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

1. SM

1、 新建，编写源代码。

⑴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】

⑵. 新建：【file】-【new】（第二个 VHDL File）-【OK】

2、 写好源代码，保存文件（SM.vhd）。

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，编译结果有警告，编译成功。

4、 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。

5、 时序仿真和功能仿真。
6、 查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

2. PC(指令计数器)

1、 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】
2、 写好源代码，保存文件（program_counter.vhd）。
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，编译结果有警告，编译成功。
4、 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。
5、 时序仿真和功能仿真。
6、 查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

3. 通用寄存器组

1. 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】
2. 写好源代码，保存文件（general_registers.vhd）。
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，编译结果有警告，编译成功。
4. 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。
5. 时序仿真和功能仿真。
6. 查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

4. 采用 Quartus 中已有的参数化模块来定制 RAM

1. 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=Cyclone Ⅱ；name=EP2C5T144C8）-【next】-【finish】

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

2. 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)
3. 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
4. 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

a) 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;
```

2. 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;
```

3. 通用寄存器组

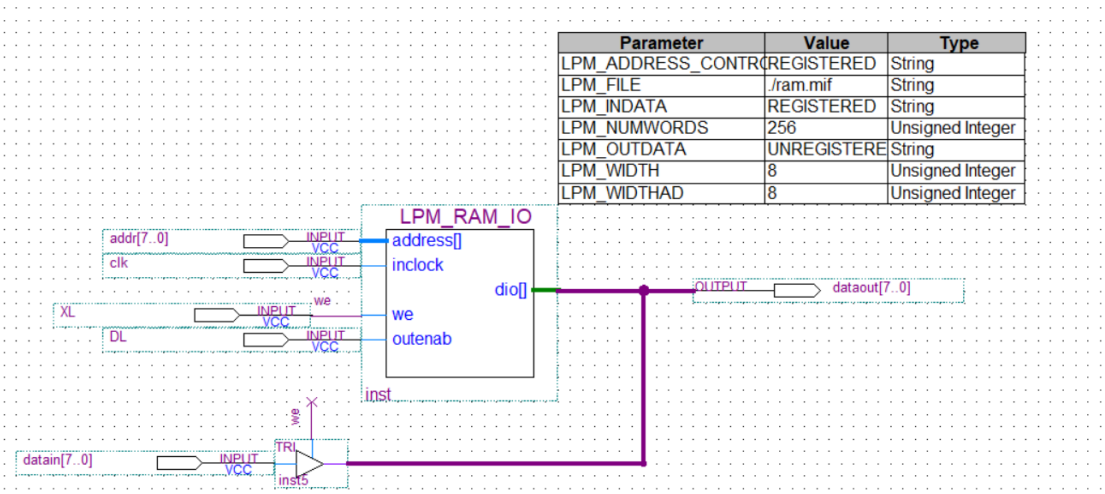
```
library ieee;
use ieee.std_logic_1164.all;
```

```
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;
```

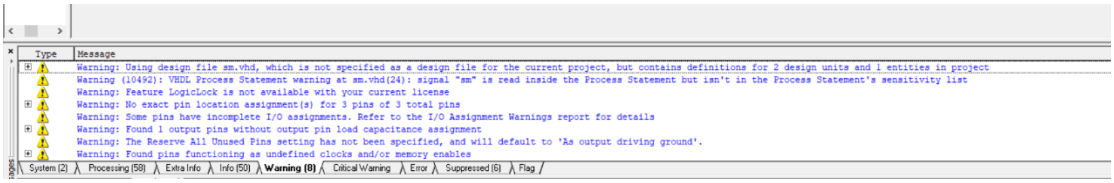
```
        end if;
        if(rwba="00") then
            d<=a;
        elsif (rwba="01") then
            d<=b;
        elsif (rwba="10") then
            d<=c;
        else
            d<=c;
        end if;
    end process;
end general_registers;
```

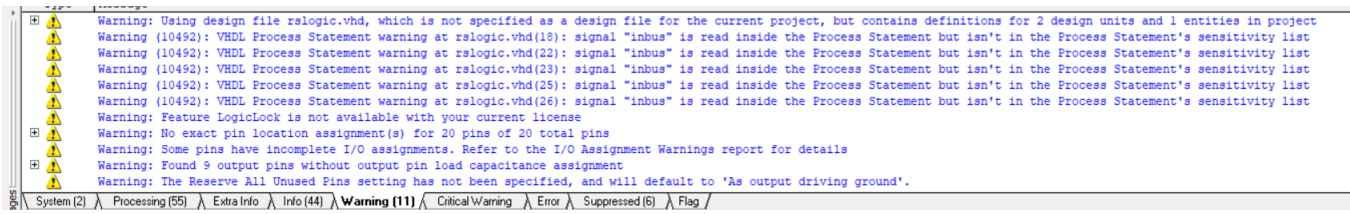
4. RAM

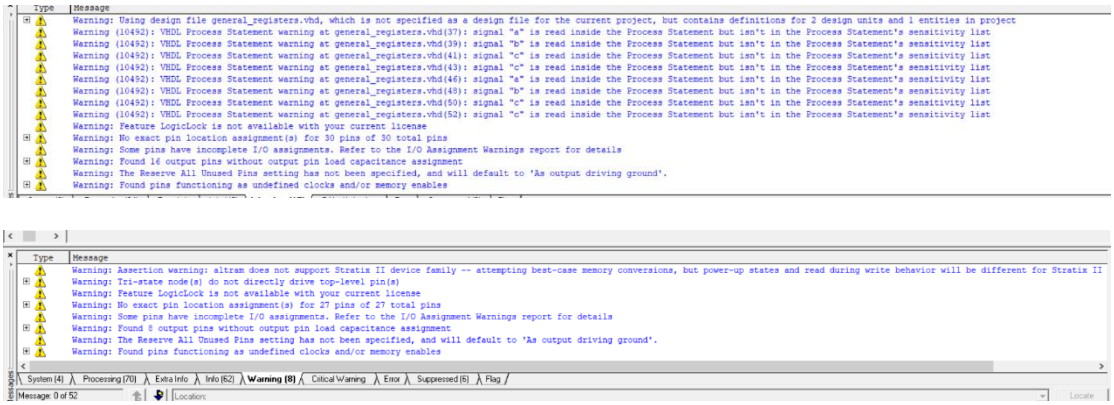


Addr	+000	+001	+010	+011	+100	+101	+110	+111
00000000	11010100	11110001	11101111	10100101	01011010	10000001	11010001	00000001
00001000	00100000	01100001	00010001	00001100	10110001	01010000	10100000	10100011
00010000	00010000	00010010	01000000	01110000	10000000	00000000	00000000	00000000
00011000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00100000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00101000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00110000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00111000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01001000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01010000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01011000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01100000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01101000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01110000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
01111000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10001000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10010000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10011000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10100000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10101000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10110000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10111000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11001000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11010000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11011000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11100000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11101000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11110000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
11111000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

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**

Flow Status	Successful - Fri Dec 18 20:24:28 2020
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	sm
Top-level Entity Name	sm
Family	Stratix II
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	1 / 12,480 (< 1 %)
Dedicated logic registers	2 / 12,480 (< 1 %)
Total registers	2
Total pins	3 / 343 (< 1 %)
Total virtual pins	0
Total block memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Flow Status	Successful - Fri Dec 18 20:39:41 2020
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	program_counter
Top-level Entity Name	program_counter
Family	Stratix II
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	10 / 12,480 (< 1 %)
Dedicated logic registers	8 / 12,480 (< 1 %)
Total registers	8
Total pins	19 / 343 (6 %)
Total virtual pins	0
Total block memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

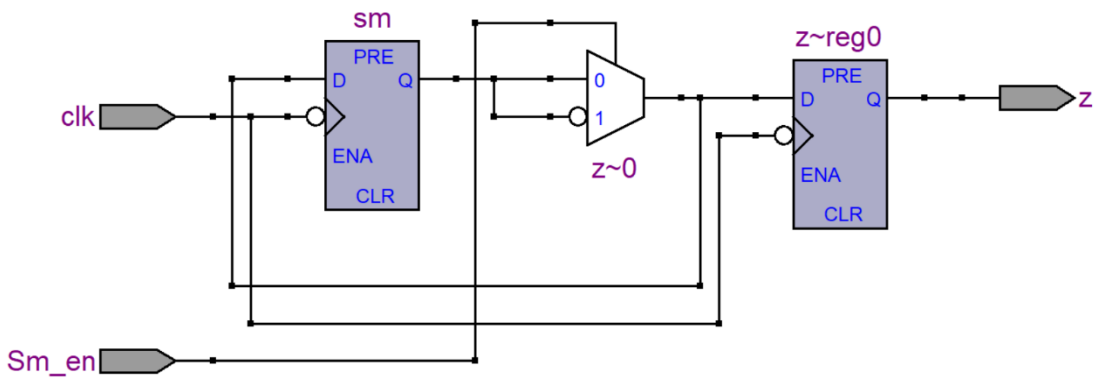
Flow Status	Successful - Fri Dec 18 20:44:04 2020
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	general_registers
Top-level Entity Name	general_registers
Family	Stratix II
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	19 / 12,480 (< 1 %)
Dedicated logic registers	24 / 12,480 (< 1 %)
Total registers	24
Total pins	30 / 343 (9 %)
Total virtual pins	0
Total block memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Flow Status	Successful - Fri Dec 18 20:49:49 2020
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	ram
Top-level Entity Name	ram
Family	Stratix II
Met timing requirements	Yes
Logic utilization	< 1 %
Combinational ALUTs	18 / 12,480 (< 1 %)
Dedicated logic registers	0 / 12,480 (0 %)
Total registers	0
Total pins	27 / 343 (8 %)
Total virtual pins	0
Total block memory bits	2,048 / 419,328 (< 1 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

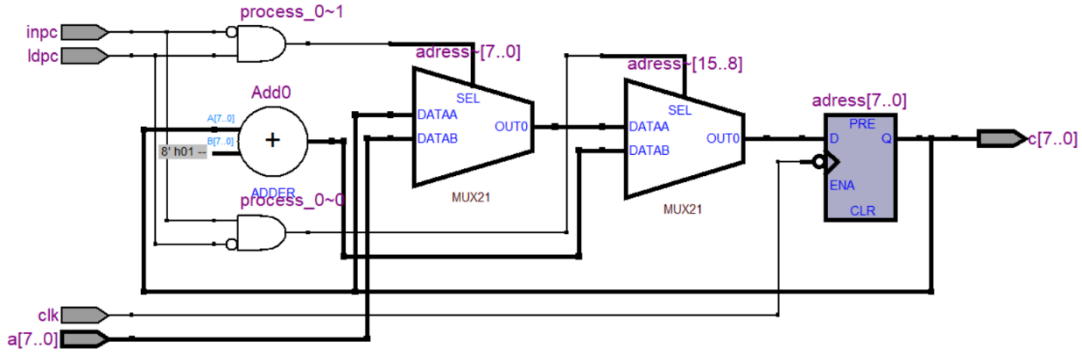
以上资源消耗基本都趋近于 0.

c) RTL 视图

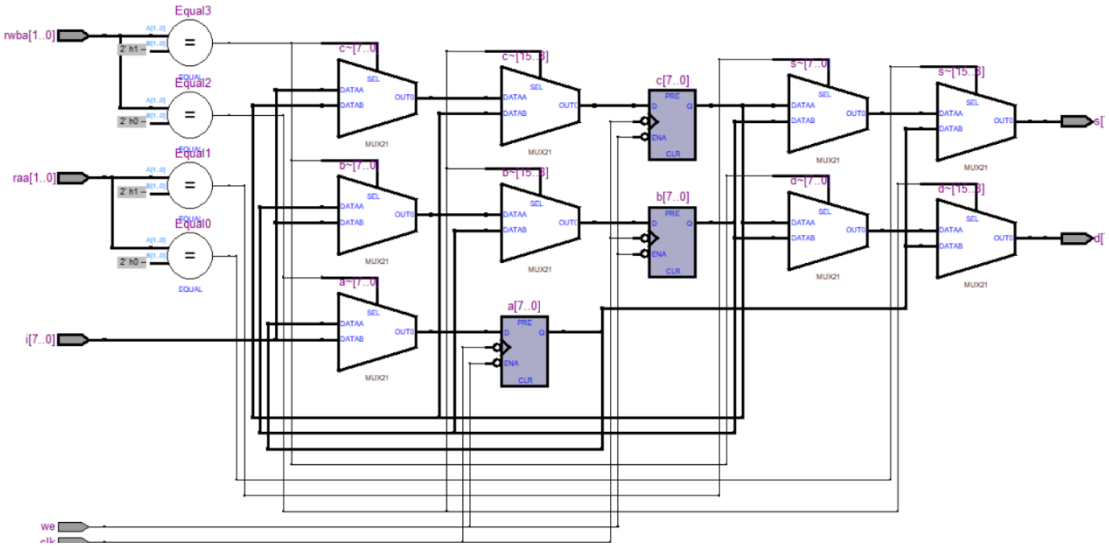
1. SM



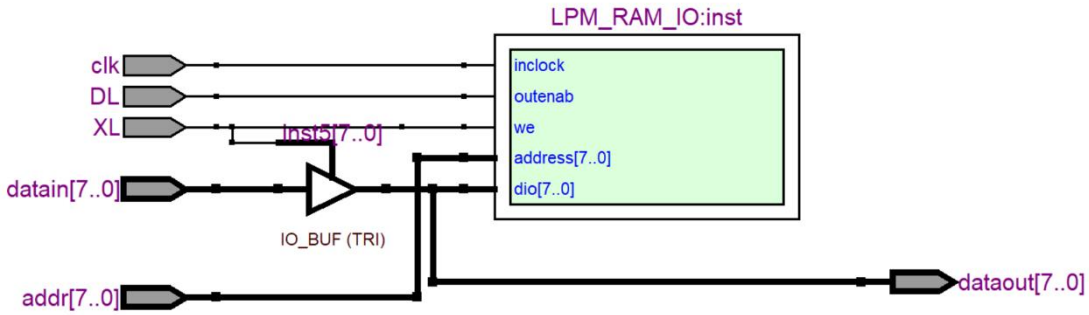
2. 指令计数器 PC



3. 通用寄存器组



4. 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

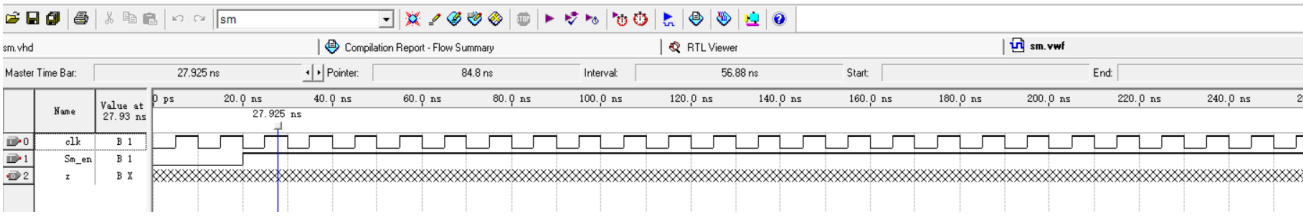
  无语法错误，资源占用正常

- 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

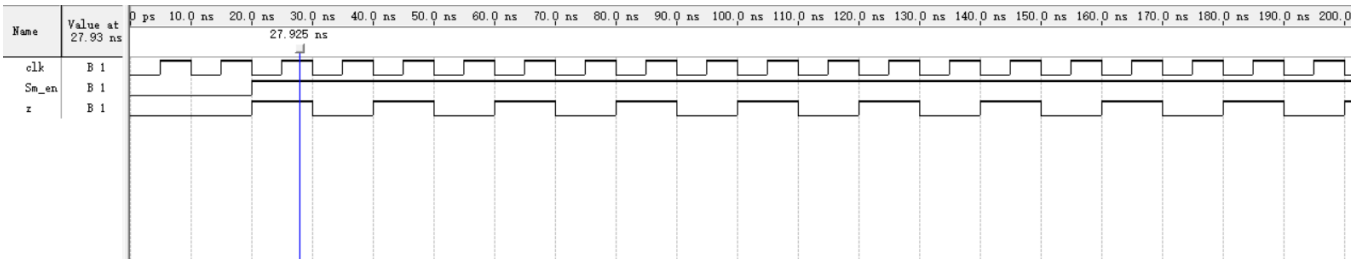
d) 波形仿真

1. SM

a) 波形仿真过程（过程详见实验步骤）



b) 波形仿真波形图



c) 结果分析及结论

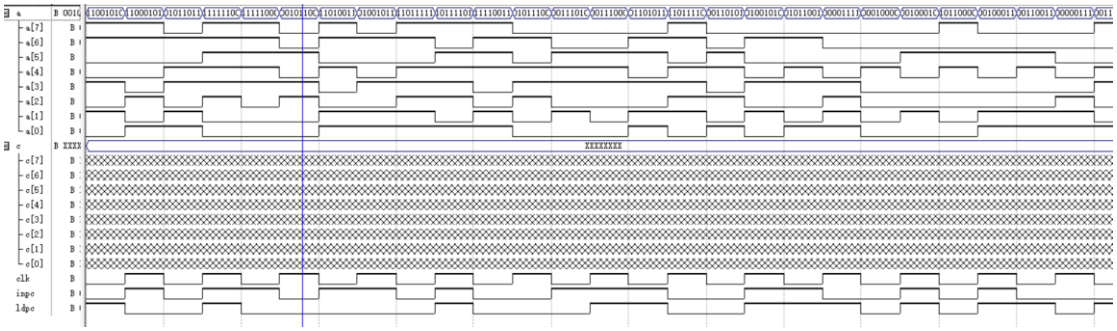
以第一个周期为例

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

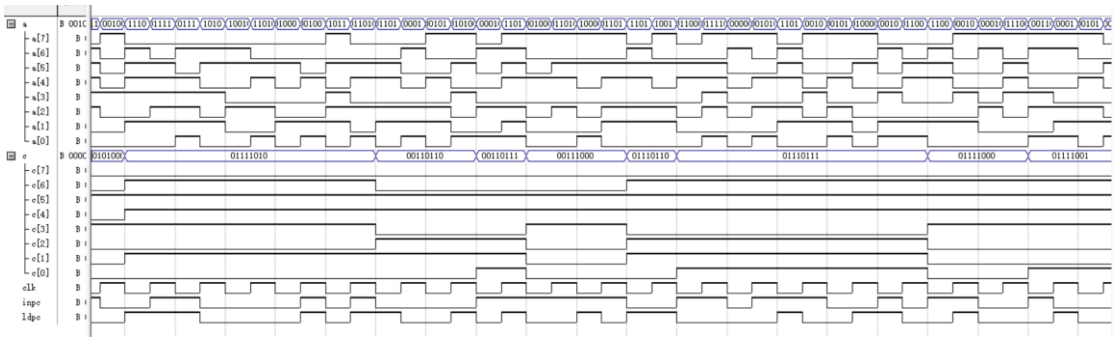
结果正确

2. 指令计数器 PC

a) 波形仿真过程（过程详见实验步骤）



b) 波形仿真波形图



c) 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

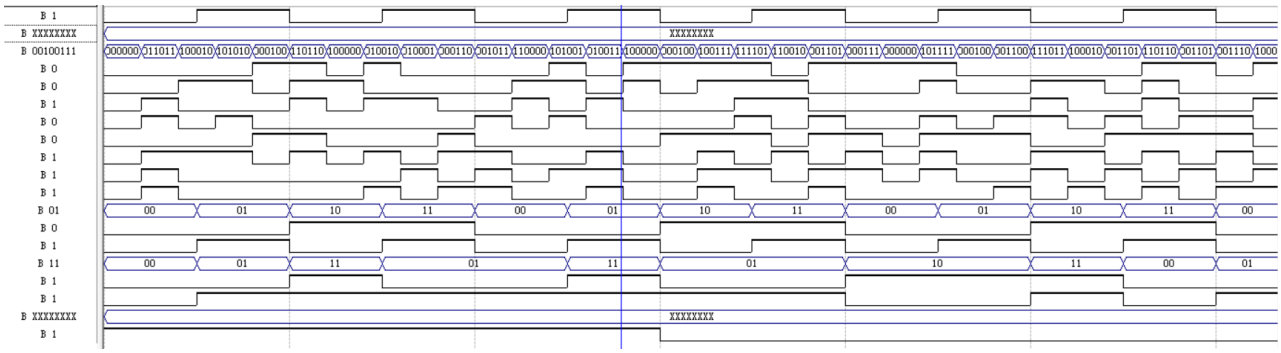
400-450ns：加载使能 ldpc 为 0，自增使能 inpc 为 1，输入地址为 00000000，此时地址自增作为下一次输出，本次输出地址为 11111111

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

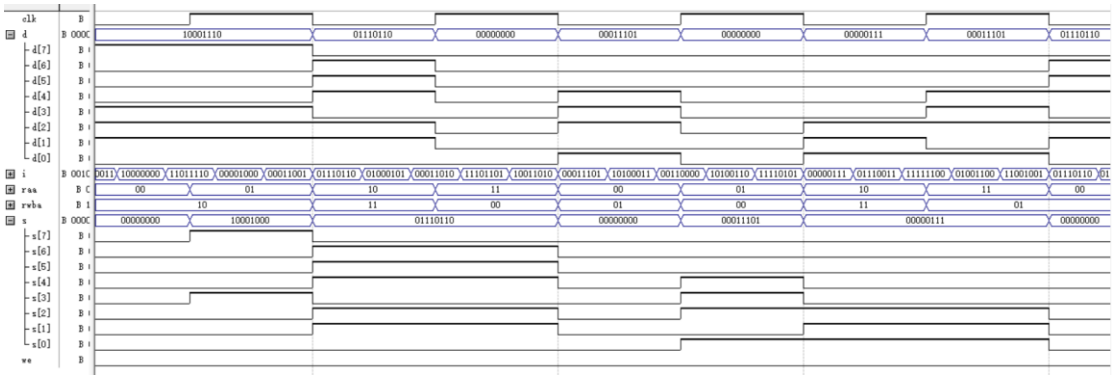
结果正确

3. **控制信号产生逻辑**

a) 波形仿真过程（过程详见实验步骤）



b) 波形仿真波形图



c) 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

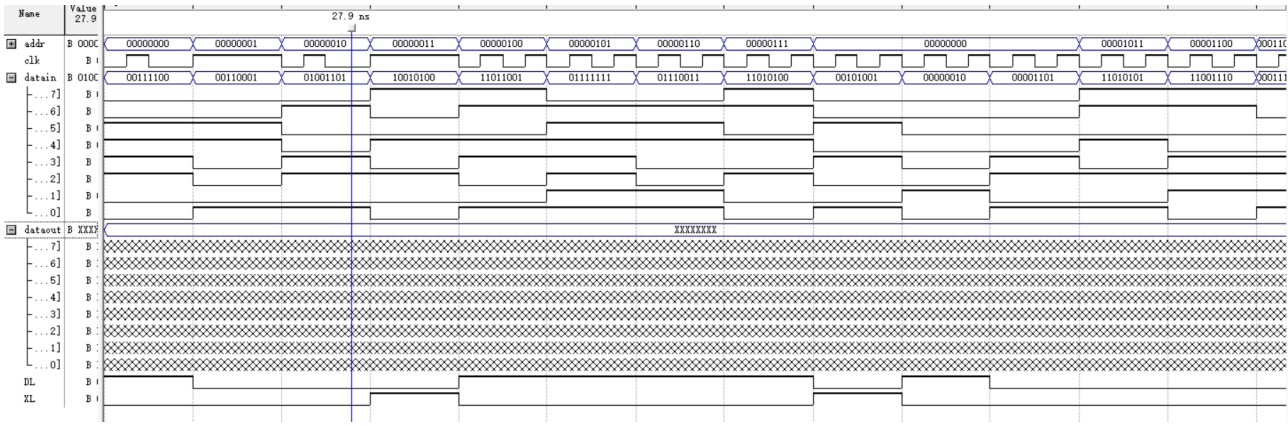
250-300ns：WE 为 1，RA 为 10，WA 为 00，data 为 00000000，输出 AO 为 11010001，BO 为 10110001。此时从寄存器 C 中读取数据 11010001 输出到 AO，从寄存器 A 中读取数据 10110001 输出到 BO

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

4. RAM

a) 波形仿真过程（过程详见实验步骤）



b) 波形仿真波形图

c) 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

e) 时序仿真

- SM 和指令计数器 PC 没有 tpd
- 通用寄存器组

a) 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）

b) 时序仿真图

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	10.033 ns	raa[1]	s[7]
2	N/A	None	9.855 ns	raa[0]	s[7]
3	N/A	None	9.852 ns	rwba[1]	d[6]
4	N/A	None	9.828 ns	rwba[0]	d[6]
5	N/A	None	9.590 ns	raa[1]	s[0]
6	N/A	None	9.452 ns	rwba[1]	d[2]
7	N/A	None	9.438 ns	rwba[1]	d[0]
8	N/A	None	9.357 ns	raa[0]	s[3]
9	N/A	None	9.334 ns	rwba[1]	d[4]
10	N/A	None	9.310 ns	rwba[0]	d[4]
11	N/A	None	9.300 ns	raa[0]	s[0]
12	N/A	None	9.288 ns	raa[1]	s[3]
13	N/A	None	9.281 ns	raa[0]	s[1]
14	N/A	None	9.211 ns	raa[1]	s[1]
15	N/A	None	9.176 ns	rwba[0]	d[0]
16	N/A	None	9.104 ns	rwba[0]	d[2]
17	N/A	None	8.810 ns	raa[1]	s[6]
18	N/A	None	8.800 ns	rwba[1]	d[3]
19	N/A	None	8.771 ns	raa[0]	s[2]
20	N/A	None	8.764 ns	rwba[1]	d[1]
21	N/A	None	8.701 ns	raa[1]	s[2]
22	N/A	None	8.659 ns	raa[0]	s[4]
23	N/A	None	8.590 ns	raa[1]	s[4]
24	N/A	None	8.448 ns	rwba[1]	d[7]
25	N/A	None	8.425 ns	rwba[0]	d[7]
26	N/A	None	8.409 ns	rwba[0]	d[1]
27	N/A	None	8.401 ns	rwba[1]	d[5]
28	N/A	None	8.386 ns	raa[0]	s[6]
29	N/A	None	8.378 ns	rwba[0]	d[5]
30	N/A	None	8.324 ns	raa[1]	s[5]
31	N/A	None	8.187 ns	rwba[0]	d[3]
32	N/A	None	7.999 ns	raa[0]	s[5]

c) 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

a) 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）

b) 时序仿真图

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	9.834 ns	XL	dataout[4]
2	N/A	None	9.780 ns	datain[4]	dataout[4]
3	N/A	None	9.763 ns	DL	dataout[4]
4	N/A	None	9.668 ns	XL	dataout[1]
5	N/A	None	9.504 ns	DL	dataout[1]
6	N/A	None	9.493 ns	datain[2]	dataout[2]
7	N/A	None	9.360 ns	datain[1]	dataout[1]
8	N/A	None	9.230 ns	XL	dataout[2]
9	N/A	None	9.159 ns	DL	dataout[2]
10	N/A	None	9.020 ns	datain[0]	dataout[0]
11	N/A	None	8.833 ns	datain[5]	dataout[5]
12	N/A	None	8.725 ns	datain[6]	dataout[6]
13	N/A	None	8.470 ns	XL	dataout[0]
14	N/A	None	8.372 ns	XL	dataout[6]
15	N/A	None	8.307 ns	datain[3]	dataout[3]
16	N/A	None	8.301 ns	DL	dataout[6]
17	N/A	None	8.217 ns	XL	dataout[7]
18	N/A	None	8.212 ns	DL	dataout[0]
19	N/A	None	8.146 ns	DL	dataout[7]
20	N/A	None	8.065 ns	XL	dataout[3]
21	N/A	None	8.016 ns	DL	dataout[5]
22	N/A	None	8.000 ns	XL	dataout[5]
23	N/A	None	7.919 ns	datain[7]	dataout[7]
24	N/A	None	7.909 ns	DL	dataout[3]

c) 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。

2) 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

2、 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。