西安电子科技大学
XIDIAN UNIVERSITY

基于 FPGA 的数字系统设计

实验报告

实验名称：LAB8 计数器

任课教师：沈沛意老师

学号姓名：

提交日期：

一、 实验环境

win11

ISE14.7, Modelsim SE-64 10.4

二、 实验介绍

本次实验将计数器的各个子模块：ALU,MEM,COMP,CNTRL_FSM 等组合在一起，完成计数器的顶层模块 SIMPLE_CALC

三、 实验目标

合并之前已经设计好的子模块;

使用 VHDL 中的 generate 描述语句;

在仿真过程中对输出数据做比较;

验证整个电路功能的正确性;

运行仿真。

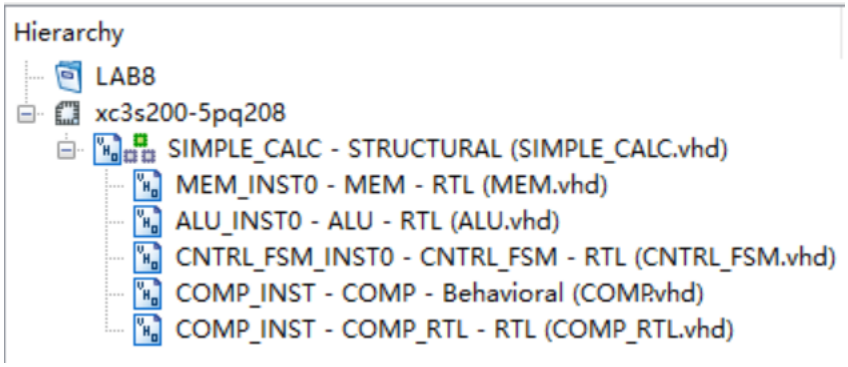
四、 实验过程以及步骤

本实验包含六个主要的部分:创建一个新的 ISE 工程;导入之前写好的工程文件;修改 MEM 模块,更新 ROM 中的值;编写 SIMPLE_CALC 的顶层模块,并检查语法错误;创建测试平台,并对工程进行仿真验证;最后,用 ISE 中的 Place & Route 工具在 FPGA 器件中实现这个计数器,并查看相应的报告文件。

1. 创建一个新工程——LAB8，其属性配置与先前的实验相同
2. 导入源文件：

   使用 Add Copy of Source 导入实验三，五，六，七中分别创建的包集合 CLAC_PACKAGE 和下层模块 MEM、COMP、COMP_RTL、ALU 和

CNTRL_FSM。



3. 修改 MEM 模块

修改 MEM 模块的源文件,更新 ROM 中的数据,将测试数据输入 MEM 模块中。

以下是为 ALU 所有功能设计的简单测试数据, 我将其填入了 MEM 模块相应位置

A_IN	B_IN	OP_CODE	C_IN	EXP_OUT
0001	0001	0000	0	0001(A)
0001	0001	0000	1	0010(A+1)
0001	0001	0001	0	0010(A+B)
0001	0001	0001	1	0011(A+B+1)
0010	0001	0010	0	0000（A+ not B）
0010	0001	0010	1	0001（A-B）
0010	0010	0011	0	0001(A-1)
0010	0001	0011	1	0010(A)
0011	0001	0100	0	0001(与)
0011	0001	0101	0	0011（或）
0101	0110	0110	0	0011（异或）
0101	0110	0111	0	1010（反码）
0101	0110	1000	0	0000（传输 0）
0101	0110	0111	1	X（不合法输出）

4. 编写顶层模块的 VHDL 源代码并做语法检查

编写 SIMPLE_CALC 模块的 VHDL 源代码,并检查语法错误。

(1)选择菜单栏中的 Project→New Source。在 Select Source Type 窗口中,选择左侧的 VHDL Module,在右侧 File Name 栏中填入文件名

SIMPLE_CALC。

(2)单击 Next 按钮,进入 Define Module 窗口。在 Define Module 窗口中输入如表 6.5 所示的输入/输出端口定义,单击 Next 按钮,确认模块信息后,单击 Finish 按钮完成模块的配置。

表 6.5 SIMPLE_CALC 的端口定义

端口名称	位宽	方向
CLK	1 bit	in
RESET	1 bit	in
RESULT	1 bit	out

(3)在打开的新建 VHDL 文件编辑窗口中,声明使用 CALC1_PACKAGE 包集合,并将结构体名称修改为 STRUCTURAL。在结构体内,声明要使用到的元件 MEM、ALU、COMP、COMP_RTL 和 CNTRL_FSM。

(4)声明用于连接各个子模块的内部信号。

用 generate 语句配置选择使用 COMP 或是 COMP_RTL 模块,COMP 是行为描述结构的模块,只用于仿真,而 COMP_RTL 是 RTL 描述结构的模块。在本实验中,我们先用仿真的方法做验证,因此用 generate 语句配置为使用 COMP 模块。

(1)在 SIMPLE_CALC 的实体定义中,用 generic 定义一个 BOOLEAN 类型的参数 SYNTH,SYNTH 参数的默认值为 FALSE。

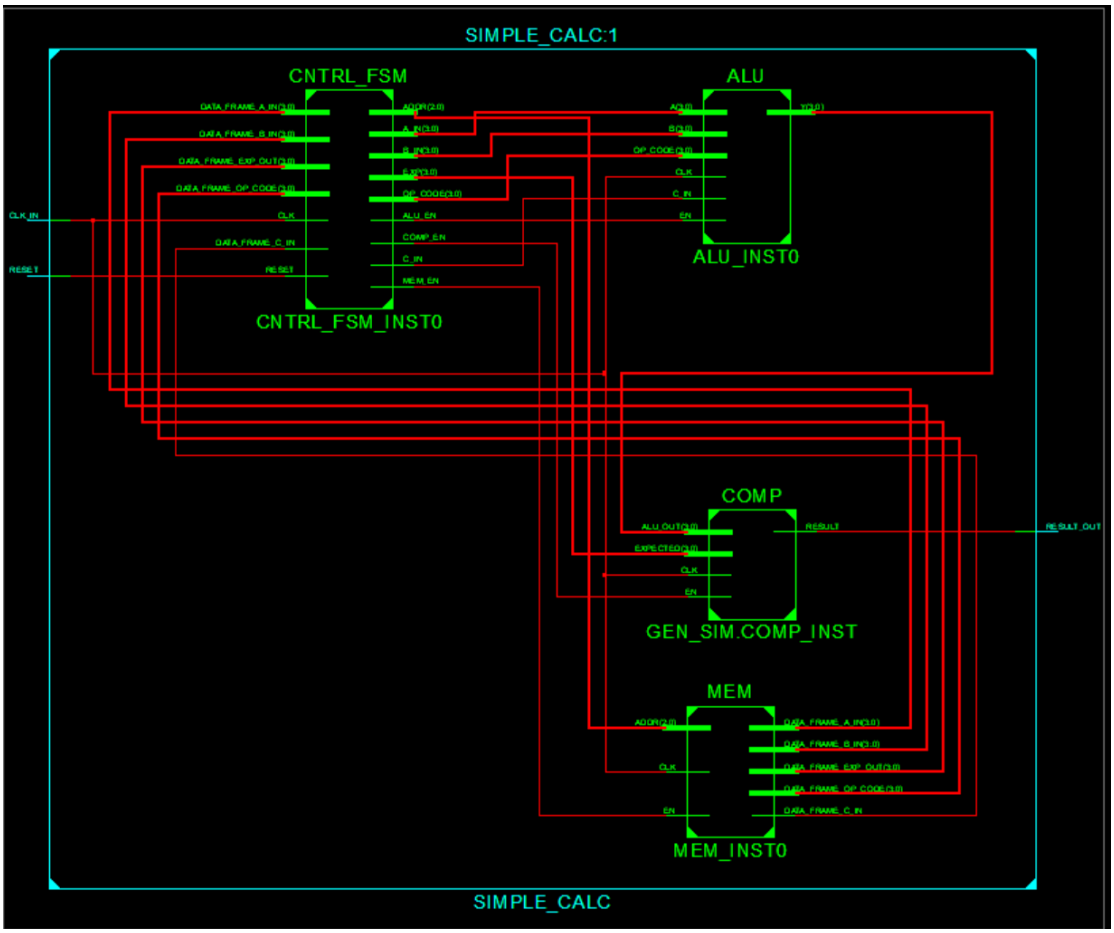
(2)在结构体的关键字 begin 之后,根据 SYNTH 的值创建两个 generate 描述语句。

(3)定义一个 MY_RECORD 类型的信号,用于连接 MEM 子模块和 CNTRL_FSM 子模块。

检查语法错误:

(1)在 Sources 窗口中,选中 SIMPLE_CALC.VHD 模块,在 Processes 窗口中,展开 Synthesize 树,并双击 Check Syntax。

(2)双击 View RTL Schematic,查看综合后的电路与设计目标电路是否相符——相符合。



5. 仿真,验证功能正确性

创建 SIMPLE_CALC 模块的测试平台,并运行仿真,验证功能正确性。

(1) 选择菜单栏中的 Project→New Source。在 Select Source Type 窗口中,选择左侧的 VHDL Test Bench,在右侧 File Name 栏中填入文件名 SIMPLE_CALC_TB。单击 Next 按钮,软件弹出一个界面,询问测试平台文件与哪个设计文件关联,选择 SIMPLE_CALC。

(2) 在测试平台文件编辑窗口中,设置 CLK 的时钟频率为 50MHz。对

测试平台文件进行仿真，在仿真过程中,将需要查看的子模块中的信号拖入仿真波形窗口。

6. 6. 在 FPGA 中实现工程设计,并查看报告文件

使用 ISE 中的 Place& Route 工具实现工程设计,查看 MAP 和 PAR 的报告文件。

双击 Implement Design 后，获得 Map Report，根据其内容填写有表格:

Resource	Utilization
Number of slice filp-flops	1%
Number of four-input LUTs	1%
Number of bonded IOBs	2%

再展开 Generate Post-Place & Route Static Timing,双击 Analyze Post Place and Route Static Timing 以获得时序报告，从而可以得到以下内容

Worst-case OFFSET OUT（时钟上升沿到输出端口的最差延迟）:

8.611 ns

Clock to Setup on CLK_IN: 4.637 ns

Max operation frequency = 1000 / (建立时间 + 时钟到输出延迟)

= 1000 / (4.637 + 8.611) ≈ 75.45 MHz

五、 实验总结

在本实验中,我们完成了计数器工程的全部内容,学会了使用 generate 描述语句配置两种不同描述方式的模块。在实验过程中，我为了测试 ALU 的每个种功能，向 MEM 的 ROM 添加了一共 14 条测试记录。这个计数器工程已经是一个完整的工程了,下一步我们只需要将这个工程下载到 FPGA 器件中实际运行,并测试硬件系统电路即可。

六、 实验代码

SIMPLE_CALC.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use WORK.CALC1_PACKAGE.ALL;
--顶层设计实体
entity SIMPLE_CALC is
    generic(SYNTH:BOOLEAN := FALSE);--通用参数：选择仿真或综合模式
    Port ( CLK_IN : in  STD_LOGIC;--全局时钟
           RESET : in  STD_LOGIC;--全局复位信号
           RESULT_OUT : out  STD_LOGIC);--最终测试结果输出（1=通过，0=失败）
end SIMPLE_CALC;
--结构体描述：系统级组件互联
architecture STRUCTURAL of SIMPLE_CALC is
--组件声明：控制状态机
component CNTRL_FSM
  port(
       DATA_FRAME: in MY_RECORD;--输入：来自 MEM 的测试数据帧
        CLK: in STD_LOGIC;  --时钟
        RESET: in STD_LOGIC;--复位
        A_IN: out STD_LOGIC_VECTOR(3 downto 0);--输出：ALU 操作数 A
        B_IN: out STD_LOGIC_VECTOR(3 downto 0);--输出：ALU 操作数 B
       C_IN: out STD_LOGIC; --输出：进位输入
       OP_CODE: out STD_LOGIC_VECTOR(3 downto 0);--操作码
        EXP: out STD_LOGIC_VECTOR(3 downto 0);--预期结果
        ADDR: out STD_LOGIC_VECTOR(3 downto 0);--输出，MEM 地址：为了配合 MEM 中 14 条测试记录，将 ADDR 从三位变成四位，使得每一条记录得以测试
       --使能信号
        COMP_EN: out STD_LOGIC;
        MEM_EN: out STD_LOGIC;
        ALU_EN: out STD_LOGIC);
end component;

--ALU 组件声明，执行具体运算
component ALU
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);-- 操作数 A
           B : in  STD_LOGIC_VECTOR (3 downto 0);-- 操作数 B
           C_IN : in  STD_LOGIC;-- 进位
```

```
            OP_CODE : in  STD_LOGIC_VECTOR (3 downto 0);--操作码
            CLK : in  STD_LOGIC;
            EN : in  STD_LOGIC;
            Y : out  STD_LOGIC_VECTOR (3 downto 0));--输出：运算结
果
    end component;
    --MEM 组件声明
    component MEM
        PORT(CLK,EN:in STD_LOGIC;
             ADDR: in STD_LOGIC_VECTOR(3 downto 0):="0000";--地址 4 位
             DATA_FRAME: out MY_RECORD);  --输出：测试用例数据帧
    end component;
    --仿真用比较器
    component COMP
        Port ( CLK : in  STD_LOGIC;
               EN : in  STD_LOGIC;
            EXPECTED : in  STD_LOGIC_VECTOR (3 downto 0);--输入：预
期值
            ALU_OUT : in  STD_LOGIC_VECTOR (3 downto 0);--输入：ALU
输出
            RESULT : out  STD_LOGIC);--比较结果
    end component;
    --综合用比较器（与上面的类似）
    component COMP_RTL
        Port ( CLK : in  STD_LOGIC;
               EN : in  STD_LOGIC;
            EXPECTED : in  STD_LOGIC_VECTOR (3 downto 0);
            ALU_OUT : in  STD_LOGIC_VECTOR (3 downto 0);
            RESULT : out  STD_LOGIC);
    end component;
    --内部信号定义（用于组件间连接）
    signal EXP_OUT_SIG : STD_LOGIC_VECTOR(3 downto 0);--预期结果信号
    signal A_IN_SIG : STD_LOGIC_VECTOR(3 downto 0);--ALU 输入 A
    signal B_IN_SIG : STD_LOGIC_VECTOR(3 downto 0); --ALU 输入 B
    signal OP_CODE_SIG : STD_LOGIC_VECTOR(3 downto 0);--操作码
    signal CI_SIG : STD_LOGIC;--进位输入
    signal ALU_EN_SIG : STD_LOGIC;--ALU 使能
    signal COMP_EN_SIG : STD_LOGIC;--COMP 使能
    signal MEM_EN_SIG : STD_LOGIC;--MEM 使能
    signal ALU_OUT_SIG : STD_LOGIC_VECTOR(3 downto 0);ALU 输出
    signal ADDR_SIG : STD_LOGIC_VECTOR(3 downto 0);-- 地址修改为 4 位
    signal DATA_FRAME_SIG : MY_RECORD;--数据帧（来自 MEM）

    begin  --组件实例化与互连
```

```
    -- 存储器实例：存储 14 条测试用例
      MEM_INST0: MEM port map(
        ADDR =>ADDR_SIG, -- 地址输入（4 位）
        DATA_FRAME => DATA_FRAME_SIG, -- 输出数据帧
        EN => MEM_EN_SIG,
        CLK=>CLK_IN);
      -- ALU 实例：执行运算操作
      ALU_INST0: ALU port map (
        OP_CODE=>OP_CODE_SIG,
        A=>A_IN_SIG,
        B=> B_IN_SIG,
        C_IN => CI_SIG,
        Y=>ALU_OUT_SIG,
        EN =>ALU_EN_SIG,
        CLK=>CLK_IN);
         -- 控制状态机实例：协调数据流
      CNTRL_FSM_INST0: CNTRL_FSM port map (
        ADDR =>ADDR_SIG, -- 输出地址到 MEM
        DATA_FRAME => DATA_FRAME_SIG, -- 从 MEM 读取数据帧
        OP_CODE=>OP_CODE_SIG,
        A_IN =>A_IN_SIG,
        B_IN => B_IN_SIG,
        C_IN=> CI_SIG,--以上四条：输出到 ALU 的各项内容
        EXP=>EXP_OUT_SIG, -- 输出预期结果到比较器
        CLK =>CLK_IN,
        RESET => RESET,
        ALU_EN =>ALU_EN_SIG,--以下位控制 ALU，COMP，MEM 的使能信号
        COMP_EN =>COMP_EN_SIG,
        MEM_EN => MEM_EN_SIG);
         -- 生成语句：根据 SYNTH 选择比较器类型
      GEN_SIM : if SYNTH = FALSE generate-- 仿真模式使用行为级比较器
        COMP_INST: component COMP port map (CLK => CLK_IN, EN =>
COMP_EN_SIG,
        EXPECTED =>EXP_OUT_SIG, ALU_OUT=>ALU_OUT_SIG,
        RESULT => RESULT_OUT);
      end generate GEN_SIM ;

      GEN_SYNTH : if SYNTH = TRUE generate-- 综合模式使用 RTL 级比较器
        COMP_INST: component COMP_RTL port map (CLK =>CLK_IN, EN =>
COMP_EN_SIG,
        EXPECTED =>EXP_OUT_SIG, ALU_OUT=>ALU_OUT_SIG,
        RESULT => RESULT_OUT );
      end generate GEN_SYNTH;
    end STRUCTURAL;
```

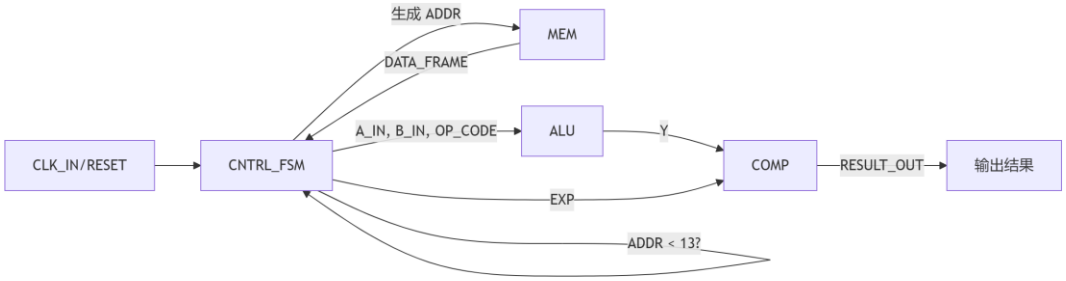
SIMPLE_CALC_TB.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;


ENTITY SIMPLE_CALC_TB IS
END SIMPLE_CALC_TB;

ARCHITECTURE TEST OF SIMPLE_CALC_TB IS

    -- 声明被测单元（Unit Under Test, UUT）的组件

    COMPONENT SIMPLE_CALC
    PORT(
         CLK_IN : IN  std_logic;--全局时钟
         RESET : IN  std_logic;--复位信号
         RESULT_OUT : OUT  std_logic—测试结果（1/0，成功/失败）
        );
    END COMPONENT;


   --Inputs
   signal CLK_TB : std_logic := '0';--测试平台时钟信号（初始值为 0）
   signal RESET_TB : std_logic := '0'; --测试平台复位信号（初始值
为 0）

    --Outputs
   signal RESULT : std_logic; --接收 UUT 的测试结果

BEGIN

    -- 实例化被测单元 (UUT)
   uut: SIMPLE_CALC PORT MAP (
          CLK_IN => CLK_TB, -- 时钟信号连接到测试平台时钟
          RESET => RESET_TB, -- 复位信号连接到测试平台复位
          RESULT_OUT => RESULT-- 结果输出连接到 RESULT 信号
        );
-- 生成 50MHz 时钟信号（周期 20ns，每 10ns 翻转一次）
   CLK_TB <= not CLK_TB after 10 ns;
-- 原始复位信号（已注释，因会导致测试中断）
   --RESET_TB <= '1' after 5 ns, '0' after 10 ns, '1' after 700 ns,
```

'0' after 725 ns;

        -- 修改后的复位信号（仿真时间延长到 2000ns）
        -- 关键修改说明：
        -- 1. 复位仅在仿真开始时触发一次（5ns 置高，20ns 置低）
        -- 2. 移除第二次复位（原 700ns 复位会中断测试流程）
        -- 3. 确保足够仿真时间覆盖 14 条测试用例（约需 1400ns）
      RESET_TB <= '1' after 5 ns, '0' after 20 ns;
    end architecture TEST;

流程图



MEM 模块修改内容（根据设计用例）:

```
    constant MY_ROM:ROM_ARRAY :=
    (
      -- OP_CODE=0000 (A / A+1)
      0 => (A_IN => "0001", B_IN => "0001", OP_CODE => "0000", C_IN => '0', EXP_OUT => "0001"),
-- A (C_IN=0)
      1 => (A_IN => "0001", B_IN => "0001", OP_CODE => "0000", C_IN => '1', EXP_OUT => "0010"),
-- A+1 (C_IN=1)

      -- OP_CODE=0001 (A+B / A+B+1)
      2 => (A_IN => "0001", B_IN => "0001", OP_CODE => "0001", C_IN => '0', EXP_OUT => "0010"),
-- A+B (C_IN=0)
      3 => (A_IN => "0001", B_IN => "0001", OP_CODE => "0001", C_IN => '1', EXP_OUT => "0011"),
-- A+B+1 (C_IN=1)

      -- OP_CODE=0010 (A+?B / A-B)
      4 => (A_IN => "0010", B_IN => "0001", OP_CODE => "0010", C_IN => '0', EXP_OUT => "0000"),
-- A+?B (C_IN=0)
      5 => (A_IN => "0010", B_IN => "0001", OP_CODE => "0010", C_IN => '1', EXP_OUT => "0001"),
-- A-B (C_IN=1)

      -- OP_CODE=0011 (A-1 / A)
      6 => (A_IN => "0010", B_IN => "0010", OP_CODE => "0011", C_IN => '0', EXP_OUT => "0001"),
```

```
-- A-1 (C_IN=0)
      7 => (A_IN => "0010", B_IN => "0001", OP_CODE => "0011", C_IN => '1', EXP_OUT => "0010"),
-- A (C_IN=1)

      -- OP_CODE=0100 (AND)
      8 => (A_IN => "0011", B_IN => "0001", OP_CODE => "0100", C_IN => '0', EXP_OUT => "0001"),
-- A AND B (C_IN=0)

      -- OP_CODE=0101 (OR)
      9 => (A_IN => "0011", B_IN => "0001", OP_CODE => "0101", C_IN => '0', EXP_OUT => "0011"),
-- A OR B (C_IN=0)

      -- OP_CODE=0110 (XOR)
      10 => (A_IN => "0101", B_IN => "0110", OP_CODE => "0110", C_IN => '0', EXP_OUT =>
"0011"), -- A XOR B (C_IN=0)

      -- OP_CODE=0111 (?B / 非法)
      11 => (A_IN => "0101", B_IN => "0110", OP_CODE => "0111", C_IN => '0', EXP_OUT =>
"1010"), -- ?B (C_IN=0)
      12 => (A_IN => "0101", B_IN => "0110", OP_CODE => "0111", C_IN => '1', EXP_OUT =>
"XXXX"), -- 非法 (C_IN=1)

      -- OP_CODE=1000 (传输 0)
      13 => (A_IN => "0101", B_IN => "0110", OP_CODE => "1000", C_IN => '0', EXP_OUT =>
"0000") -- 强制输出 0 (C_IN=0)
    );
```

CNTRL_FSM 修改内容

```
-- 修改：地址递增到 13（二进制 1101）后停止
          when S4_DONE=>
              --if ADDR_Q>="101" THEN
                 if ADDR_Q = "1101" then  -- 13（十进制）
                 NEXT_STATE<=S4_DONE;
               else
                 NEXT_STATE<=S1_FETCH;
                   ADDR_I<=ADDR_Q+1;
               end if;
               MEM_EN<='0';
               ALU_EN<='0';
               COMP_EN<='0';
```

七、 实验中的问题

为了测试 ALU 中每一种功能是否可以成功执行，我在修改 MEM 的 ROM 内

容时一共添加了 14 条记录，但由于控制模块 CNTRL_FSM 的地址生成逻辑没有修改成可以递增到 13 的形式，复位逻辑或状态机转移条件错误，所以导致即使 MEM 里面含有 0~13 的测试记录，但是最后仿真时仍然只有 0~5 的测试记录被使用，地址计数器无法递增到最大值 13 即 1101，最后我通过修改 CNTRL_FSM 的状态转移逻辑和地址生成逻辑，把其他需要用到 ADDR 信号的模块中 ADDR 的位数改成了 4 为（3 downto 0），防止地址被截断，最终仿真结果成功跑通了 14 条测试用例。

可以看到，预期输出结果和 ALU 输出结果一直——result 为 1