计数器.实验报告

2018011324

尤艺霖

计82

1.实验要求/目的

1.1.实验目的

掌握时序逻辑电路的基本分析和设计方法

理解同步时序电路和异步时序电路的区别

掌握计数器电路设计原理，用硬件描述语言实现指定功能的计数器设计

学会利用软件仿真实现对数字电路的逻辑功能进行验证和分析

1.2.实验要求

用硬件描述语言实现从00到59的计数器，并通过未经译码处理的数码管进行显示。手动单次进行时钟计数，计数到59后从00开始重新计数，并设置一个复位按键，使计数恢复到00的状态。

使用时钟信号，将计数器改成秒表，并在秒表中使用开关控制启动暂停。

2.思路

使用元件例化的方法，先实现单个触发器，再通过单个触发器实现四位计数器，并用两个四位计数器分别控制两个十进制位，并单独用一个实体控制进位。显示到数码管上的工作则由另一个实体完成。用了一个输入来控制是秒表状态还是计数器状态，同时引脚数量也正好够在Jielab上运行。

3.代码

3.1.单个触发器

按照触发器的逻辑写出的，q和nq是两个输出，同时具有d、clk、rst三个输入。

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

entity ff1 is
port(
    clk,d,rst:in std_logic;
    q,nq:out std_logic
);
end ff1;

architecture ff of ff1 is
begin
    process(clk,rst) --触发器，通过时钟上升沿控制
```

```
    begin
        if(rst='1') then
            q<='0';
            nq<='1';
        elsif(rising_edge(clk)) then
            q<=d;
            nq<=not d;
        end if;
    end process;
end ff;
```

3.2.四位计数器

参考了教材上的四位计数器结构，由四个触发器组成。

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

entity ff4 is
port(
    clk,rst:in std_logic;
    output:out std_logic_vector(3 downto 0)
);
end ff4;

architecture ff of ff4 is
    component ff1
    port(
        clk,d,rst:in std_logic;
        q,nq:out std_logic
    );
    end component;
    signal mid: std_logic_vector(3 downto 0); --暂时存放中间变量
begin --四个触发器组成四位计数器
    f1:ff1 port map(clk=>clk,rst=>rst,d=>mid(0),q=>output(0),nq=>mid(0));
    f2:ff1 port map(clk=>mid(0),rst=>rst,d=>mid(1),q=>output(1),nq=>mid(1));
    f3:ff1 port map(clk=>mid(1),rst=>rst,d=>mid(2),q=>output(2),nq=>mid(2));
    f4:ff1 port map(clk=>mid(2),rst=>rst,d=>mid(3),q=>output(3),nq=>mid(3));
end ff;
```

3.3.译码器

直接复制了一段“点亮数字人生”实验中写过的代码段

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

entity display7 is
port(
```

```
    input:in std_logic_vector(3 downto 0);
    output:out std_logic_vector(6 downto 0)
);
end display7;

architecture display of display7 is
begin
    process(input)
    begin
        case input is
            when "0000"=>output<="1111110";
            when "0001"=>output<="0110000";
            when "0010"=>output<="1101101";
            when "0011"=>output<="1111001";
            when "0100"=>output<="0110011";
            when "0101"=>output<="1011011";
            when "0110"=>output<="1011111";
            when "0111"=>output<="1110000";
            when "1000"=>output<="1111111";
            when "1001"=>output<="1110011";
            when "1010"=>output<="1110111";
            when "1011"=>output<="0011111";
            when "1100"=>output<="1001110";
            when "1101"=>output<="0111101";
            when "1110"=>output<="1001111";
            when "1111"=>output<="1000111";
            when others=>output<="0000000";
        end case;
    end process;
end display;
```

3.4.计数器控制器

写这一段花费了一些时间，主要在控制进位和重置的一段，需要配合触发器的写法，及时让reset系列变量进入低电平，同时也需要满足硬件描述语言的语法要求。具体实现已经注释好了

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

entity counter is
port(
    highoutput,lowoutput:out std_logic_vector(6 downto 0);
    clk,rst,is_cnt,pause:in std_logic
);
end counter;

architecture count of counter is
    component display7
    port(
        input:in std_logic_vector(3 downto 0);
        output:out std_logic_vector(6 downto 0)
```

```
    );
    end component;
    component ff4
    port(
        clk,rst:in std_logic;
        output:out std_logic_vector(3 downto 0)
    );
    end component;
    signal highmid1,lowmid1:std_logic_vector(3 downto 0);
    signal highmid2,lowmid2:std_logic_vector(3 downto 0);
    signal cnt:integer:=0;
    signal highclock,lowclock,rstlow,rsthigh:std_logic;
begin
    f4:ff4 port map(clk=>lowclock,rst=>rstlow,output=>lowmid1);--low
    f5:ff4 port map(clk=>highclock,rst=>rsthigh,output=>highmid1);--high
    d1:display7 port map(input=>highmid2,output=>highoutput);
    d2:display7 port map(input=>lowmid2,output=>lowoutput);
    process(lowmid1,highmid1,rst)
    begin
        if(rst='1') then -- reset 操作优先级最高
            rstlow<='1';
            rsthigh<='1';
        elsif(lowmid1="1010") then -- 低位进位
            highclock<='1';
            rstlow<='1';
        elsif(highmid1="0110") then -- 恢复到00
            rsthigh<='1';
        else -- 上面两种情况运行过之后一定会再经过一遍这种情况，从而解除reset
            highmid2<=highmid1;
            lowmid2<=lowmid1;
            highclock<='0';
            rsthigh<='0';
            rstlow<='0';
        end if;
    end process;
    process(clk)
    begin
        if(pause='0') then -- pause控制全局暂停
            if(is_cnt='1') then -- is_cnt为1时是计数器，为0时是秒表
                lowclock<=clk;
            elsif(rising_edge(clk)) then
                if(cnt<1000000) then -- 做秒表时每1M个时钟上升沿加一次
                    cnt<=cnt+1;
                    lowclock<='0';
                else
                    cnt<=0;
                    lowclock<='1';
                end if;
            end if;
        end if;
    end process;
end count;
```

3.5.TestBench

TestBench框架由教程中给出的网址生成，测试内容为自己填充。

由于电脑配置不好，本地测秒表功能的话仿真时间过长，故仿真只测了计数器功能。

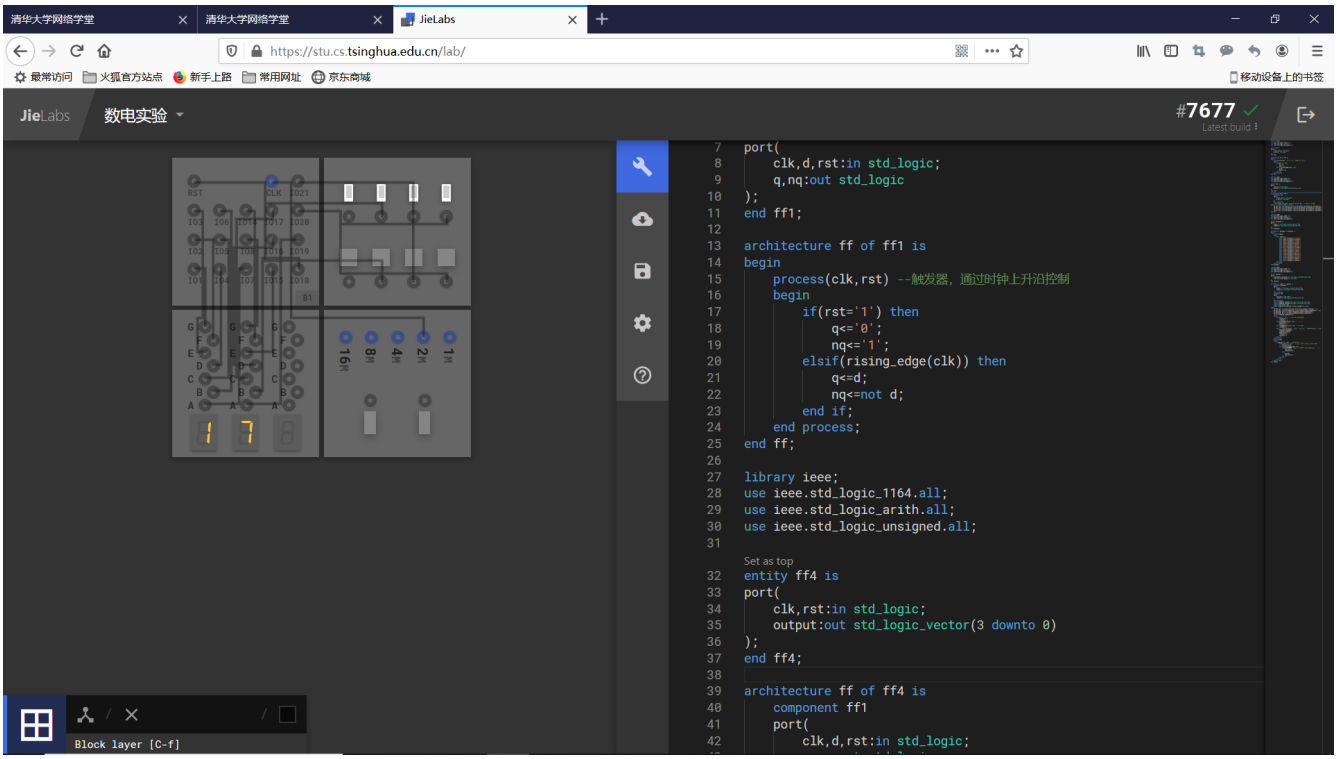
```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity counter_tb is
end;

architecture bench of counter_tb is

  component counter
  port(
    highoutput,lowoutput:out std_logic_vector(6 downto 0);
    clk,rst,is_cnt,pause:in std_logic
  );
  end component;

  signal highoutput,lowoutput: std_logic_vector(6 downto 0);
  signal clk,rst,is_cnt,pause: std_logic ;

  constant clock_period: time := 50 ns;
  signal stop_the_clock: boolean;

begin

  uut: counter port map ( highoutput => highoutput,
                          lowoutput  => lowoutput,
                          clk        => clk,
                          rst        => rst,
                          is_cnt     => is_cnt,
                          pause      => pause );

  stimulus: process
  begin

    -- Put initialisation code here
     rst<='1';
     wait for 100ns;
     rst<='0';
     is_cnt<='1';
     pause<='0';
     wait for 50000ns;
     rst<='1';
     wait for 100ns; -- 测试reset功能
     rst<='0';
     wait for 50000ns;


    -- Put test bench stimulus code here
```

```
    stop_the_clock <= true;
    wait;
  end process;

  clocking: process
  begin
    while not stop_the_clock loop
      clk <= '0', '1' after clock_period / 2;
      wait for clock_period;
    end loop;
    wait;
  end process;

end;
```

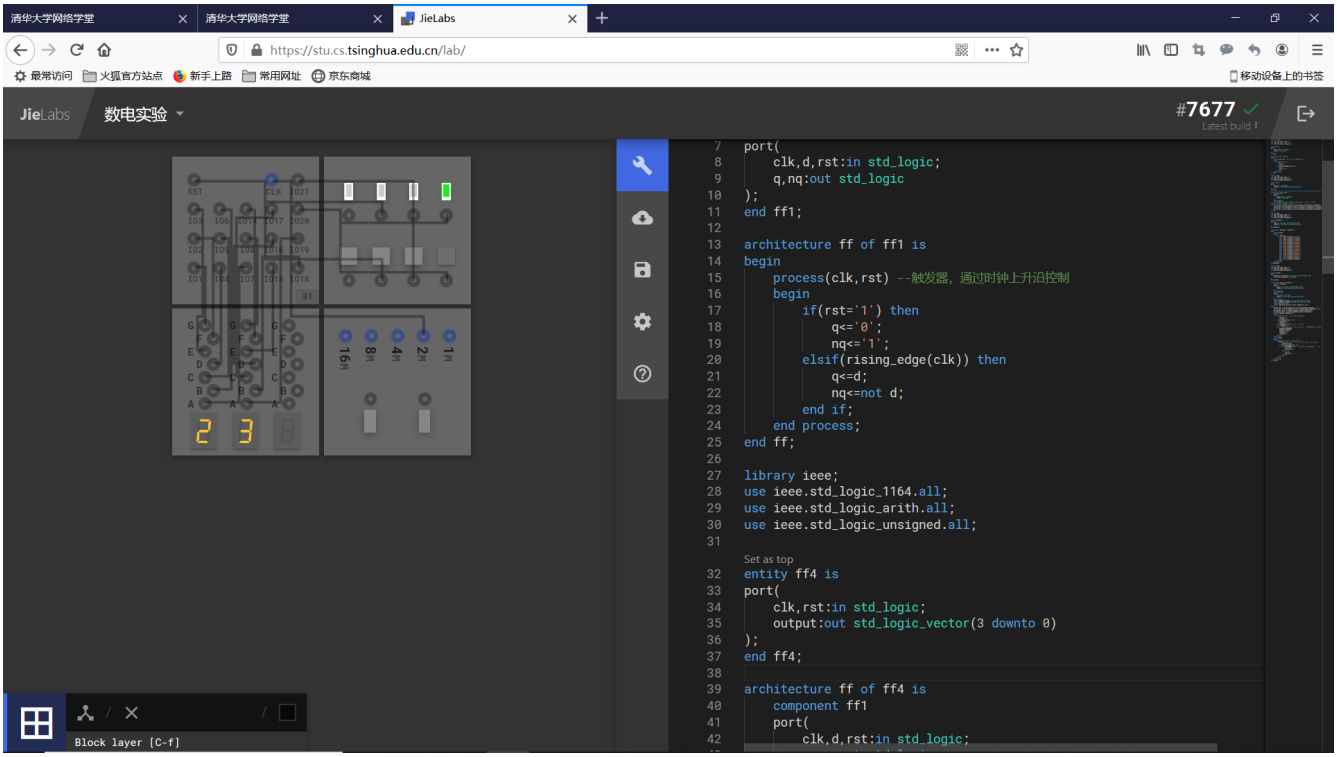
4.实验结果

4.1.Jielab实验结果

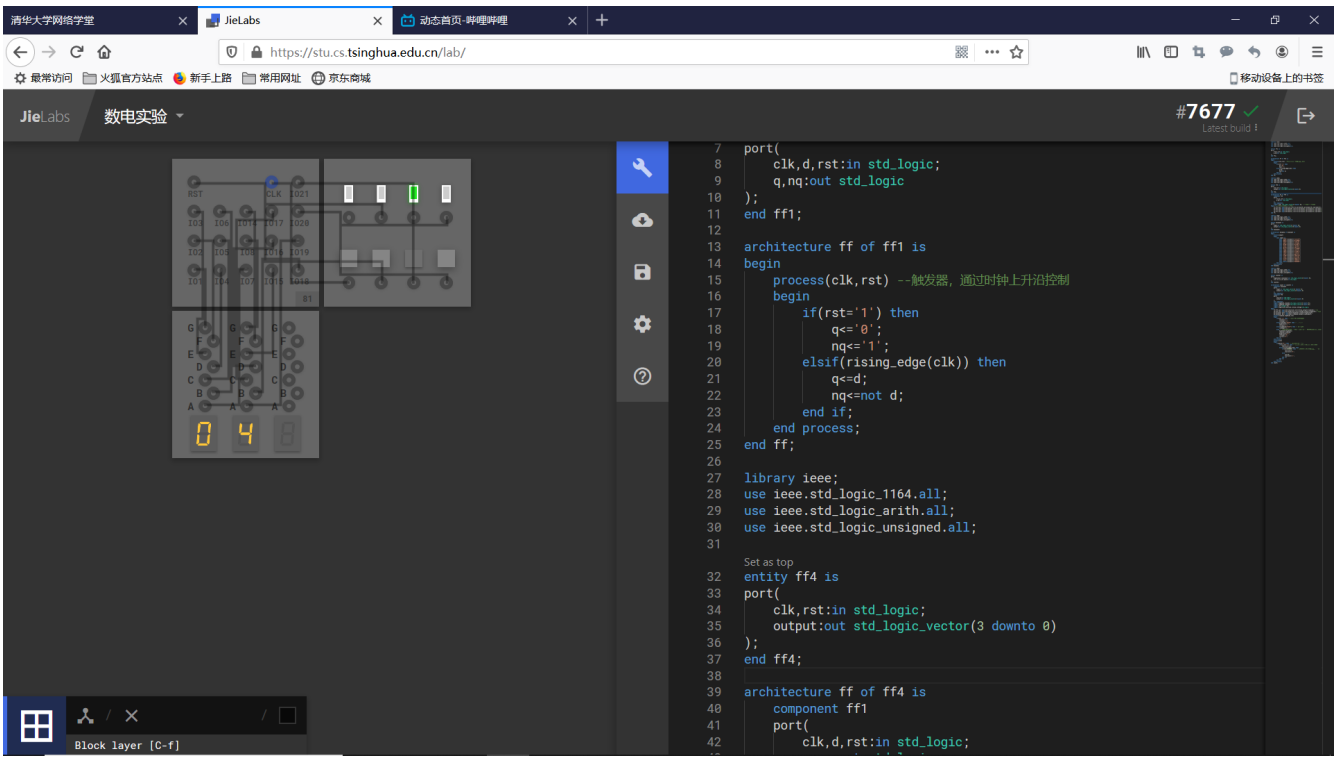
在Jielab上本程序在合理的接线下可以完成所需求的功能。截屏如下：



这是用作秒表时的截屏，为了更快看到进位和恢复，使用了2M频率，使用1M时则是正常秒表。



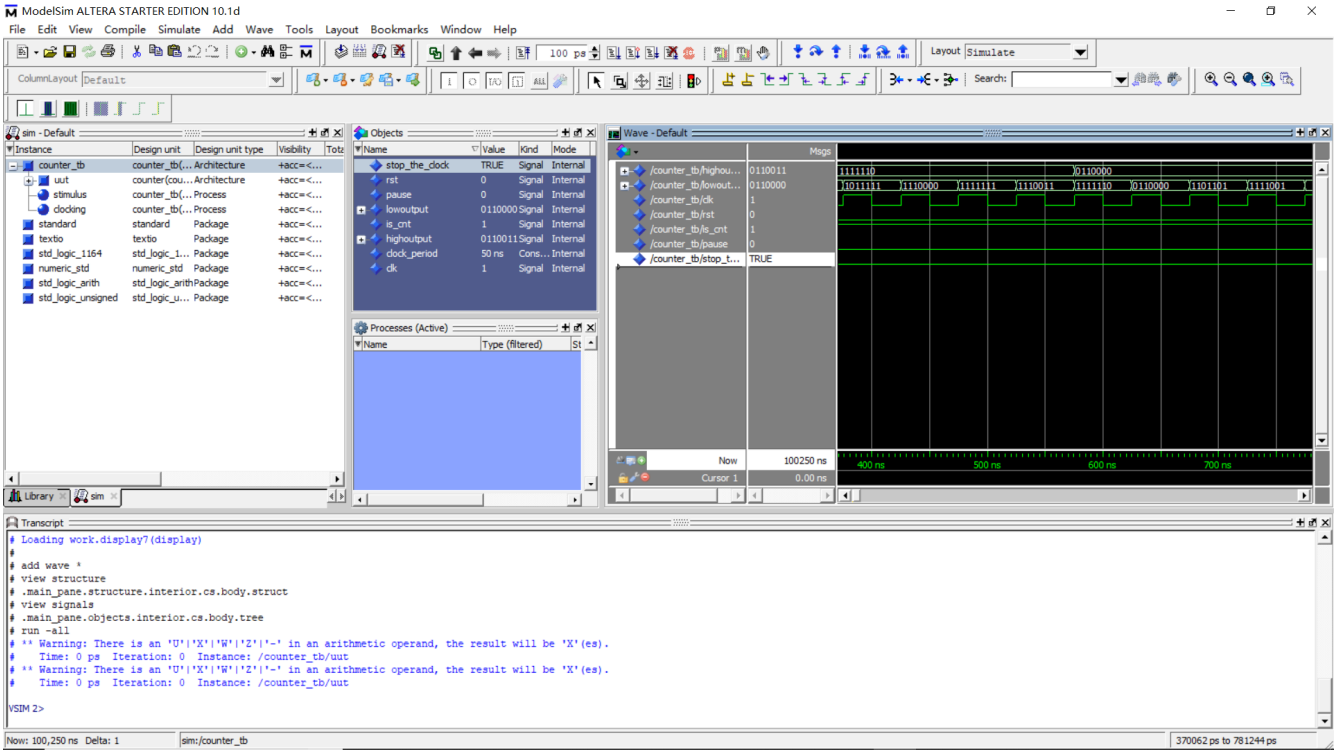
按下暂停可以让计时停住



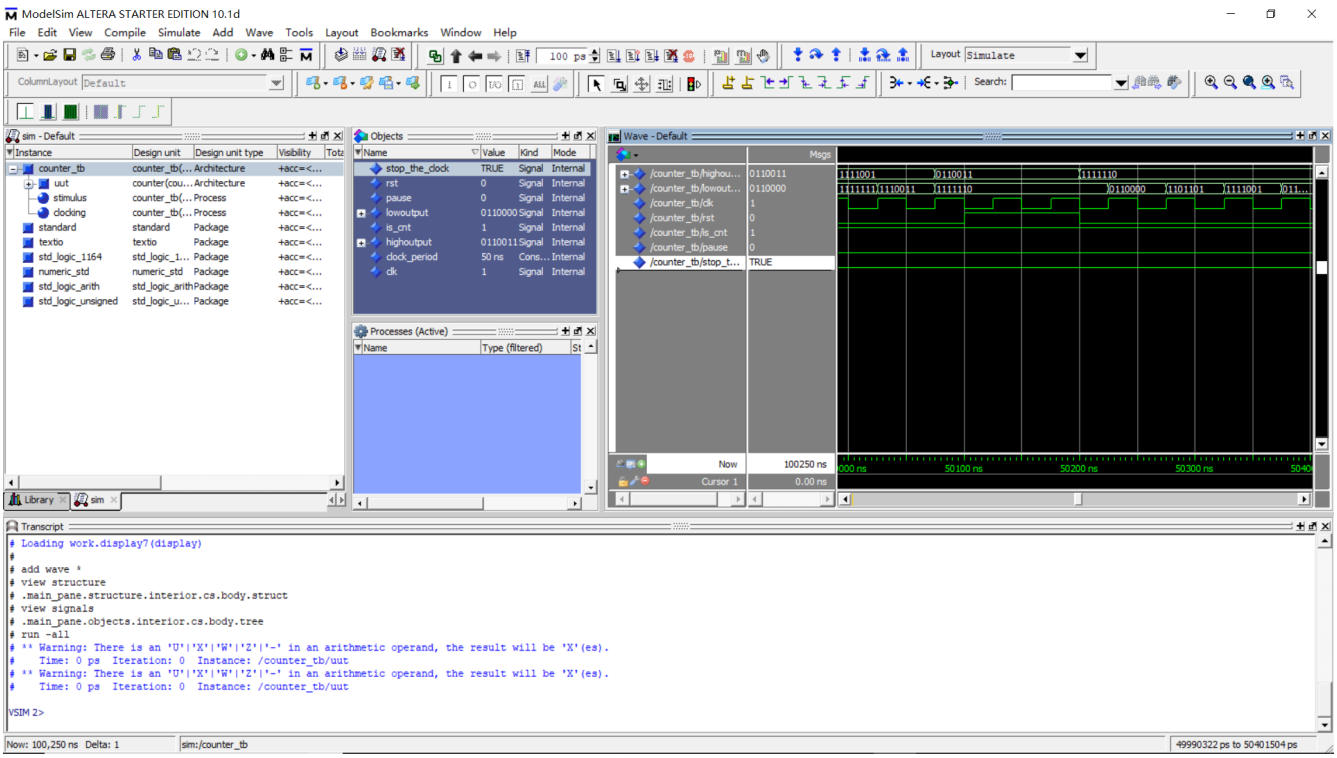
这是用作计数器时的截屏，左起第一个按钮负责原先的时钟信号。

4.2.本地仿真结果

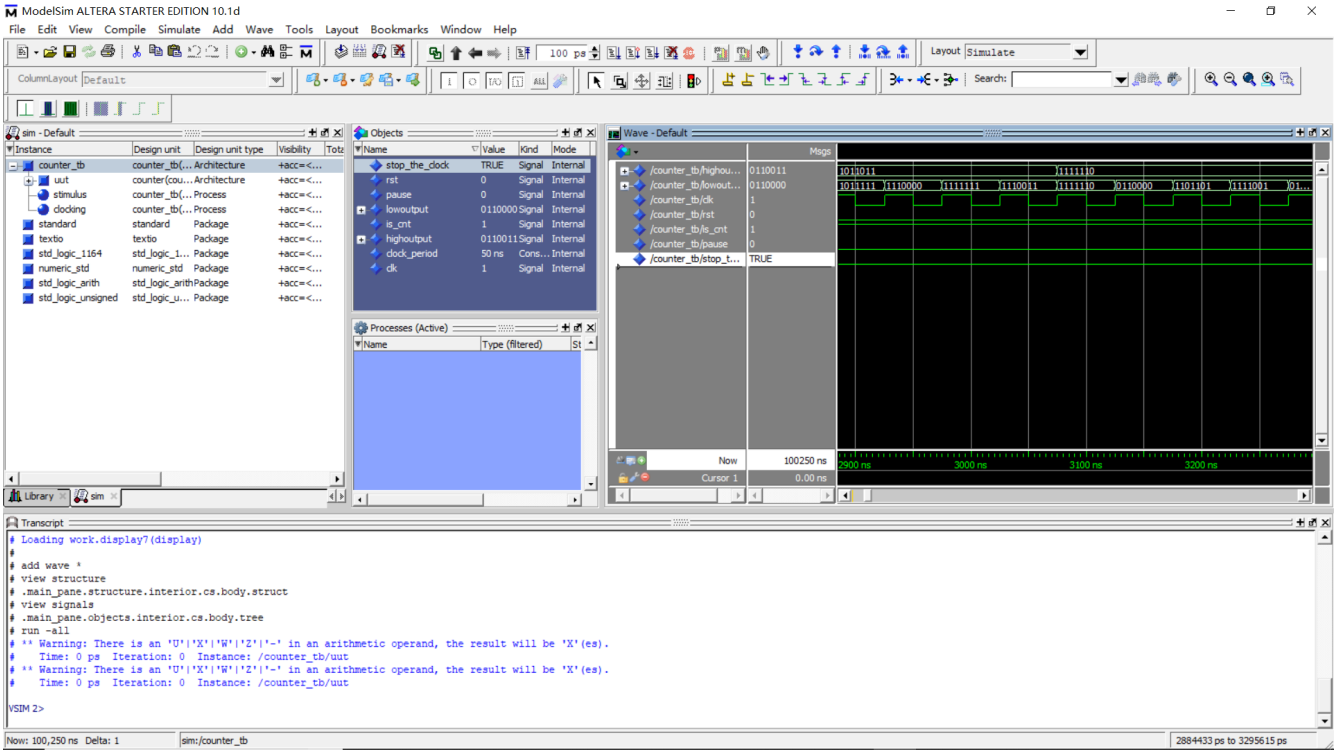
本地仿真主要测试进位、恢复、reset等功能，截图如下：



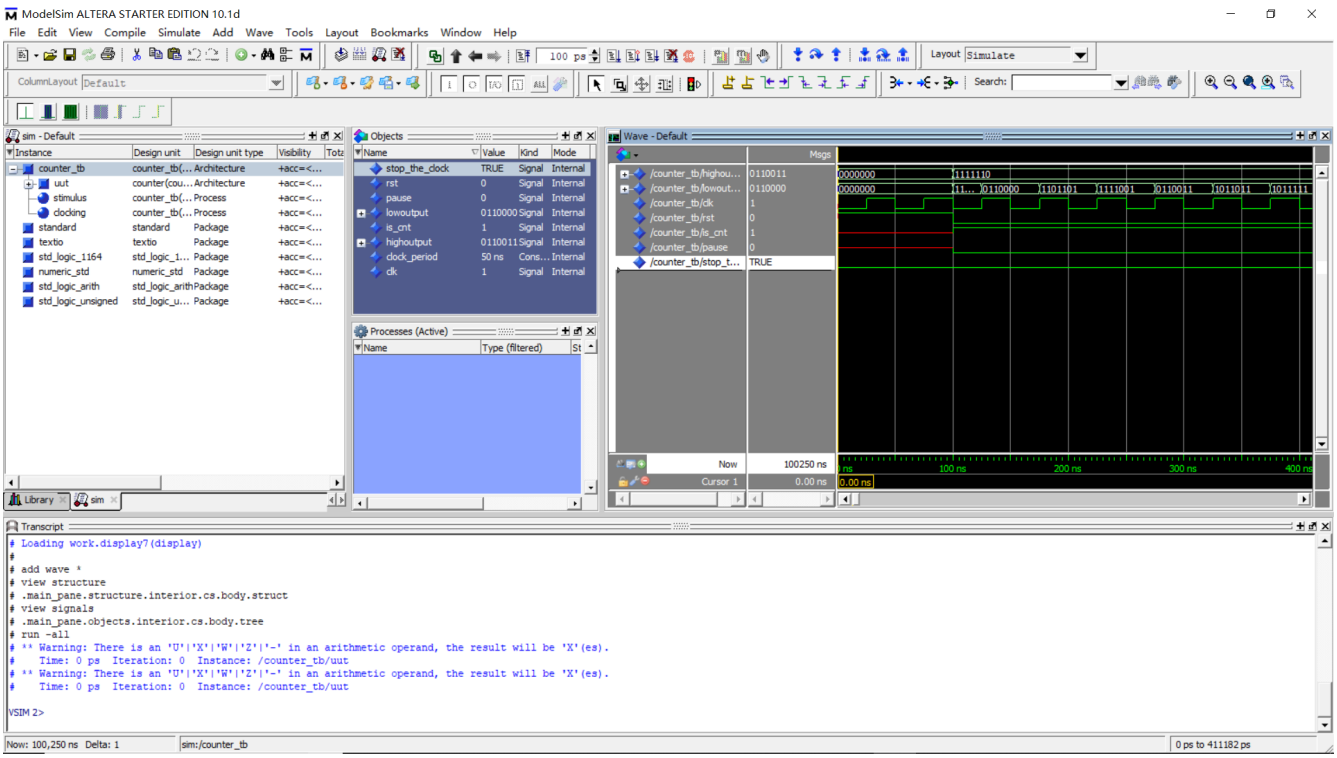
这张截图说明在进位时可以正常工作。



这张截图是按下reset按钮时的反应，说明可以正常使用reset功能。



这张截图是进到59时恢复到00的情况。



这张截图是最开始的情况。

5.实验总结

在这次实验中，我使用元件例化的方式成功完成了一个同时具备秒表功能和计数器功能的电路，支持从00计数到59，然后自动恢复到00。支持一键重置，并允许在秒表功能下暂停计时。通过这一实验，我更深入的理解了触发器的工作原理，学习了使用触发器构成四位计数器，在构造进位控制逻辑时加深了对硬件语言的理解。

在这次实验中，遇到的主要麻烦还是在设计上，通过编译之后很快就通过了测试。日后还需要在逻辑和设计上多加练习。