序语句可以与其他并行语句或

Process并行

If语句：在不同判断条件下给信号或变量赋值

尤其是if (clk’event and clk = ‘1’) if (clk’event and clk = ‘0’)表示上升沿下降沿

条件信号赋值语句

When-Case语句: 在判断条件有限的情况下可以使用该语句，在不同情况给信号或变量

赋值

when-else语句：为并行语句，无需Process包装，在不同情况给信号或变量赋值

1. 实验总结与实验心得

本次实验的内容与要求是简易模型机中时序器件的VHDL语言实现以及理解简易模型机的结构和工作原理。涉及的能力有VHDL组合逻辑电路与时序电路编程能力，时序电路的设计与分析能力，实验报告的撰写能力，实验过程分析总结能力。通过使用软件实现硬件，既培养了编码能力，又增加了硬件设计能力和社会实践能力。该次实验使我对时序电路的设计与CPU中命令的传递与选择有了更深刻的认识，巩固了时序电路的设计相关的知识点。我因此受益良多。