

井冈山大学  
JINGGANGSHAN UNIVERSITY



## EDA 课程设计报告

课程: EDA 技术实用教程

学院: 电子与信息工程学院

班级:

姓名:

学号:

教师:

完成日期: 2013. 01. 02

原创力文档  
max.book118.com  
预览与源文档一致, 下载高清无水印

## 目录

|                                  |    |
|----------------------------------|----|
| 实验一、3-8 译码器的仿真 .....             | 5  |
| 实验二、2 选一多路选择器.....               | 8  |
| 实验三、十进制计数器.....                  | 10 |
| 实验四、四选一多路选择器.....                | 14 |
| 实验五、ADC0809 采样状态机 .....          | 20 |
| 实验六、11010011 序列检测.....           | 23 |
| 实验七、两个 8 位乘 8 位的有符号数乘法器.....     | 25 |
| 实验八、全加器.....                     | 27 |
| 实验九、LPM_COUNTER 计数模块 .....       | 29 |
| 实验十、LPM_COUNTER 计数模块例化 .....     | 31 |
| 实验十一、LPM 随机存储器的设置和调用 .....       | 33 |
| 实验十二、LPM_ROM 的定制和使 .....         | 36 |
| 实验十三、FIFO 定制.....                | 38 |
| 实验十四、LPM 嵌入式锁相环调用 .....          | 39 |
| 实验十五、NCO 核数控振荡器使用方法 .....        | 40 |
| 实验十六、使用 IP CORE 设计 FIR 滤波器 ..... | 42 |
| 实验十七、数字时钟.....                   | 43 |
| 实验十八、交通灯.....                    | 47 |



## 实验一、3-8 译码器的仿真

一：实验名称：3-8 译码器仿真

二：实验要求：熟悉对 max+plus II 10.0 的使用，并且能简单的使用进行 3-8 译码器的仿真和论证。

三：实验步骤：

1：使用 max+plus II 10.0 软件，设计 3-8 译码器的实验原理图如下所示：

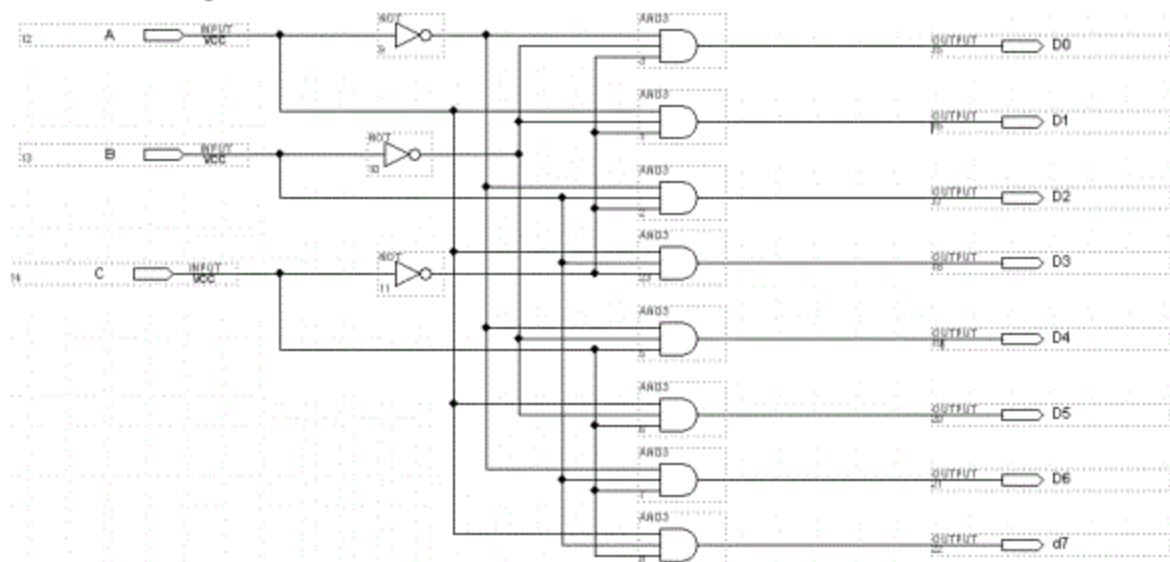


图 1 实验原理图

2：波形的仿真与分析

启动 max+plus II 10.0\Waveform editor 菜单，进入波形编辑窗口，选择欲仿真的所有 I/O 管脚。如下图所示：

原创力文档  
max.book118.com  
预览与源文档一致, 下载高清无水印

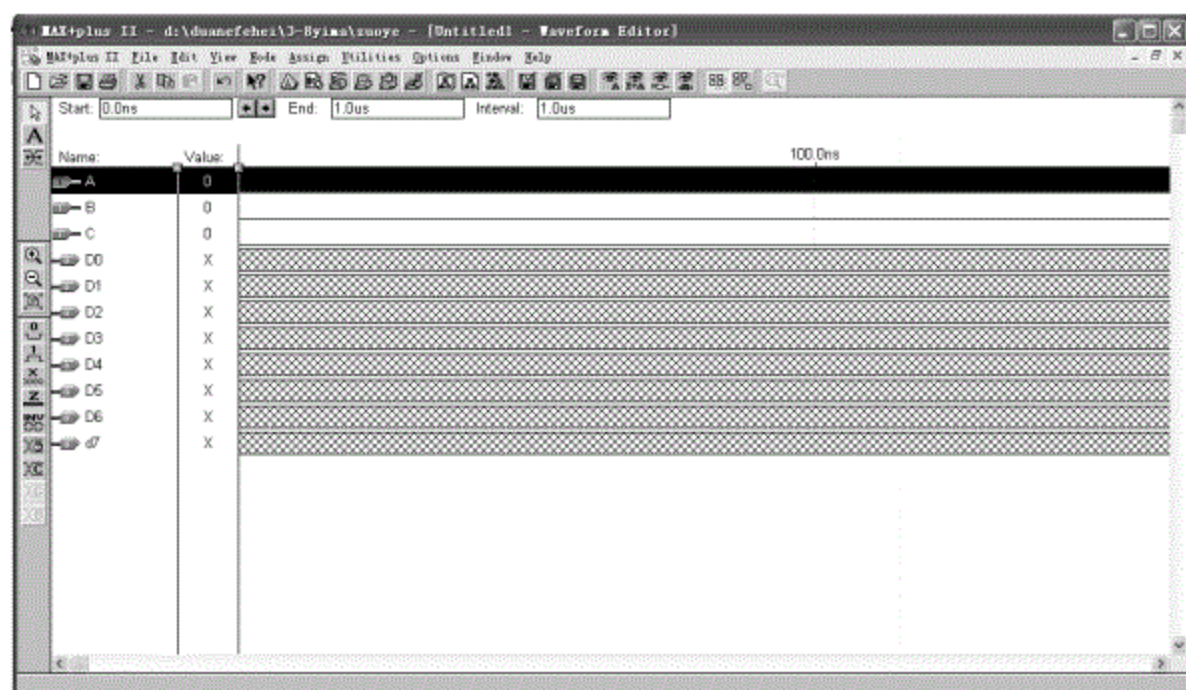


图 2 波形编辑

为输入端口添加激励波形，使用时钟信号。选择初始电平为“0”，时钟周期倍数为“1”。添加完后，波形图如下所示：

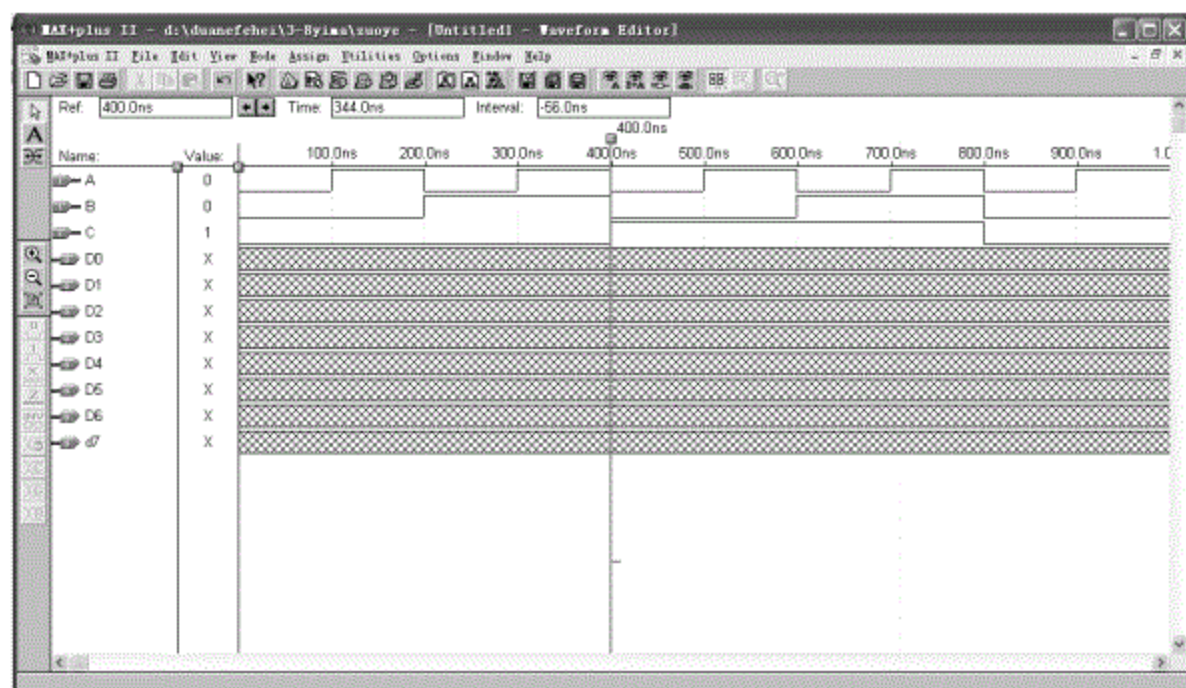


图 3 添加激励后的波形

打开 max+plus II 10.0\Simulator 菜单，确定仿真时间，单击 Start 开始仿真，如下图所示：

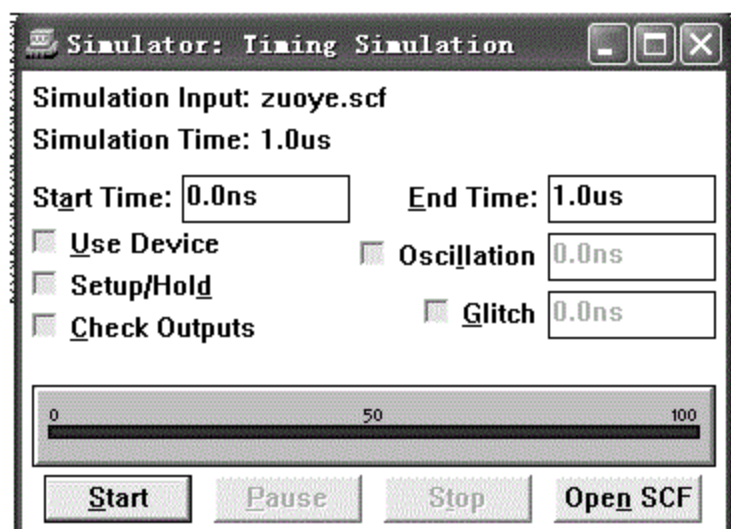


图 4 仿真过程

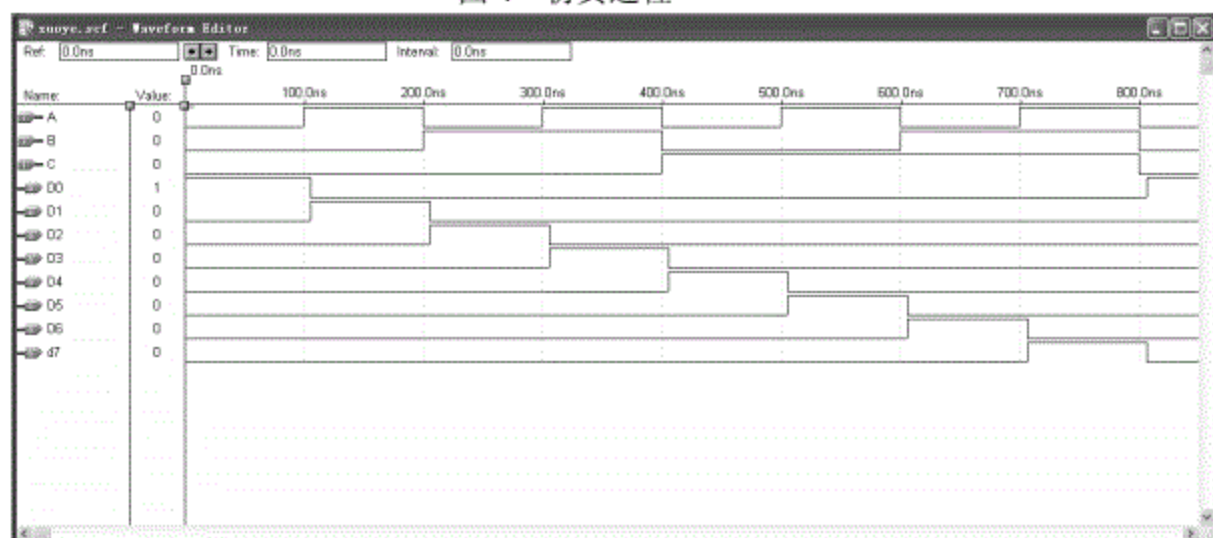


图 5 仿真结果

**四：实验结论：** 使用 max+plus II 10.0 能很好的完成很多电路的仿真与工作。

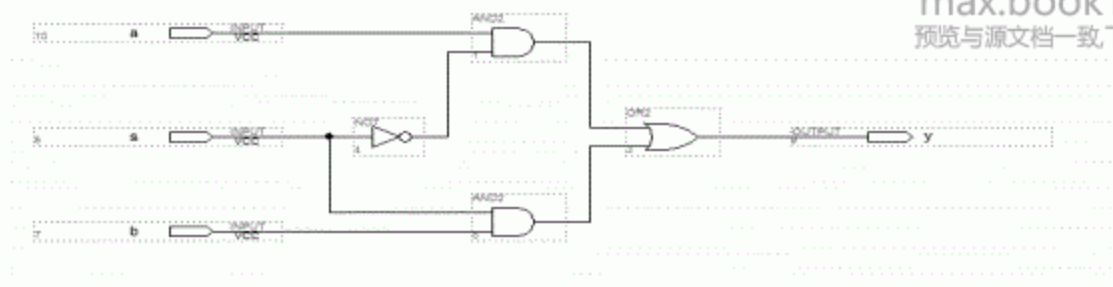
## 实验二、2 选 1 多路选择器

### 一、原理图设计输入法

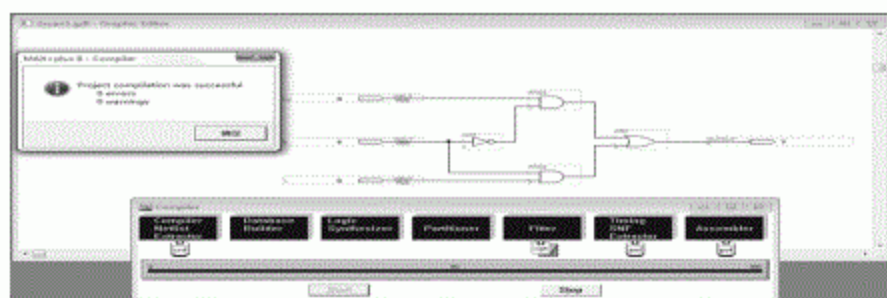
原创力文档

max.book118.com

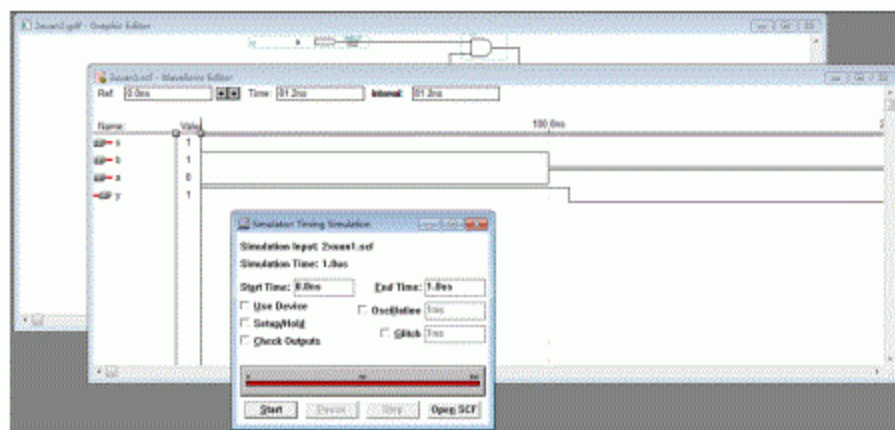
预览与源文档一致, 下载高清无水印



图一 2 选 1 多路选择器结构体



图二 电路编译结果

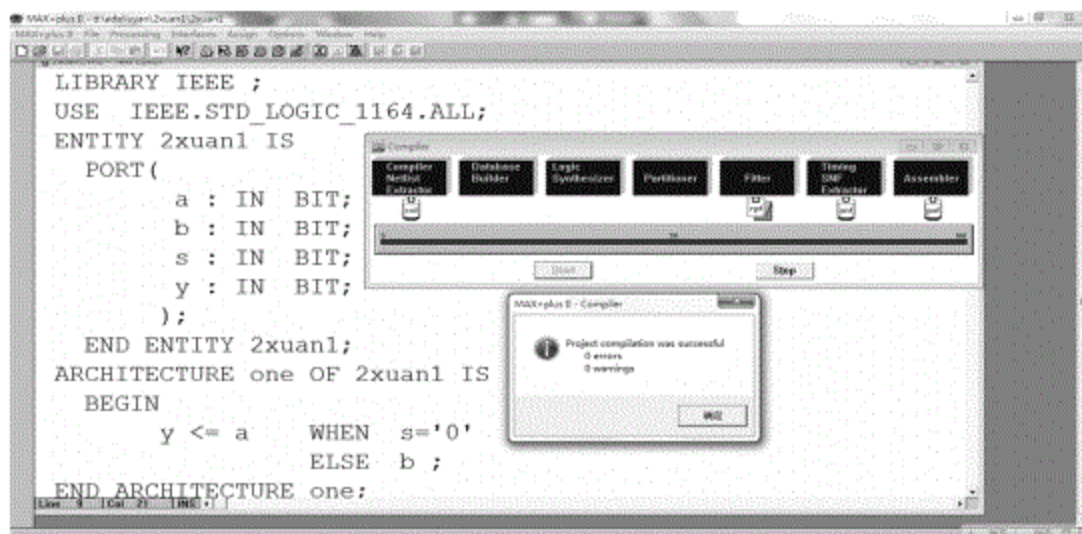


图三 波形仿真

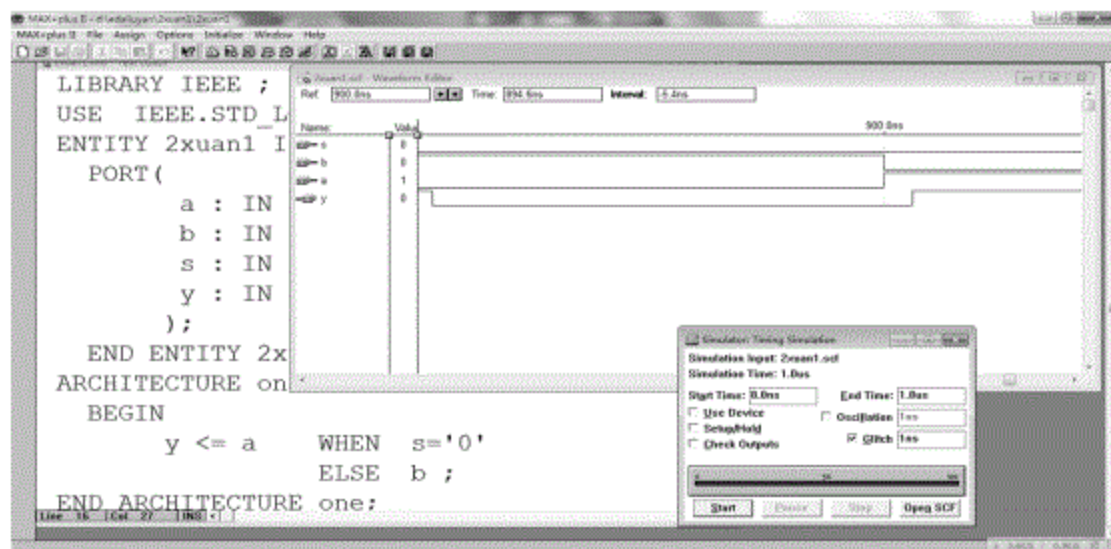
由波形图可知：

当 a、b 两个输入口分别输入不同频率信号时，针对选通控制端 s 上所加的不同电平，输出端 y 将有对应不同信号输出。例如当 s 为低电平时，y 口输出了来自 a 端的较高频率的时钟信号；反之，即当 s 为高电平时，y 口输出了来自 b 端的较低频率的时钟信号。

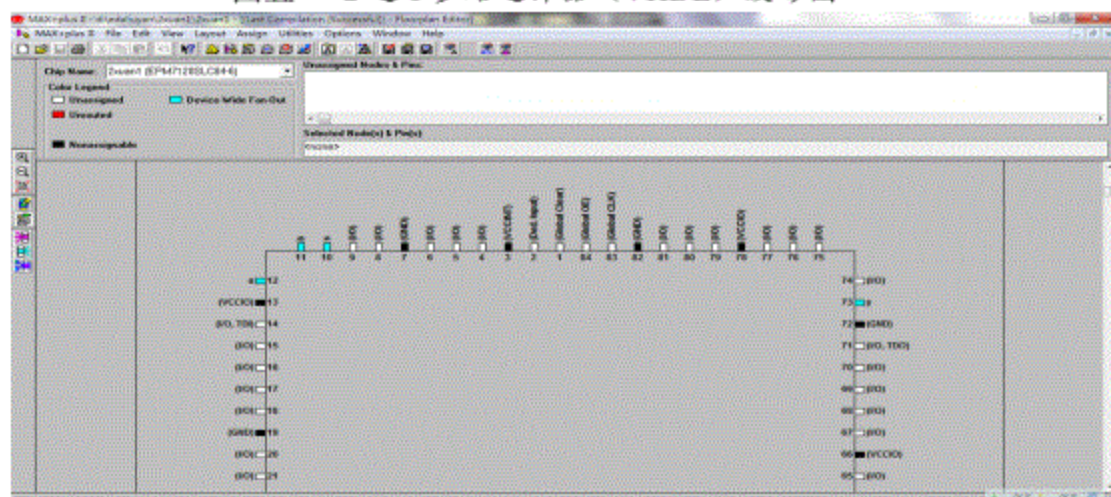
### 二、文本设计输入（VHDL）法



图四 2选1多路选择器（VHDL）



图五 2选1多路选择器（VHDL）波形图



图六 2选1多路选择器（VHDL）引脚分布图

## 实验三、十进制计数器

### 一、VHDL 程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_UNSIGNED.all;
entity CNT10 is
port (CLK,RST,EN,LOAD: IN STD_LOGIC;
      DATA: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      DOUT: out std_logic_vector(3 DOWNTO 0);
      COUT: OUT STD_LOGIC);
ENd entity CNT10;
ARCHITECTURE behav of CNT10 IS
BEGIN
PROCESS (CLK,RST,EN,LOAD)
variable Q: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
IF RST='0' THEN Q:= (OTHERS=>'0');
ELSIF CLK 'EVENT AND CLK ='1' THEN
IF EN='1' THEN
IF (LOAD ='0') THEN Q:=DATA; ELSE
IF Q<9 THEN Q:=Q+1;
ELSE Q:=(OTHERS=>'0');
END IF;
END IF;
END IF;
END IF;
IF Q="1001" THEN COUT<='1';
else COUT<='0'; END IF;
```

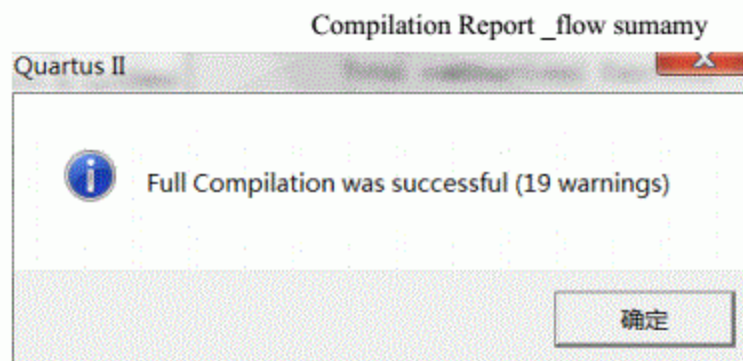
DOUT <=Q;

END PROCESS;

END behav;

它是一个带有异步复位和同步加载功能的十进制加法计数器。

## 二、编译报告

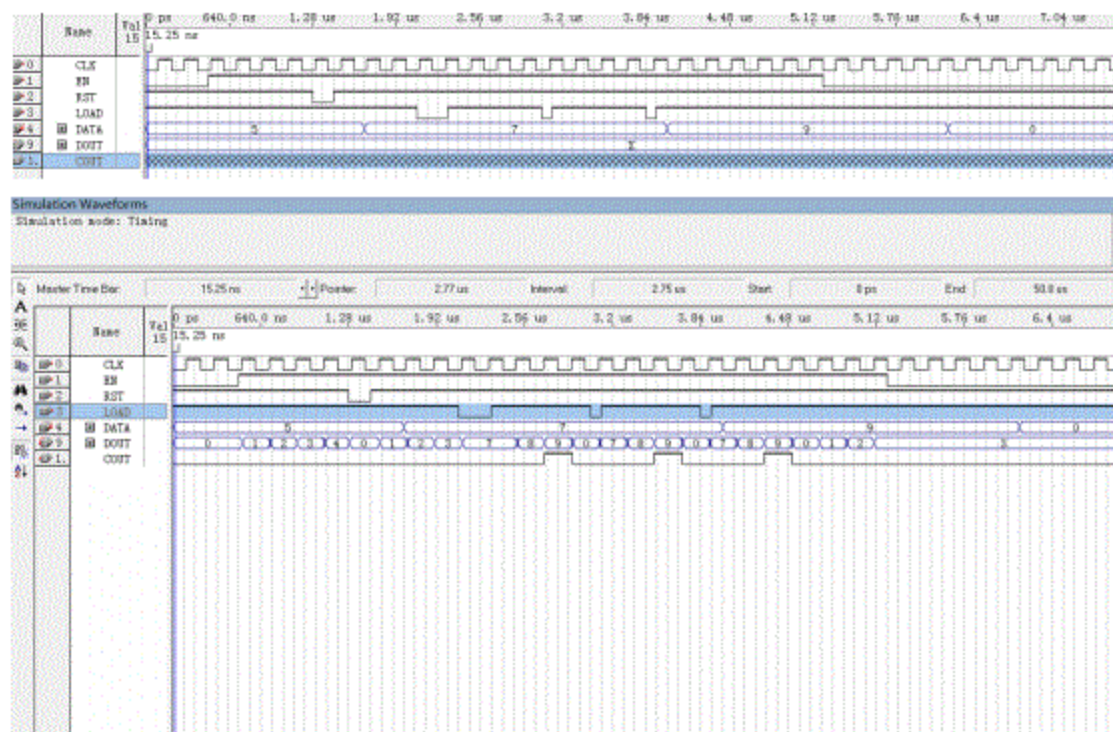


|                                    |                                               |
|------------------------------------|-----------------------------------------------|
| Flow Status                        | Successful - Mon Dec 03 17:22:49 2012         |
| Quartus II Version                 | 9.0 Build 235 06/17/2009 SP 2 SJ Full Version |
| Revision Name                      | cnt10                                         |
| Top-level Entity Name              | CNT10                                         |
| Family                             | Cyclone III                                   |
| Met timing requirements            | N/A                                           |
| Total logic elements               | 9 / 5,136 ( < 1 % )                           |
| Total combinational functions      | 9 / 5,136 ( < 1 % )                           |
| Dedicated logic registers          | 4 / 5,136 ( < 1 % )                           |
| Total registers                    | 4                                             |
| Total pins                         | 13 / 183 ( 7 % )                              |
| Total virtual pins                 | 0                                             |
| Total memory bits                  | 0 / 423,936 ( 0 % )                           |
| Embedded Multiplier 9-bit elements | 0 / 46 ( 0 % )                                |
| Total PLLs                         | 0 / 2 ( 0 % )                                 |
| Device                             | EP3C5F256C6                                   |
| Timing Models                      | Final                                         |

原创力文档  
max.book118.com  
预览与源文档一致, 下载高清无水印

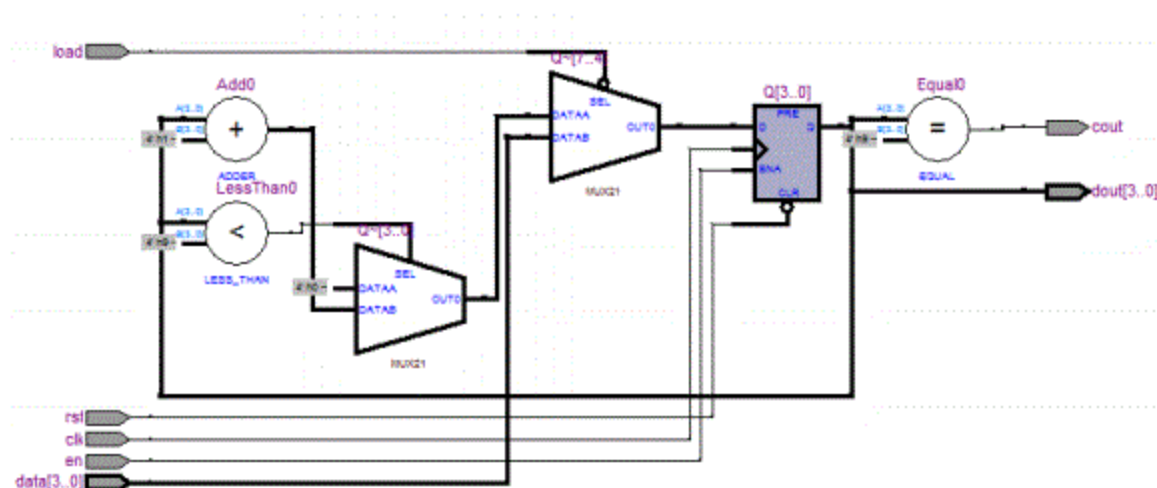


cnt10.vwf

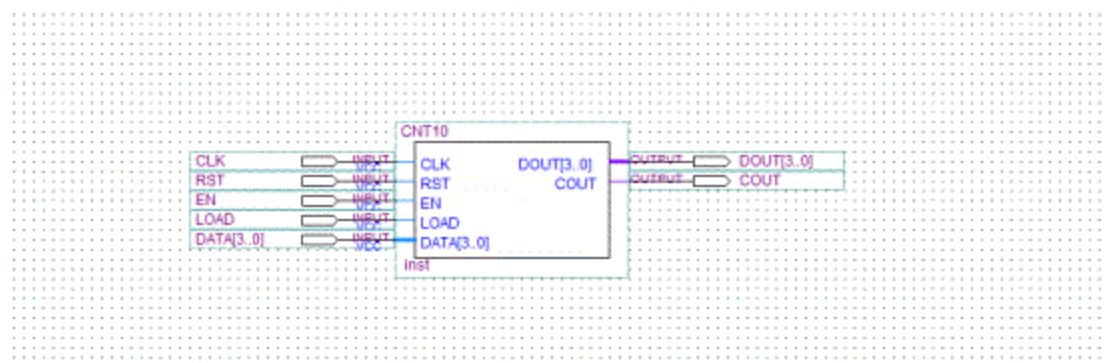


由图可知，（1）当计数使能 EN 为高电平时允许计数；RST 低电平时计数器被清零。（2）由于 LOAD 是同步加载控制信号，其第一个负脉冲恰好在 CLK 的上升沿处，故将 5 加载于计数到 9，出现了第一个进位脉冲。由于 LOAD 第二个负脉冲未在 CLK 上升沿处，故没有发生加载操作，而第 3、4 个负脉冲都出现了加载操作；（3）当计数器每次计到 9 时，输出为高电平，而且计数器又从 0 开始重新计数

### 三、RTL 图



#### 四、symbol cnt10.bdf

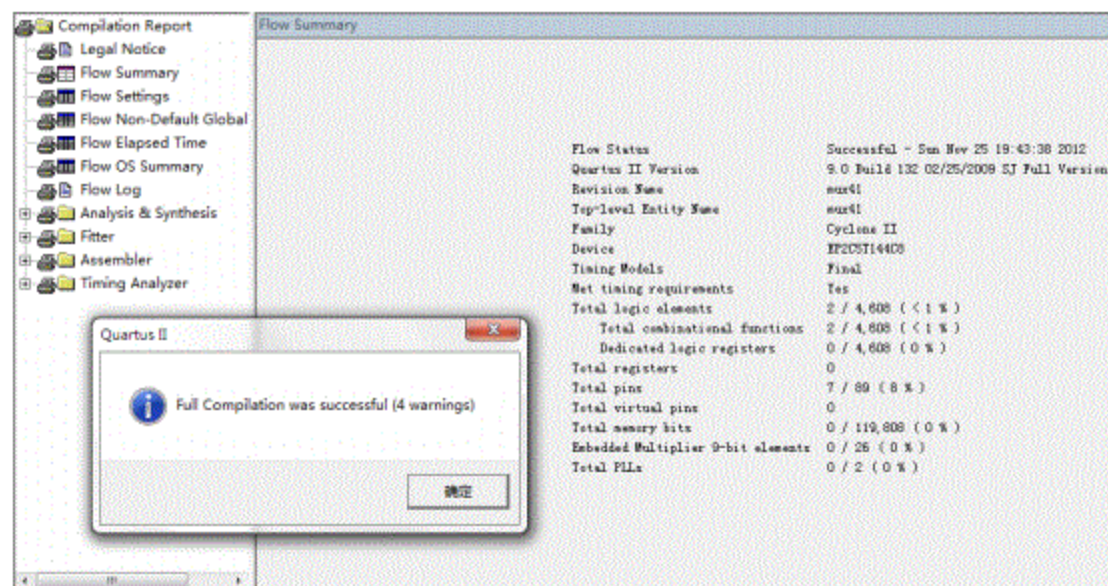


## 实验四、四选一多路选择器

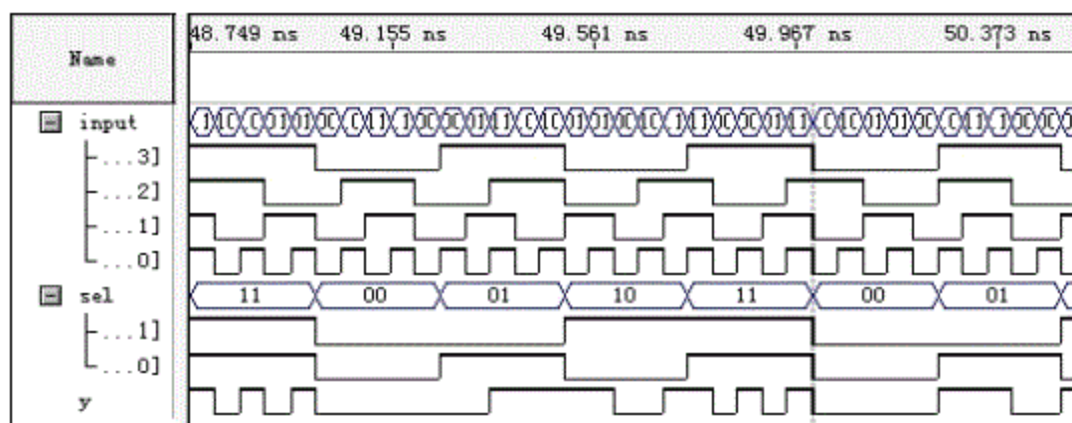
### 一、用 IF\_THEN 语句实现 4 选 1 多路选择器

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  ENTITY mux41 IS
4  PORT (a,b,c,d: IN STD_LOGIC;
5        s0: IN STD_LOGIC;
6        s1: IN STD_LOGIC;
7        y: OUT STD_LOGIC);
8  END ENTITY mux41;
9  ARCHITECTURE if_mux41 OF mux41 IS
10     SIGNAL s0s1 : STD_LOGIC_VECTOR(1 DOWNTO 0);--定义标准逻辑位矢量数据
11 BEGIN
12     s0s1<=s1&s0; --s1相并s0,即s1与s0并置操作
13     PROCESS(s0s1,a,b,c,d)
14     BEGIN
15         IF s0s1 = "00" THEN y <= a;
16         ELSIF s0s1 = "01" THEN y <= b;
17         ELSIF s0s1 = "10" THEN y <= c;
18         ELSE y <= d;
19     END IF;
20     END PROCESS;
21 END ARCHITECTURE if_mux41;
22
```

图一 用 IF\_THEN 语句实现 4 选 1 多路选择器文本设计输入



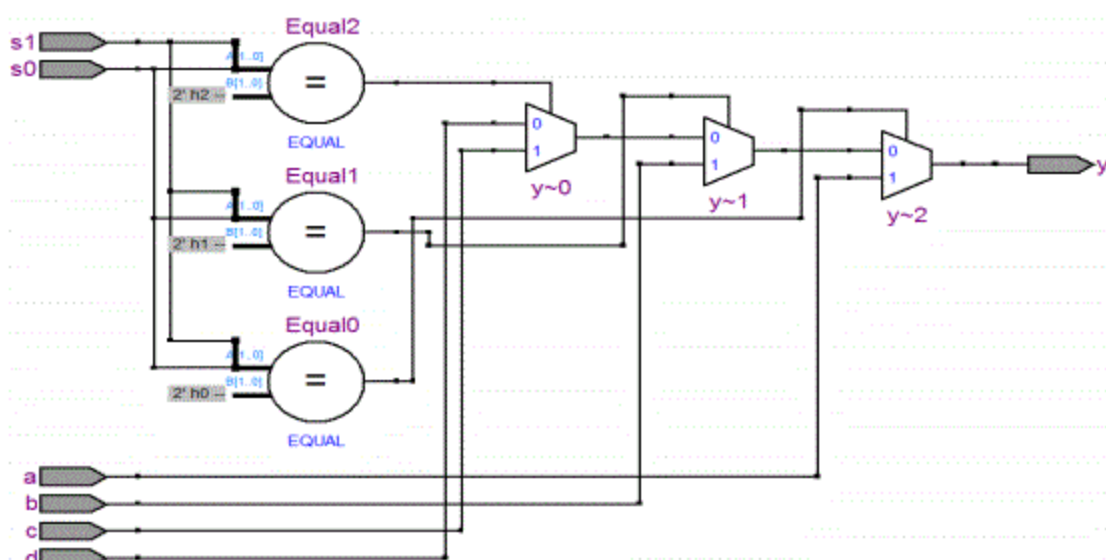
图二 程序运行编译结果



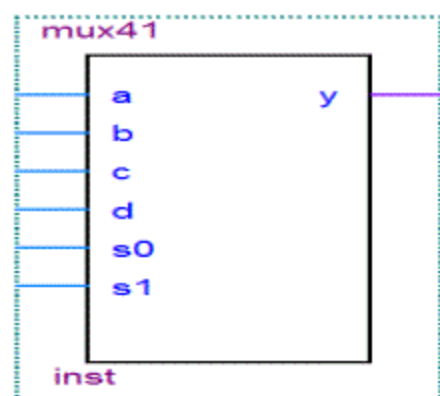
图三 四选一多路选择器的电路仿真波形图

由上图可知：

当 sel=11 时,  $y=input3$ ; 当 sel=10 时,  $y=input2$ ; 当 sel=01 时,  $y=input1$ ; 当 sel=00 时,  $y=input0$ ; 实现了四选一功能。



图四 4 选 1 多路选择器 RTL 电路图



图五 4 选 1 多路选择器 Symbol

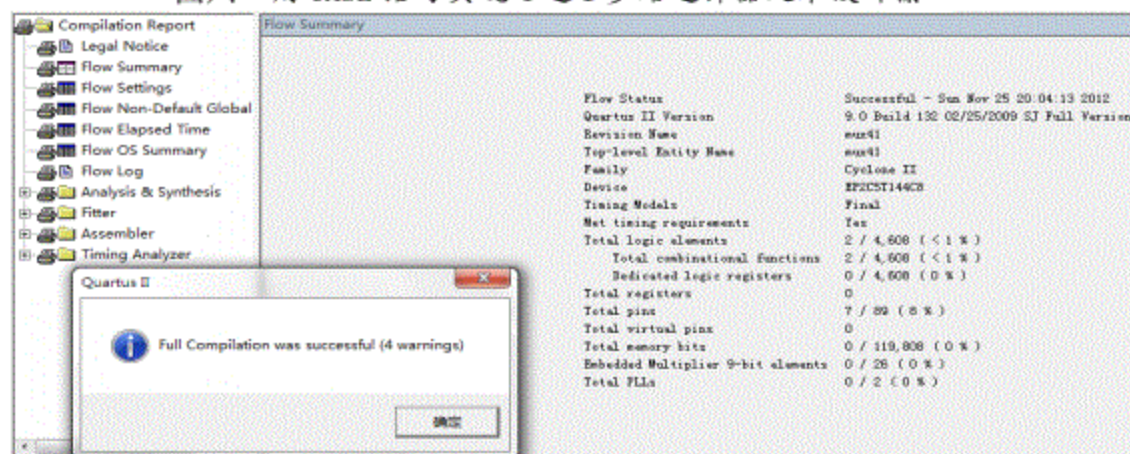
## 二、用 CASE 语句实现 4 选 1 多路选择器

```

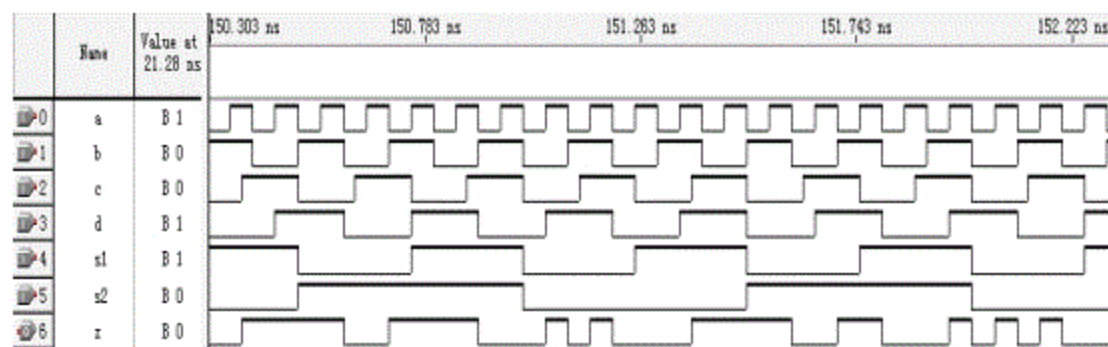
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
--ARCHITECTURE case_mux41 OF mux41 IS
    SIGNAL s0s1 : STD_LOGIC_VECTOR(1 DOWNTO 0);--定义标准逻辑位矢量数据类型
--BEGIN
    s0s1<=s1&s0;    --s1相并s0,即s1与s0并置操作
--    PROCESS(s0s1,a,b,c,d)
--    BEGIN
--        CASE s0s1 IS --类似于真值表的case语句
--            WHEN "00" => y <= a;
--            WHEN "01" => y <= b;
--            WHEN "10" => y <= c;
--            WHEN "11" => y <= d;
--            WHEN OTHERS => NULL ;
--        END CASE;
--    END PROCESS;
END ARCHITECTURE case_mux41;

```

图六 用 CASE 语句实现 4 选 1 多路选择器文本设计输入



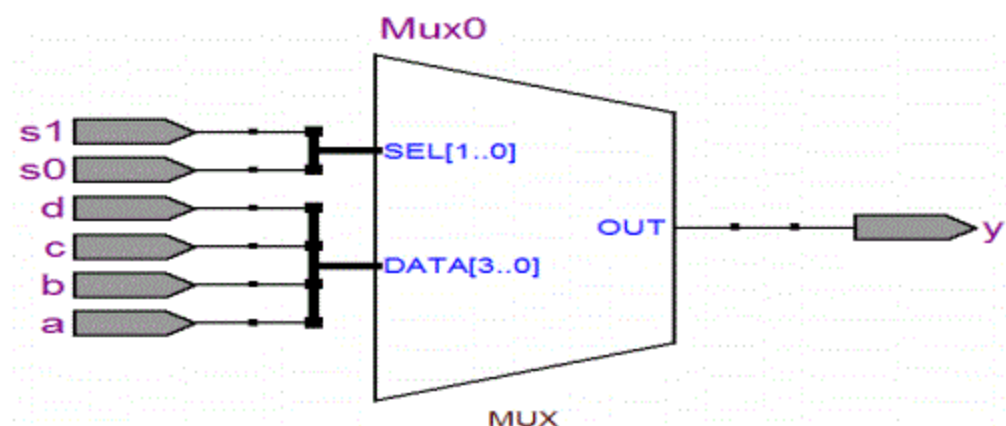
图七 程序运行编译结果



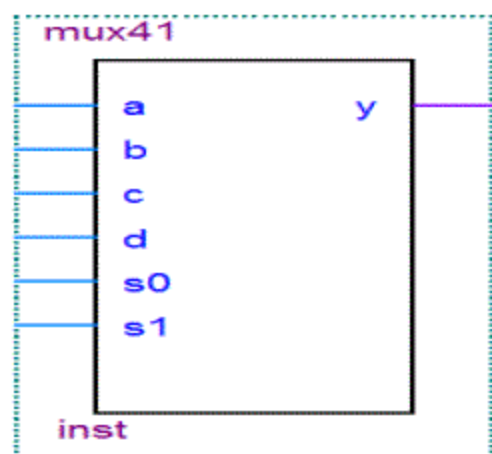
图八 四选一多路选择器的电路仿真波形图

由上图可知 ( $s <= s1 \& s2$ ) :

当  $s=00$  时,  $z=a$ ; 当  $s=01$  时,  $z=b$ ; 当  $s=10$  时,  $z=c$ ; 当  $s=11$  时,  $z=d$ ; 实现了四选一功能。



图九 4 选 1 多路选择器 RTL 电路图



图十 4 选 1 多路选择器 Symbol

### 三、用 WHEN\_ELSE 语句实现 4 选 1 多路选择器

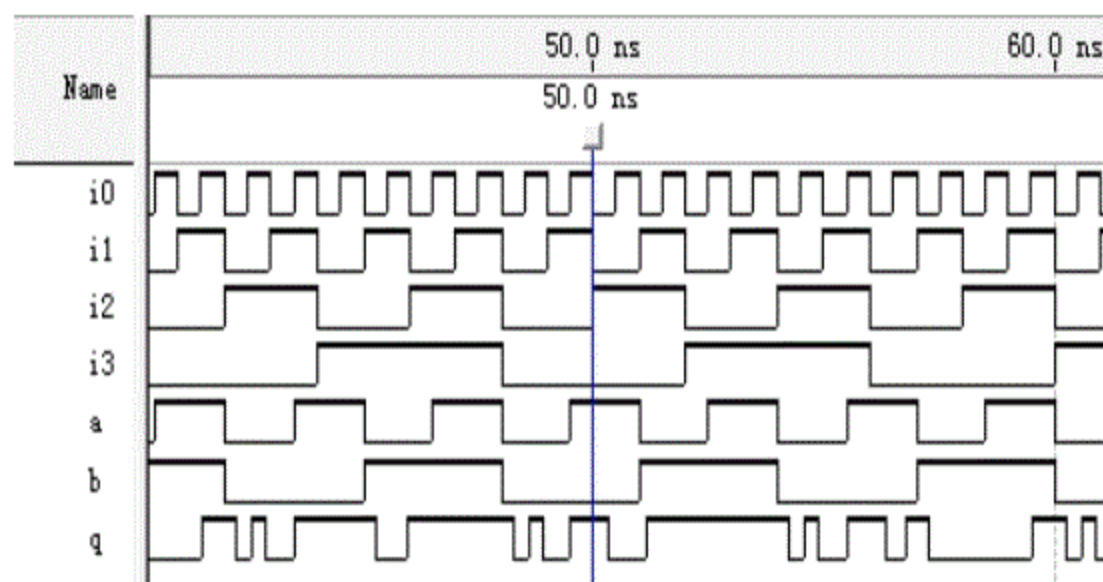
```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
-- architecture when_else_mux41 of mux41 is
    signal s :std_logic_vector(1 downto 0);
-- begin
    s<=s1 & s0;
    y<= a when s="00"else
        b when s="01"else
        c when s="10"else
        d when s="11"
        else '0';

end architecture when_else_mux41;

```

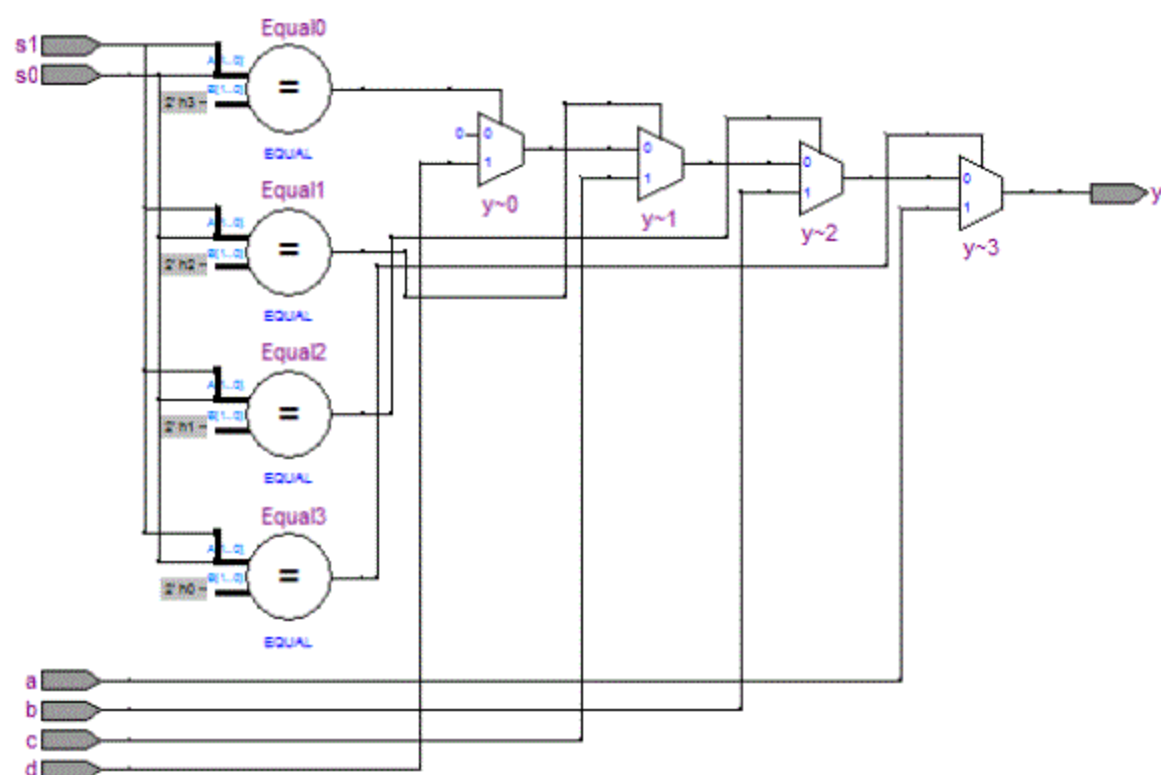
图十一 用 WHEN\_ELSE 语句实现 4 选 1 多路选择器文本设计输入



图十二 四选—多路选择器的电路仿真波形图

由上图可知 ( $sel \leq b \& a$ ) :

当  $sel=00$  时,  $q=i0$ ; 当  $sel=01$  时,  $q=i1$ ; 当  $sel=10$  时,  $q=i2$ ; 当  $sel=11$  时,  $q=i3$ ; 实现了四选—功能。



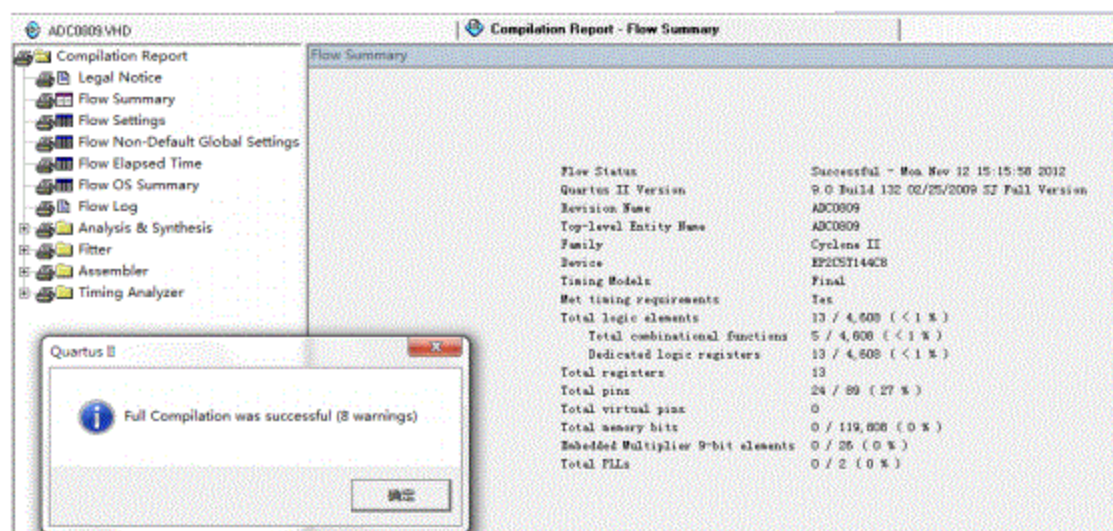
图十三 4 选 1 多路选择器 RTL 电路图

## 实验五、ADC0809 采样状态机

### 一、文本设计输入（VHDL）法

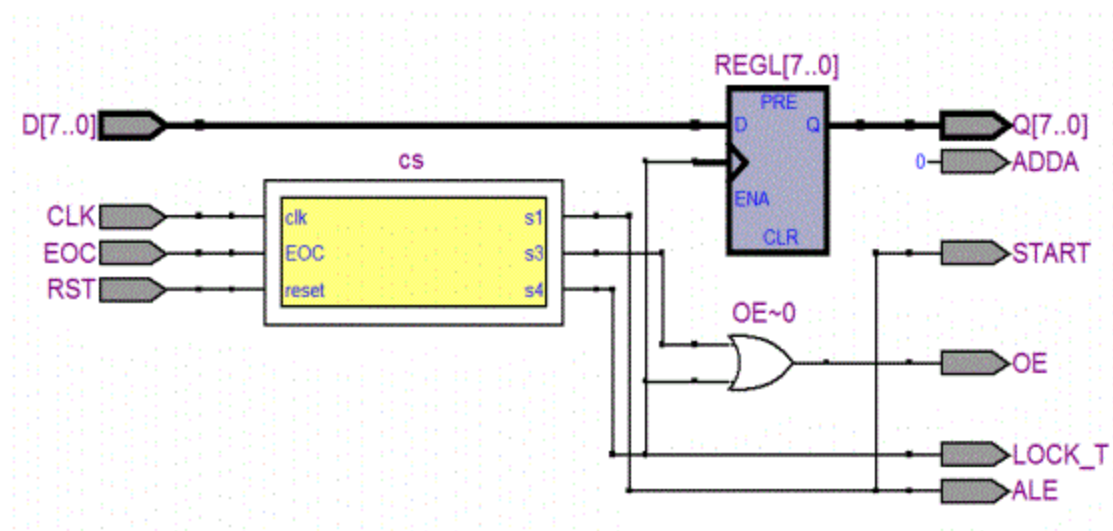
```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  ENTITY ADC0809 IS
4  PORT (D : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
5        CLK,RST : IN STD_LOGIC;
6        EOC : IN STD_LOGIC;
7        ALE : OUT STD_LOGIC;
8        START, OE : OUT STD_LOGIC;
9        ADDA,LOCK_T:OUT STD_LOGIC;
10       Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
11  END ADC0809;
12  ARCHITECTURE behav OF ADC0809 IS
13  TYPE states IS(s0,s1,s2,s3,s4);
14  SIGNAL cs,next_state : states :=s0;
15  SIGNAL REGL : STD_LOGIC_VECTOR(7 DOWNTO 0);
16  SIGNAL LOCK :STD_LOGIC;
17  BEGIN
18  ADDA <='0'; LOCK_T<=LOCK;
19  COM :PROCESS(cs,EOC) BEGIN
20  CASE cs IS
21  WHEN s0=>ALE<='0';START<='0';OE<='0';LOCK<='0';next_state <=s1;
22  WHEN s1=>ALE<='1';START<='1';OE<='0';LOCK<='0';next_state <=s2;
23  WHEN s2=>ALE<='0';START<='0';OE<='0';LOCK<='0';
24  IF (EOC='1') THEN next_state <=s3;
25  ELSE next_state <=s2; END IF;
26  WHEN s3=>ALE<='0';START<='0';OE<='1';LOCK<='0';next_state <=s4;
27  WHEN s4=>ALE<='0';START<='0';OE<='1';LOCK<='1';next_state <=s0;
28  WHEN OTHERS=>ALE<='0';START<='0';OE<='0';LOCK<='0';next_state <=s0;
29  END CASE;
30  END PROCESS COM;
31  REG: PROCESS (CLK,RST) BEGIN
32  IF RST='1' THEN cs<=s0;
33  ELSIF CLK'EVENT AND CLK='1' THEN cs<=next_state; END IF;
34  END PROCESS REG;
35  LATCH1 :PROCESS (LOCK) BEGIN
36  IF LOCK='1' AND LOCK'EVENT THEN REGL <=D; END IF;
37  END PROCESS LATCH1;
38  Q<= REGL;
39  END behav;
40
```

图一 ADC0809 采样状态机文本设计输入



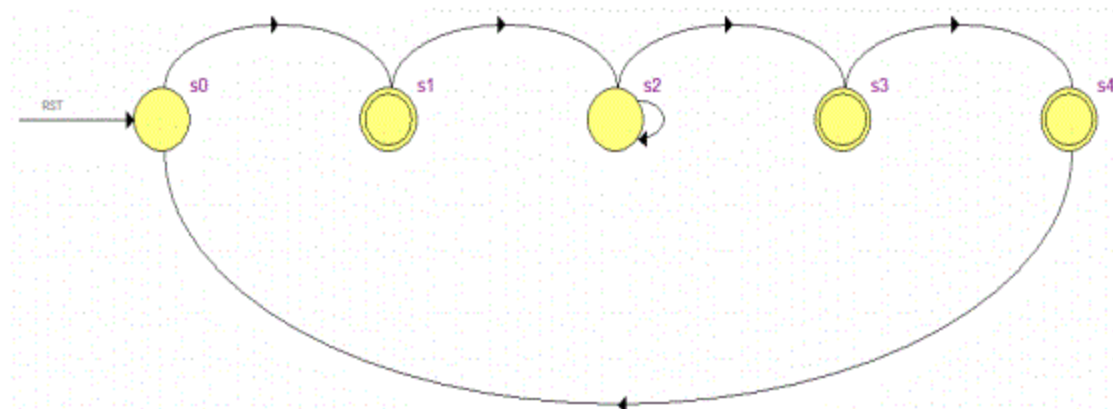
图二 程序运行编译结果

## 二、RTL 电路图



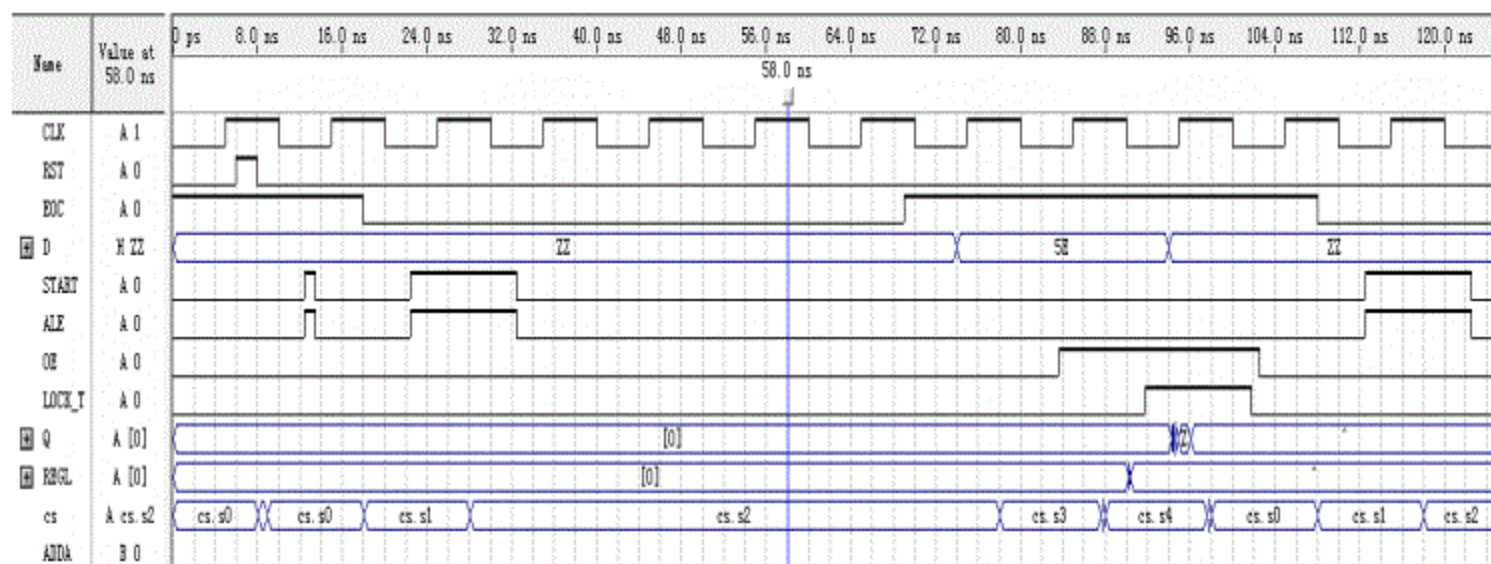
图三 ADC0809 采样状态机 RTL 电路图

## 三、ADC0809 采样状态图



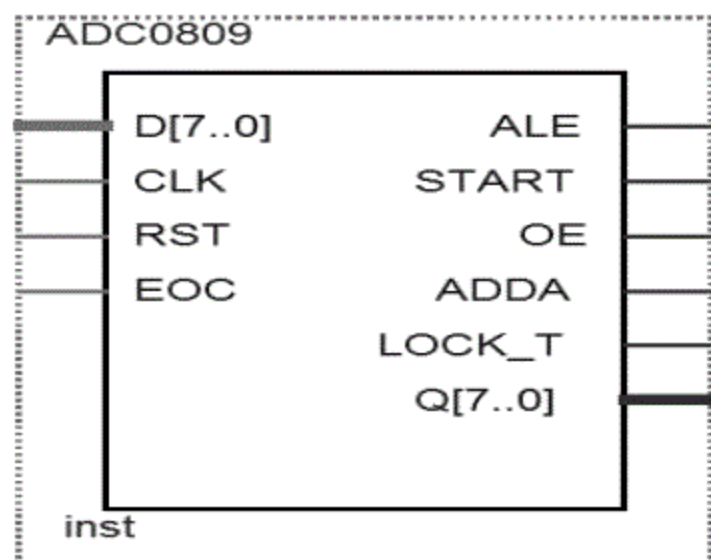
图四 ADC0809 采样状态图

#### 四、ADC0809 采样状态机工作时序



图五 ADC0809 采样状态机工作时序图

上图显示了一个完整的采样周期。复位信号后进入状态 s0；第二个时钟上升沿后，状态机进入状态 s1，由 start、ale 发出采样和地址选通的控制信号。而后，eoc 由高电平变为低电平，ADC0809 的 8 位数据输出端呈现高阻状态“ZZ”。在状态 s2，等待了 clk 的数个时钟周期之后，eoc 变为高电平，表示转换结束；进入状态 s3，在此状态的输出允许 oe 被设置成高电平。此时 ADC0809 的数据输出端 d[7..0] 即输出已经转换好的数据 5EH。在状态 s4，lock\_t 发出一个脉冲，其上升沿立即将 d 端口的 5E 锁入 q 和 regl 中。



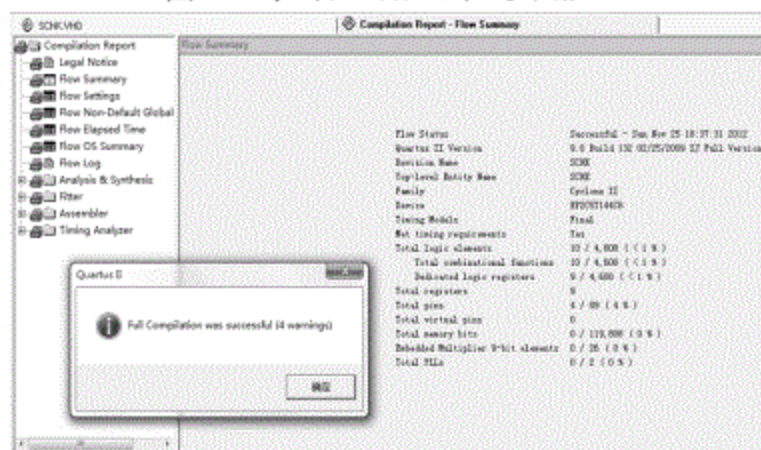
图六 ADC0809 采样状态机 Symbol

## 实验六、11010011 序列检测

### 一、文本设计输入 (VHDL) 法

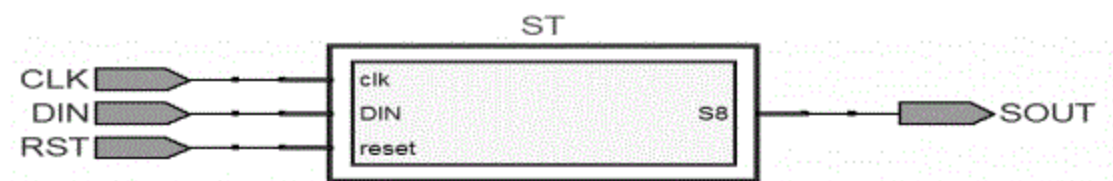
```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  ENTITY SCHK IS
4  PORT(DIN,CLK,RST : IN STD_LOGIC;
5        SOUT : OUT STD_LOGIC);
6  END SCHK;
7  ARCHITECTURE behav OF SCHK IS
8  TYPE states IS (S0,S1,S2,S3,S4,S5,S6,S7,S8);
9  SIGNAL ST,NST:states :=S0;
10 BEGIN
11 COMPROCESS(ST,DIN) BEGIN
12 CASE ST IS --11010011
13 WHEN S0=> IF DIN ='1' THEN NST<=S1;ELSE NST<=S0;END IF;
14 WHEN S1=> IF DIN ='1' THEN NST<=S2;ELSE NST<=S0;END IF;
15 WHEN S2=> IF DIN ='0' THEN NST<=S3;ELSE NST<=S0;END IF;
16 WHEN S3=> IF DIN ='1' THEN NST<=S4;ELSE NST<=S0;END IF;
17 WHEN S4=> IF DIN ='0' THEN NST<=S5;ELSE NST<=S0;END IF;
18 WHEN S5=> IF DIN ='0' THEN NST<=S6;ELSE NST<=S0;END IF;
19 WHEN S6=> IF DIN ='1' THEN NST<=S7;ELSE NST<=S0;END IF;
20 WHEN S7=> IF DIN ='1' THEN NST<=S8;ELSE NST<=S0;END IF;
21 WHEN S8=> IF DIN ='0' THEN NST<=S3;ELSE NST<=S0;END IF;
22 WHEN OTHERS => NST <=S0;
23 END CASE;
24 END PROCESS;
25 REG:PROCESS(CLK,RST) BEGIN
26 IF RST='1' THEN ST<=S0;
27   ELIF CLK'EVENT AND CLK='1' THEN ST <=NST; END IF;
28 END PROCESS REG;
29 SOUT <='1' WHEN ST=S8 ELSE '0';
30 END behav;
```

图一 序列检测器文本设计输入



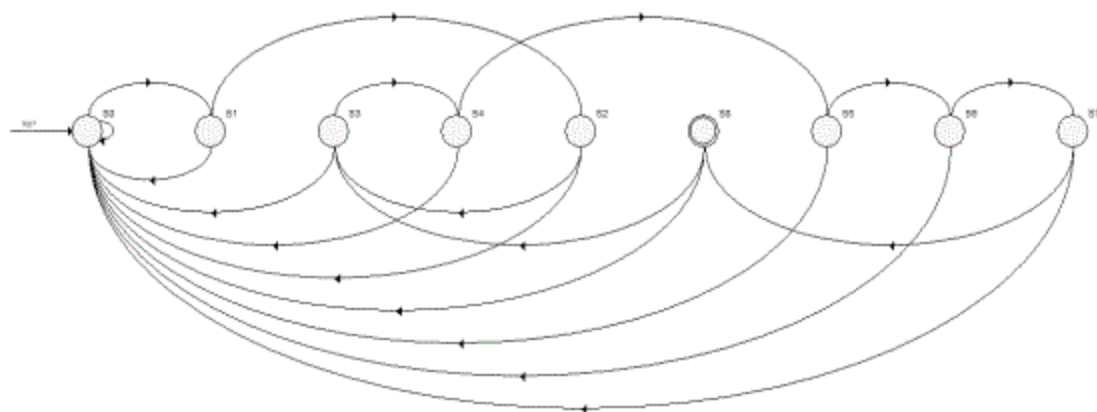
图二 程序运行编译结果

### 二、序列检测器 RTL 电路图



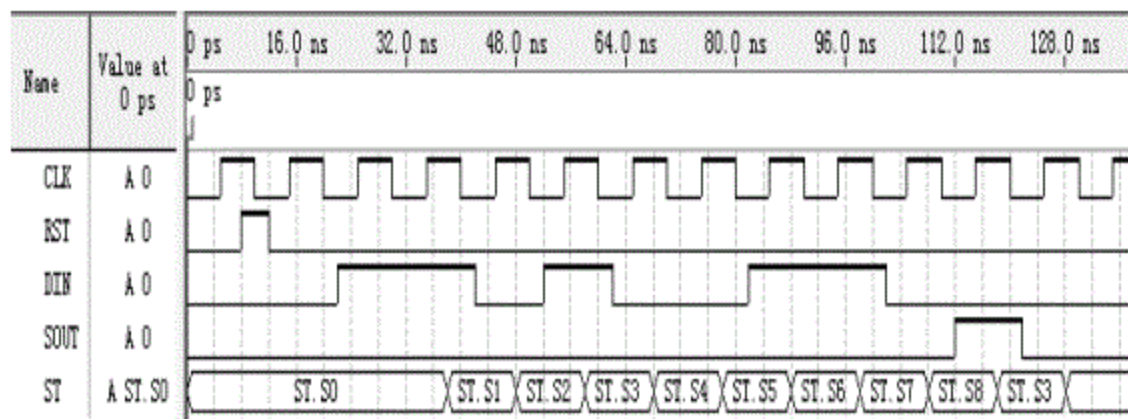
图三 序列检测器 RTL 电路图

### 三、序列检测器状态图



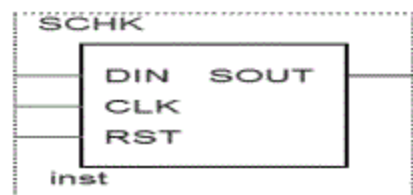
图四 序列检测器状态图

### 四、序列检测器时序仿真波形



图五 序列检测器时序仿真波形

由上图可知，当有正确序列进入时，到了状态 8 时，输出序列正确标志 SOUT=1。而当下一位数据为零时，即 DIN=0，进入状态四 s3（这时测出的数据 110 恰好与原序列数的头三位相同）。



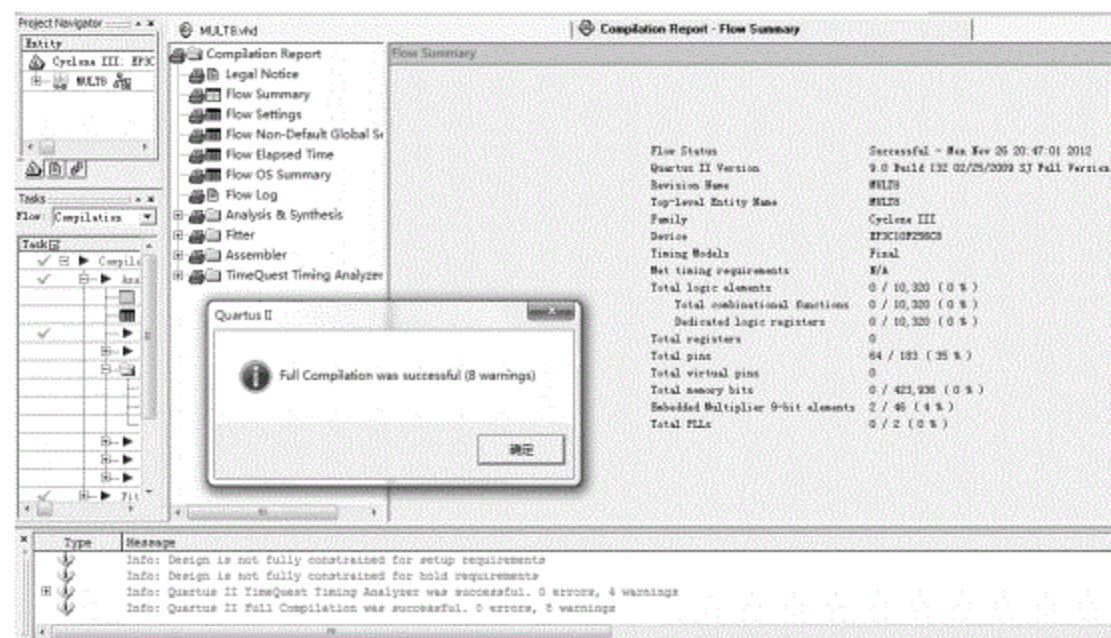
图六 序列检测器 Symbol

## 实验七、两个 8 位乘 8 位的有符号数乘法器

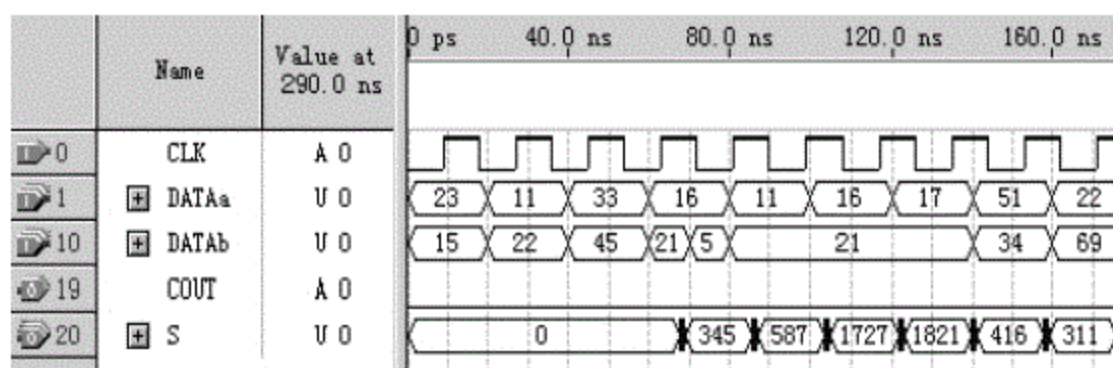
### 一、文本设计输入 (VHDL) 法

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
entity MULT8 is
    port(A1, B1, A2, B2 : in signed(7 downto 0);--定义有符号位矢
         R1, R2 : out signed(15 downto 0));
end entity MULT8;
architecture bhv of MULT8 is
begin
    R1 <= A1 * B1;
    R2 <= A2 * B2;
end bhv;
```

图一 两个 8 位乘 8 位的有符号数乘法器文本设计输入



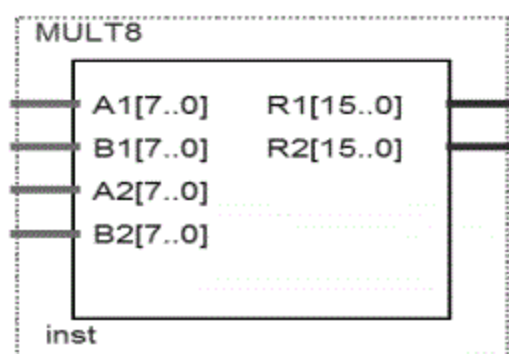
图二 程序运行编译结果



图三 仿真波形

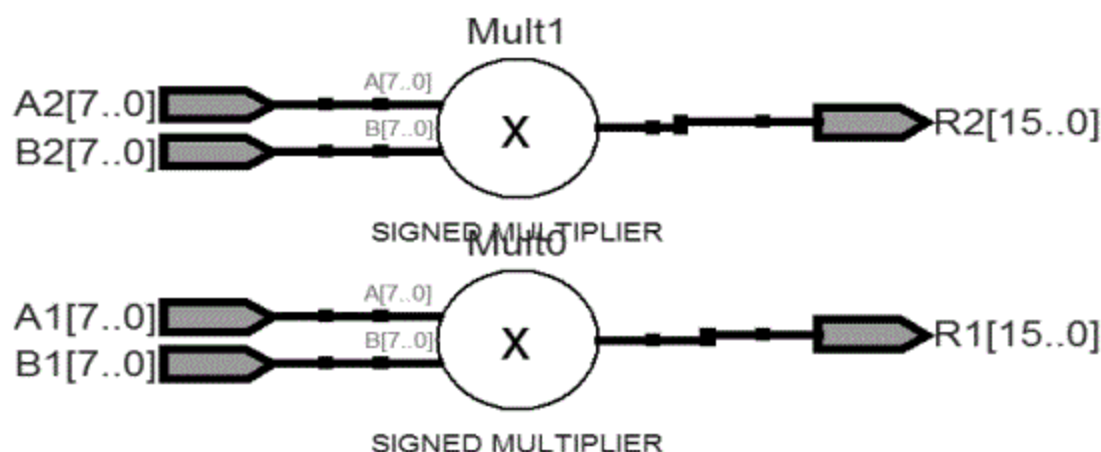
由波形可知，在 CLK 的第 4 个上升沿后才得到第一个计算数据，之前都是 0。第 4 个上升沿后得到的结果为  $s=0 \times 0 + 23 \times 15 = 345$ ；第 5 个上升沿后得到结果为  $s=23 \times 15 + 11 \times 22 = 587$ ；第 6 个上升沿后得到结果为  $s=11 \times 22 + 33 \times 45 = 1727$ ；第 7 个上升沿后得到结果为  $s=33 \times 45 + 16 \times 21 = 1821$ ；第 8 个上升沿后得到结果为  $s=16 \times 21 + 16 \times 5 = 416$ ；第 9 个上升沿后得到结果为  $s=16 \times 5 + 11 \times 21 = 311$ ；

## 二、两个 8 位乘 8 位的有符号数乘法器 Symbol



图四 两个 8 位乘 8 位的有符号数乘法器 Symbol

## 三、两个 8 位乘 8 位的有符号数乘法器 RTL 电路图



图五两个 8 位乘 8 位的有符号数乘法器 RTL 电路图

## 实验八、全加器

### 一、文本设计输入 (VHDL) 法

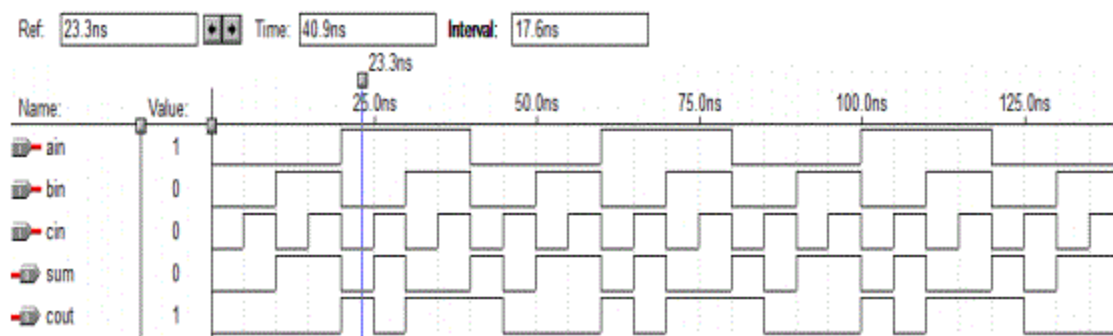
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY F_adder IS
    PORT(ain,bin,cin : in std_logic;
          cout,sum : out std_logic);
END ENTITY F_adder;
ARCHITECTURE fd1 of F_adder IS
    COMPONENT H_adder IS
        PORT( a,b: in std_logic;
              co,so: out std_logic);
    END COMPONENT;
    COMPONENT or2a IS
        PORT( a,b: in std_logic;
              c: out std_logic);
    END COMPONENT;

    SIGNAL d,e,f : STD_LOGIC;
BEGIN
    u1 : H_adder PORT MAP(a=>ain,b=>bin,co=>d,so=>e);
    u2 : H_adder PORT MAP(a=>e ,b=>cin,co=>f,so=>sum);
    u3 : or2a PORT MAP(a=>d ,b=>f ,c=>cout);
END ARCHITECTURE fd1;
```

图一 全加器文本设计输入



图二 仿真结果

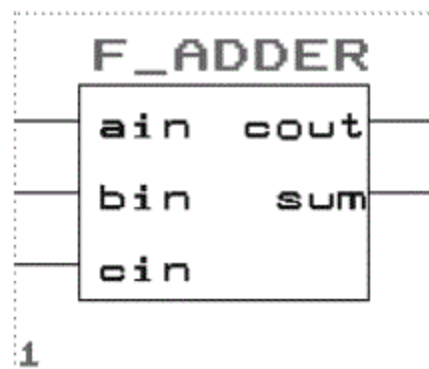


图三 全加器波形仿真图

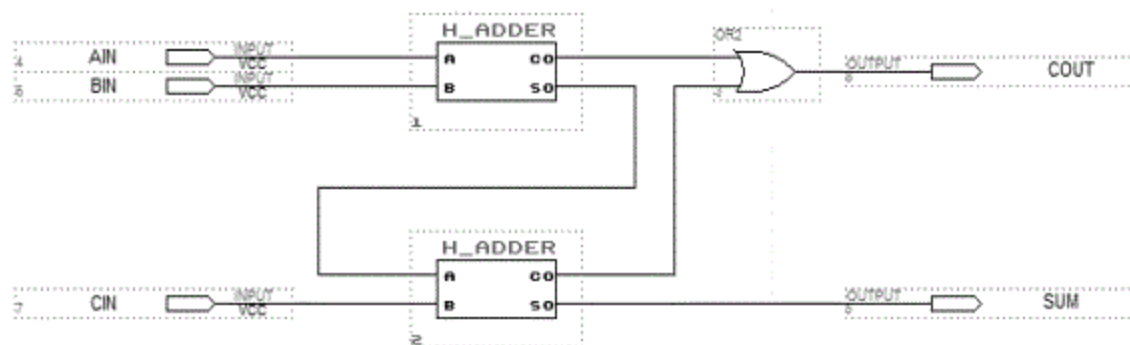
全加器真值表如下：

| AIN | BIN | CIN | COUNT | SUM |
|-----|-----|-----|-------|-----|
| 0   | 0   | 0   | 0     | 0   |
| 0   | 0   | 1   | 0     | 1   |
| 0   | 1   | 0   | 0     | 1   |
| 0   | 1   | 1   | 1     | 0   |
| 1   | 0   | 0   | 0     | 1   |
| 1   | 0   | 1   | 1     | 0   |
| 1   | 1   | 0   | 1     | 0   |
| 1   | 1   | 1   | 1     | 1   |

对比真值表和仿真波形，加数 AIN,BIN 和进位 CIN 共有 8 总情况，和 SUM 和进位 COUNT 共有 4 总情况，波形和真值表一致



图四 全加器波实体模块



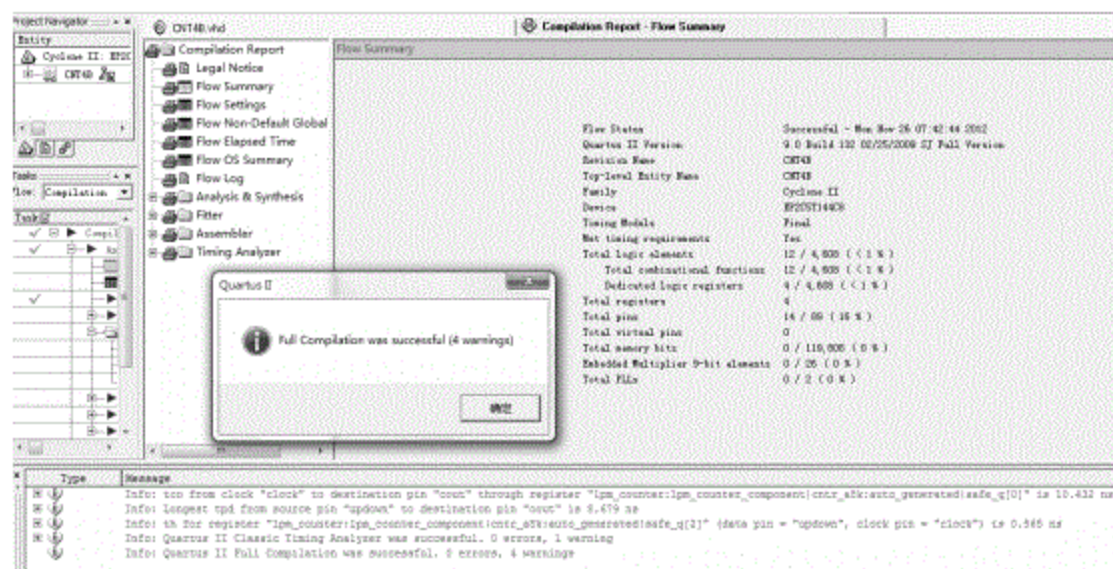
图五 全加器 F\_adder 电路图

## 实验九、LPM\_COUNTER 计数模块

### 一、文本设计输入 (VHDL) 法

```
library ieee;
use ieee.std_logic_1164.all;
library lpm;      --打开LPM库
use lpm.all;      --打开LPM程序包
entity CNT4B is
    --异步清零、时钟使能、时钟输入、同步预置数加载控制、加减控制
    port (aclr, clk_en, clock, sload, updown : in std_logic;
          data : in std_logic_vector(3 downto 0);  --4位预置数
          cout : out std_logic;                    -- 进位输出
          q : out std_logic_vector(3 downto 0)); --计数器输出
end CNT4B;
architecture SYN of CNT4B is
    signal sub_wire0 : std_logic;
    signal sub_wire1 : std_logic_vector(3 downto 0);
    component lpm_counter      --以下是参数传递说明语句
    generic(lpm_direction,lpm_port_updown, lpm_type : string;--定义字符串类
            lpm_modulus, lpm_width : natural );              --定义正整数类型
    port(sload, clk_en, aclr, clock, updown : in std_logic;
          cout : out std_logic;
          q : out std_logic_vector(3 downto 0);
          data : in std_logic_vector(3 downto 0));
    end component;
begin
    cout <= sub_wire0;
    q <= sub_wire1(3 downto 0);
    lpm_counter_component : lpm_counter generic map( --参数传递例化语句
        lpm_direction => "unused",                  --单方向计数参数未
        lpm_modulus => 12,                           --定义模12计数器
        lpm_port_updown => "port_used",               --使用加减计数
        lpm_type => "lpm_counter",                   --计数器类型
        lpm_width => 4)                               --计数位宽
    port map ( sload => sload, clk_en => clk_en, aclr => aclr, clock => clock,
        data => data, updown => updown, cout => sub_wire0, q => sub_wire1);
end SYN;
```

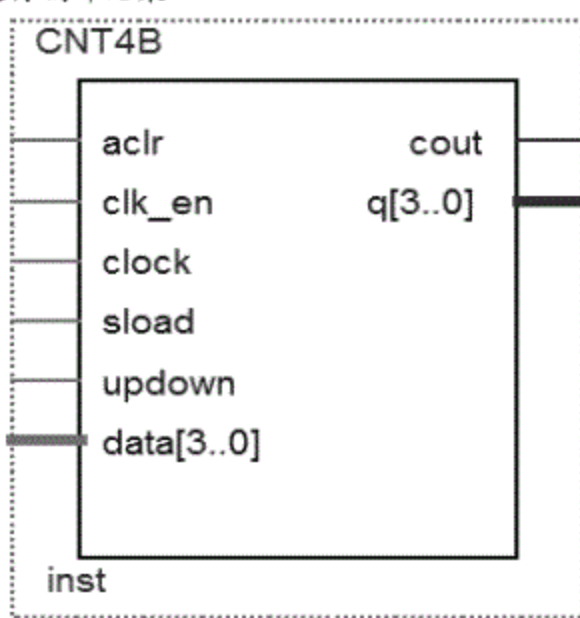
图一 LPM\_COUNTER 计数模块文本设计输入



图二 程序运行编译结果

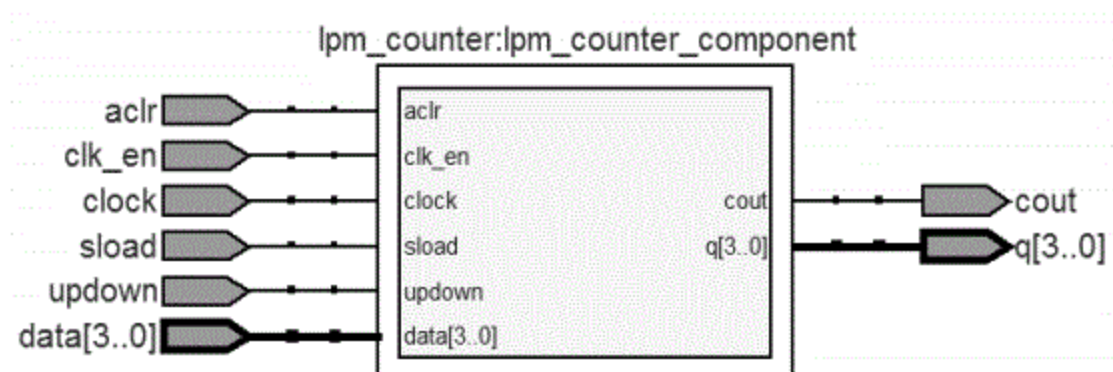


图三 LPM\_COUNTER 计数模块



图四 LPM\_COUNTER 计数模块 Symbol

## 二、LPM\_COUNTER 计数模块 RTL 电路图



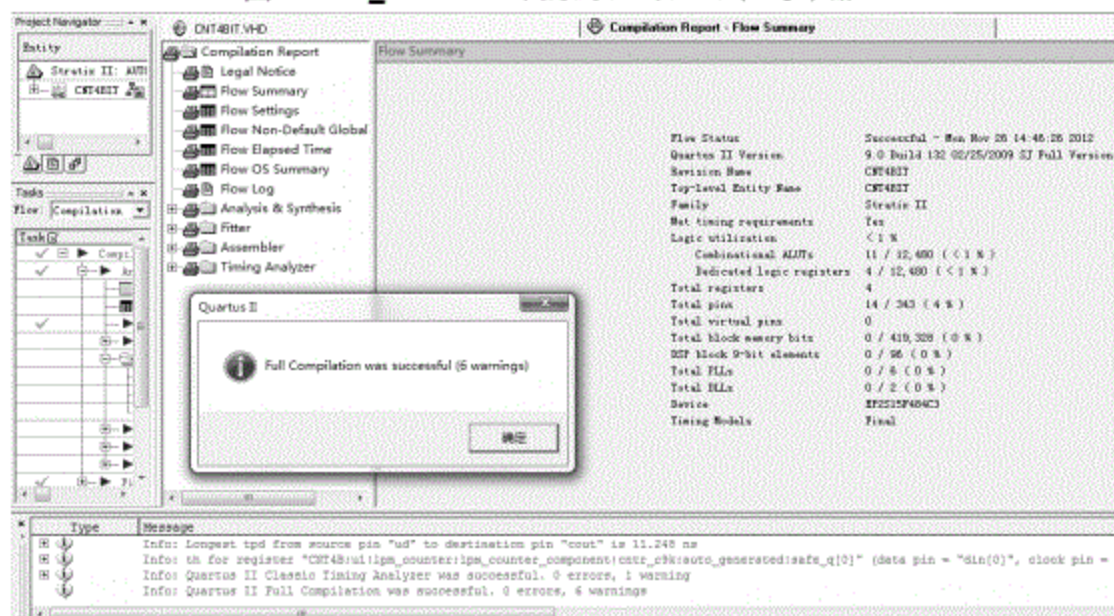
图五 LPM\_COUNTER 计数模块 RTL 电路图

## 实验十、LPM\_COUNTER 计数模块例化

### 一、文本设计输入 (VHDL) 法

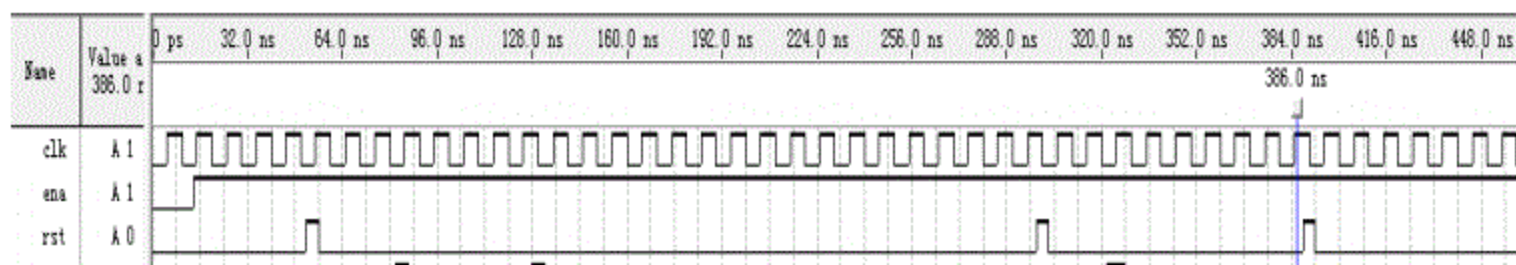
```
library ieee;
use ieee.std_logic_1164.all;
entity CNT4BIT is
    port (clk, rst, ena, sld, ud : in std_logic;
          din : in std_logic_vector(3 downto 0);
          cout : out std_logic;
          dout : out std_logic_vector(3 downto 0));
end entity CNT4BIT;
architecture TRANSLATED of CNT4BIT is
    component CNT4B --元件声明
    port (aclr, clk_en, clock, sload, updown : in std_logic;
          data : in std_logic_vector(3 downto 0);
          cout : out std_logic;
          q : out std_logic_vector(3 downto 0));
    end component;
begin --元件例化
    u1 : CNT4B port map ( sload => sld, clk_en => ena, aclr => rst, cout => cout,
                          clock => clk, data => din, updown => ud, q => dout);
end architecture TRANSLATED;
```

图一 LPM\_COUNTER 计数模块例化文本设计输入



图二程序运行编译结果

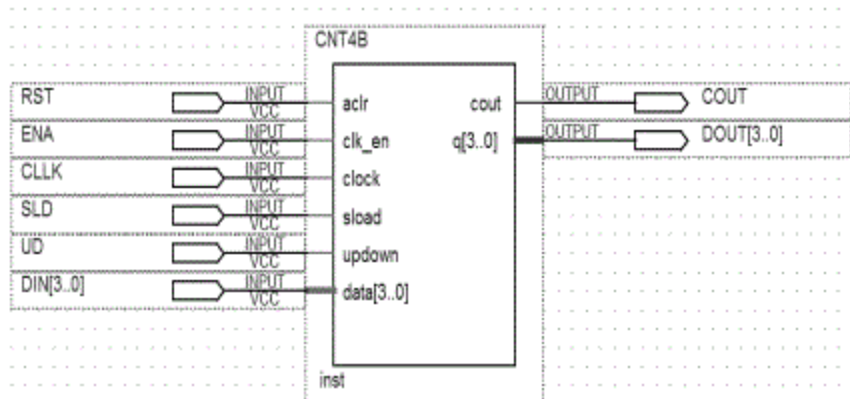
### 二、CNT4BIT.V 仿真波形



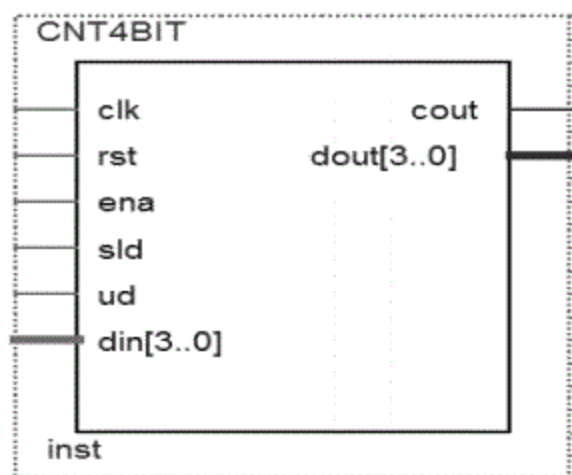
图三 CNT4BIT.V 的仿真波形

由仿真波形图可知：

在第 2 个 SLD 加载信号在没有 CLK 上升沿处发生时，无法进行加载，显然 SLD 是同步的。从波形中可以了解此计数器模块的功能和性能。

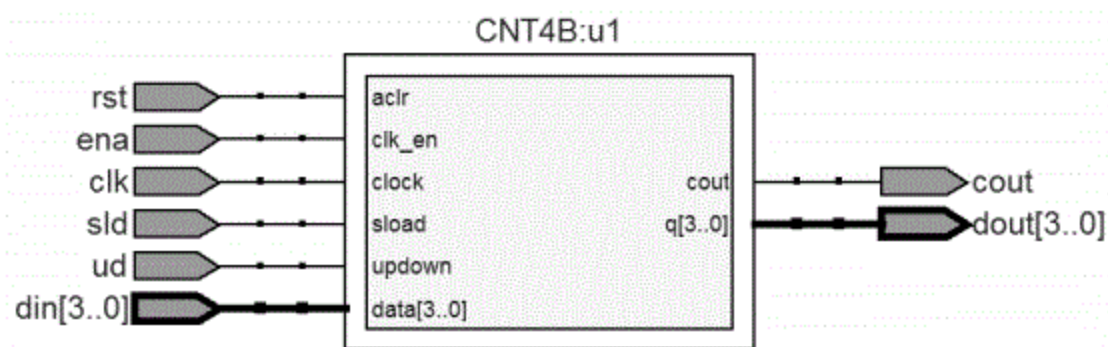


图四 CNT4BIT 原理图输入设计



图五 CNT4BIT 计数模块 Symbol

## 二、CNT4BIT 计数模块 RTL 电路图



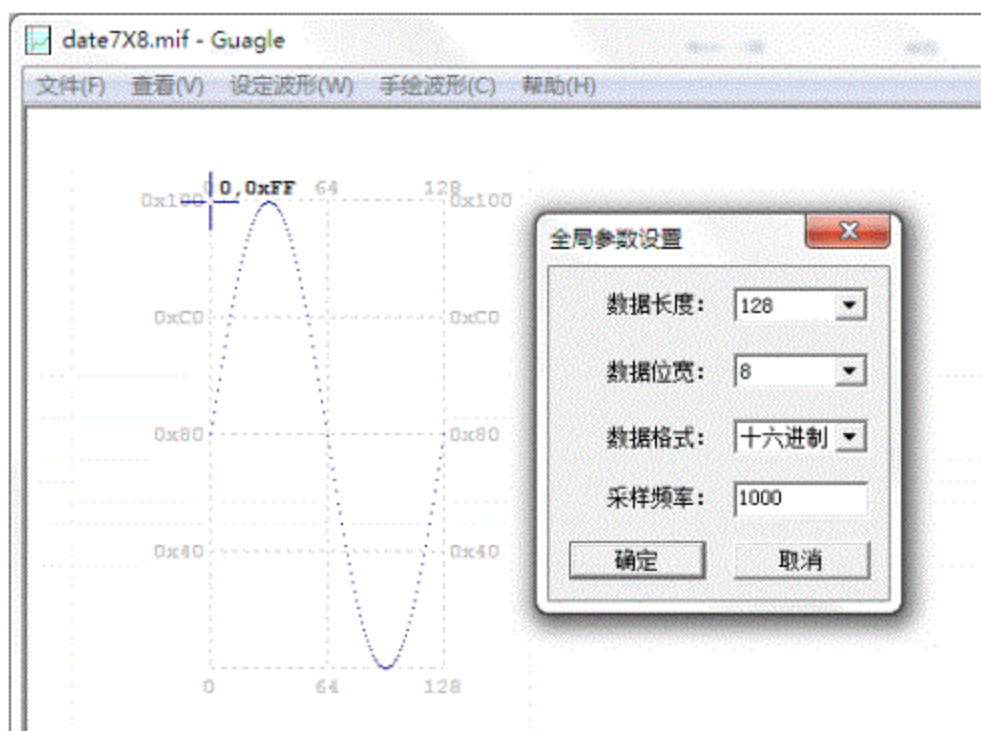
图六 CNT4BIT 计数模块 RTL 电路图

## 实验十一、LPM 随机存储器的设置和调用

### 一、建立 MIF 格式文件

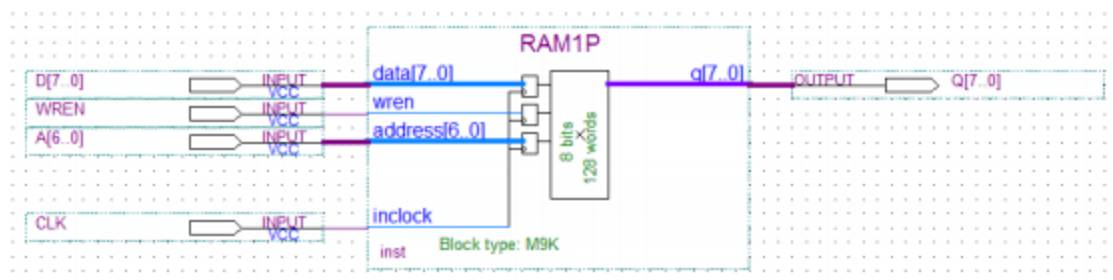
| Addr | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 |
|------|----|----|----|----|----|----|----|----|
| 00   | 80 | 86 | 8C | 92 | 98 | 9E | A5 | AA |
| 08   | B0 | B6 | BC | C1 | C6 | CB | D0 | D5 |
| 10   | DA | DE | E2 | E6 | EA | ED | F0 | F3 |
| 18   | F5 | F8 | FA | FB | FD | FE | FE | FF |
| 20   | FF | FF | FE | FE | FD | FB | FA | F8 |
| 28   | F5 | F3 | F0 | ED | EA | E6 | E2 | DE |
| 30   | DA | D5 | D0 | CB | C6 | C1 | BC | B6 |
| 38   | B0 | AA | A5 | 9E | 98 | 92 | 8C | 86 |
| 40   | 7F | 79 | 73 | 6D | 67 | 61 | 5A | 55 |
| 48   | 4F | 49 | 43 | 3E | 39 | 34 | 2F | 2A |
| 50   | 25 | 21 | 1D | 19 | 15 | 12 | 0F | 0C |
| 58   | 0A | 07 | 05 | 04 | 02 | 01 | 01 | 00 |
| 60   | 00 | 00 | 01 | 01 | 02 | 04 | 05 | 07 |
| 68   | 0A | 0C | 0F | 12 | 15 | 19 | 1D | 21 |
| 70   | 25 | 2A | 2F | 34 | 39 | 3E | 43 | 49 |
| 78   | 4F | 55 | 5A | 61 | 67 | 6D | 73 | 79 |

图一 MIF 文件编辑窗

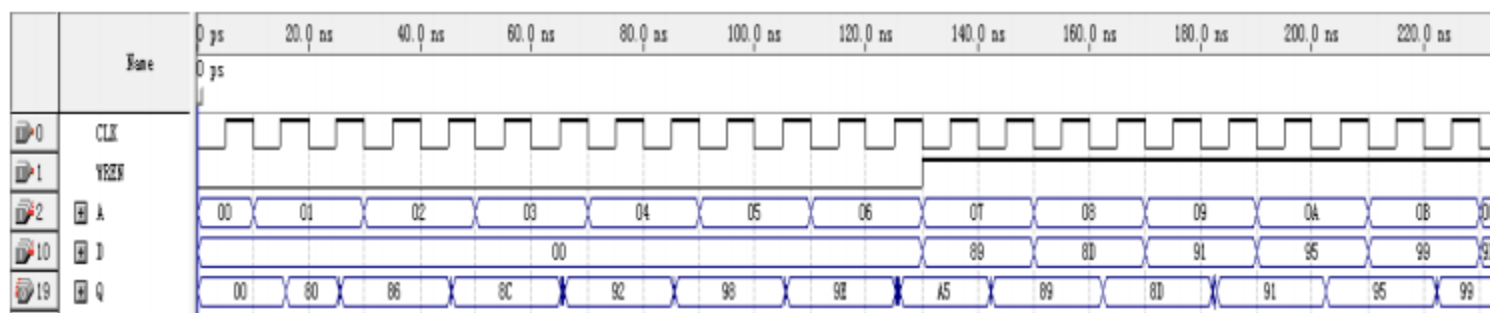


图二 利用康芯 MIF 生成 MIF 正弦波数据文件

### 二、对 LPM\_RAM 仿真测试



图三 在原理图编辑器上连接好的 RAM 模块



图四 RAM 仿真波形

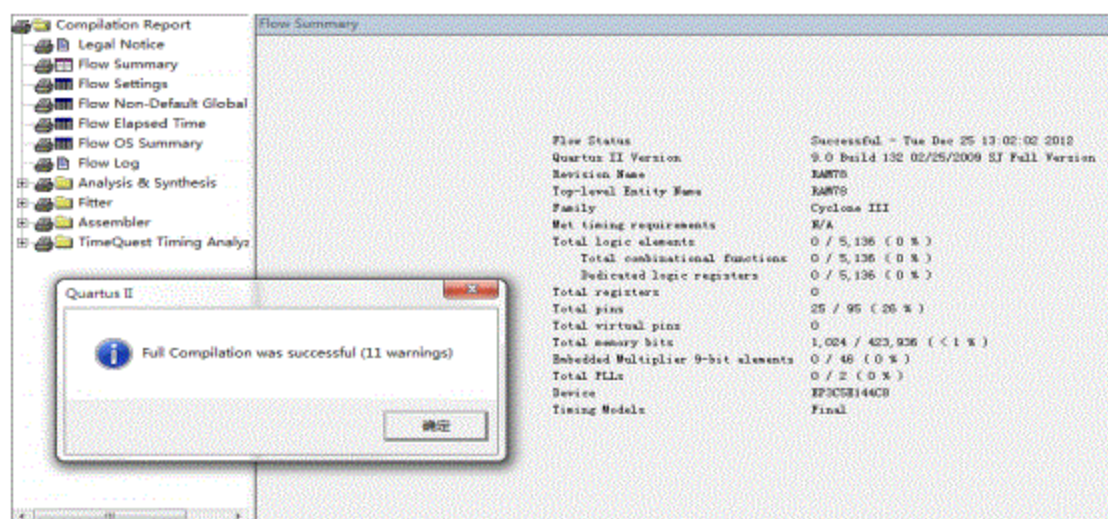
### 三、利用用户自定义数据类型语句来实现存储器描述

```

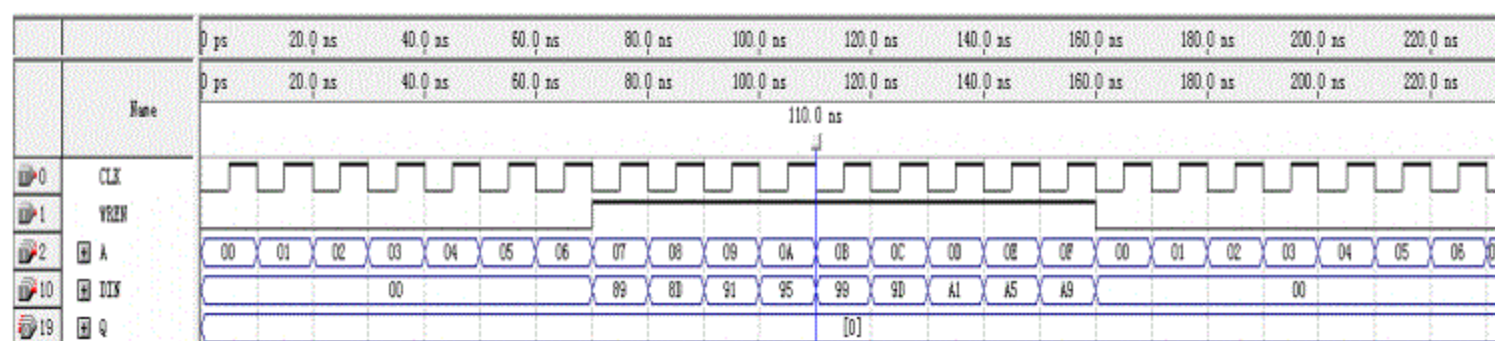
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
-- ENTITY RAM78 IS
-- PORT (CLK,WREN :IN STD_LOGIC;
--       A : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
--       DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
--       Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
-- END ;
-- ARCHITECTURE bhv OF RAM78 IS
-- TYPE G_ARRAY IS ARRAY(0 TO 127) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
-- SIGNAL MEM :G_ARRAY;
-- BEGIN
--   PROCESS (CLK)
--   BEGIN
--     IF RISING_EDGE(CLK) THEN
--       IF WREN='1' THEN
--         MEM(CONV_INTEGER(A))<= DIN;
--       END IF; END IF;
--     IF (FALLING_EDGE(CLK)) THEN Q<=MEM(CONV_INTEGER(A));
--     END IF;
--   END PROCESS;
-- END BHV;

```

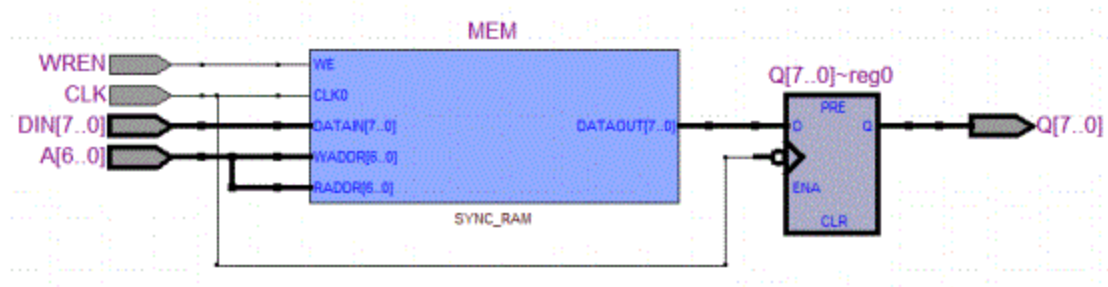
图五 存储器 VHDL 程序



图六 仿真结果



图七 波形仿真结果



图八 存储器 RTL 电路图



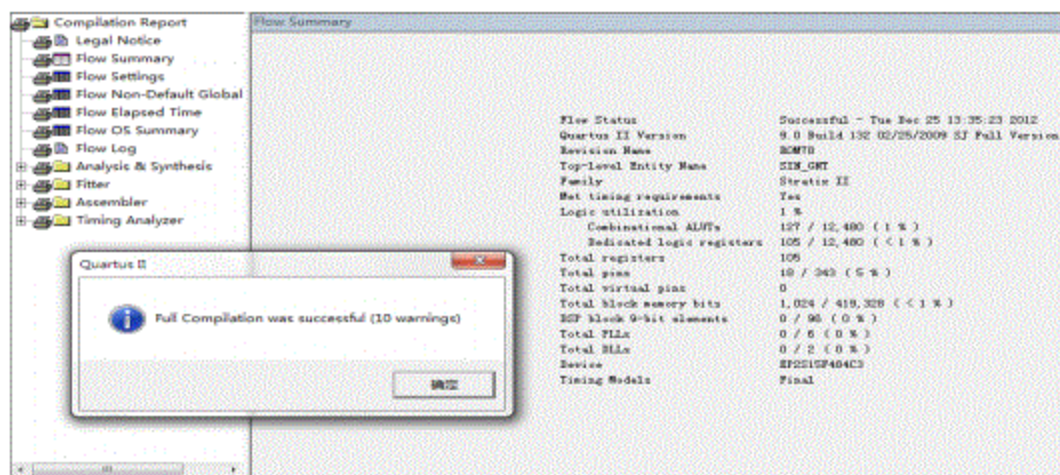
图九 存储器 Symbol

## 实验十二、LPM\_ROM 的定制和使

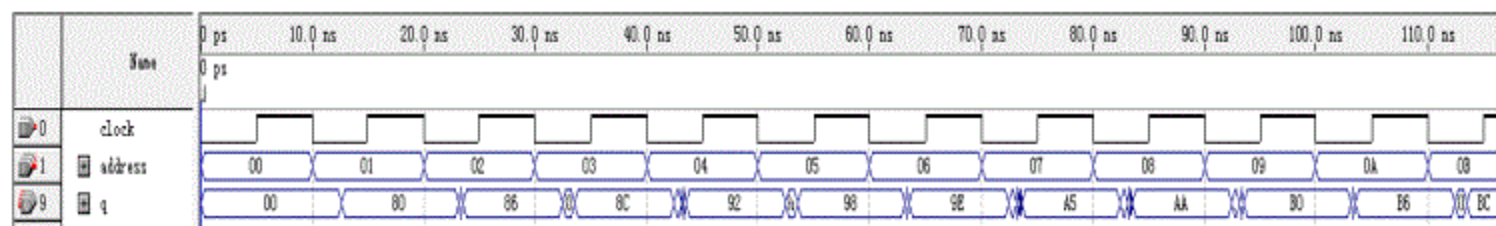
### 一、正弦信号发生器源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SIN_GNT IS
    PORT (RST,CLK,EN:IN STD_LOGIC;
          AR: OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
          Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END;
ARCHITECTURE ONE OF SIN_GNT IS
    COMPONENT ROM78
    PORT (address :IN STD_LOGIC_VECTOR (6 DOWNTO 0);
          clock :IN STD_LOGIC;
          q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
    END COMPONENT;
    SIGNAL Q1 : STD_LOGIC_VECTOR(6 DOWNTO 0);
    BEGIN
    PROCESS (CLK,RST,EN) BEGIN
    IF (RST='0') THEN Q1 <="0000000";
    ELSIF CLK'EVENT AND CLK ='1' THEN
    IF (EN='1') THEN Q1<=Q1+1; END IF ;END IF;
    END PROCESS;
    AR<=Q1;
    u1 : ROM78 PORT MAP(address=>Q1,q => Q,clock=>CLK);
END;
```

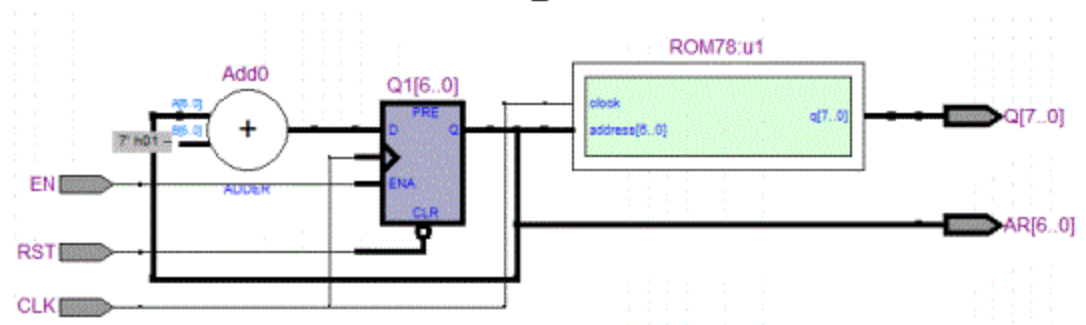
图一 正弦信号发生器源程序



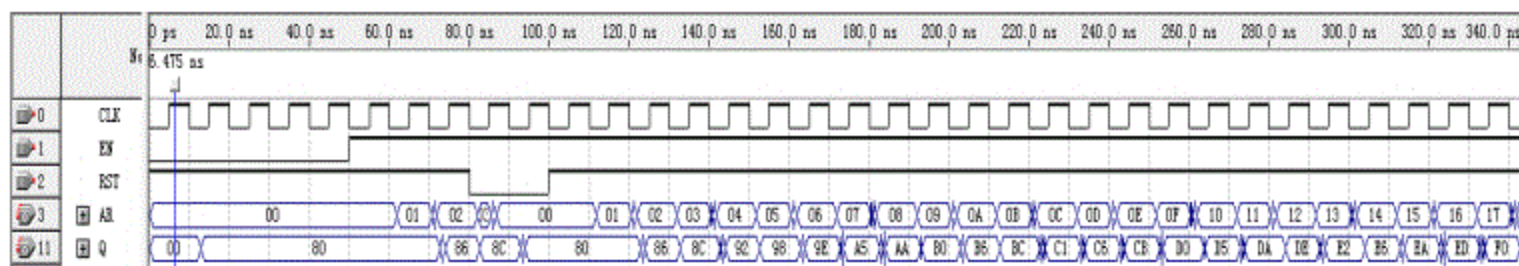
图二 仿真结果



图三 LPM\_ROM 仿真测试

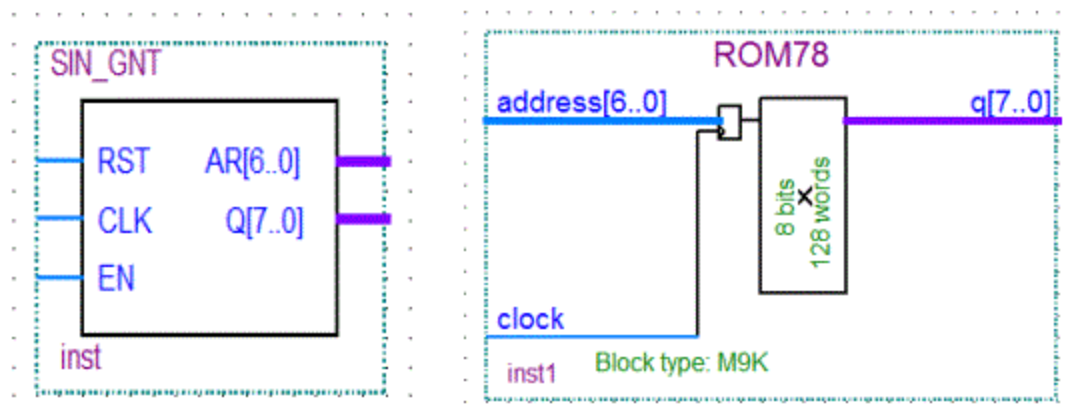


图四 正弦信号发生器 RTL 电路图



图五 正弦信号发生器仿真波形

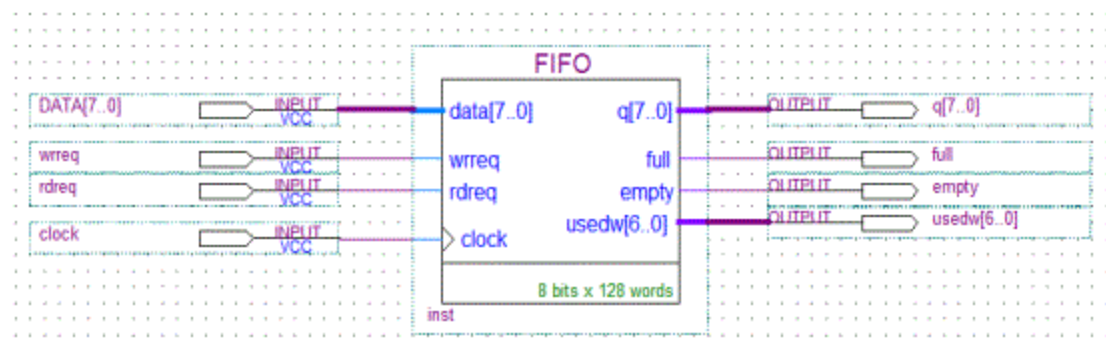
由波形可见，随着每一个时钟上升沿的到来，输出端口将正弦波数据依次输出，输出的数据与加载数据相符。



图六 正弦信号发生器 Symbol

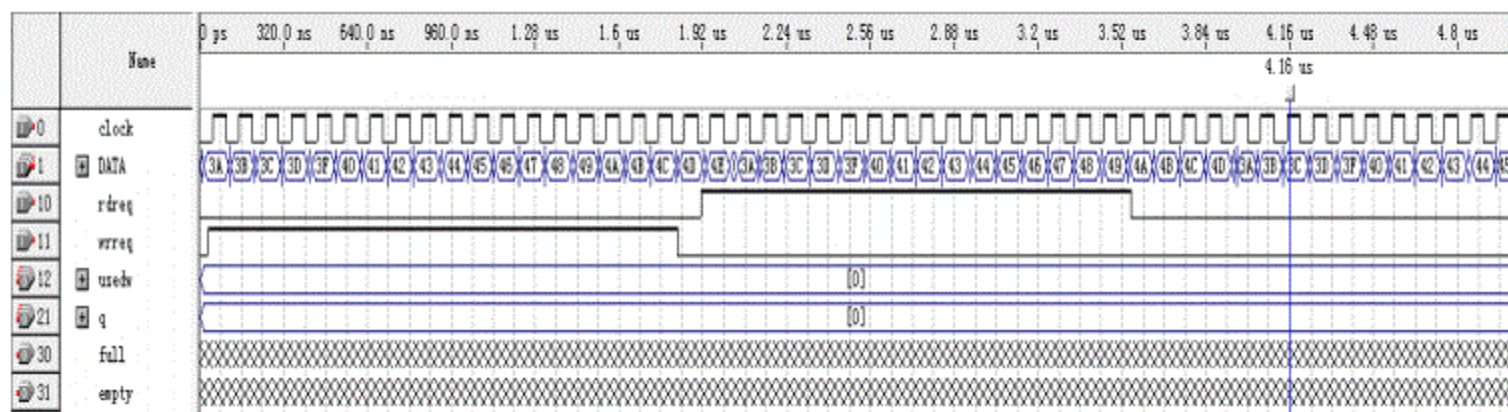
## 实验十三、FIFO 定制

### 一、FIFO 电路原理图



图一 FIFO 电路原理图

此FIFO的数据位宽为8,深度为256。其中data[7..0]为数据输入口; q[7..0]为数据输出口; wrreq 和 rdreq 分别为数据写入和读出请求信号, 高电平有效; aclr 为异步清零; full 为存储数据溢出指示信号; empty 为 FIFO 空指示信号; usedw[7..0]为当前已使用地址数指示; 选择了速度优化方式。

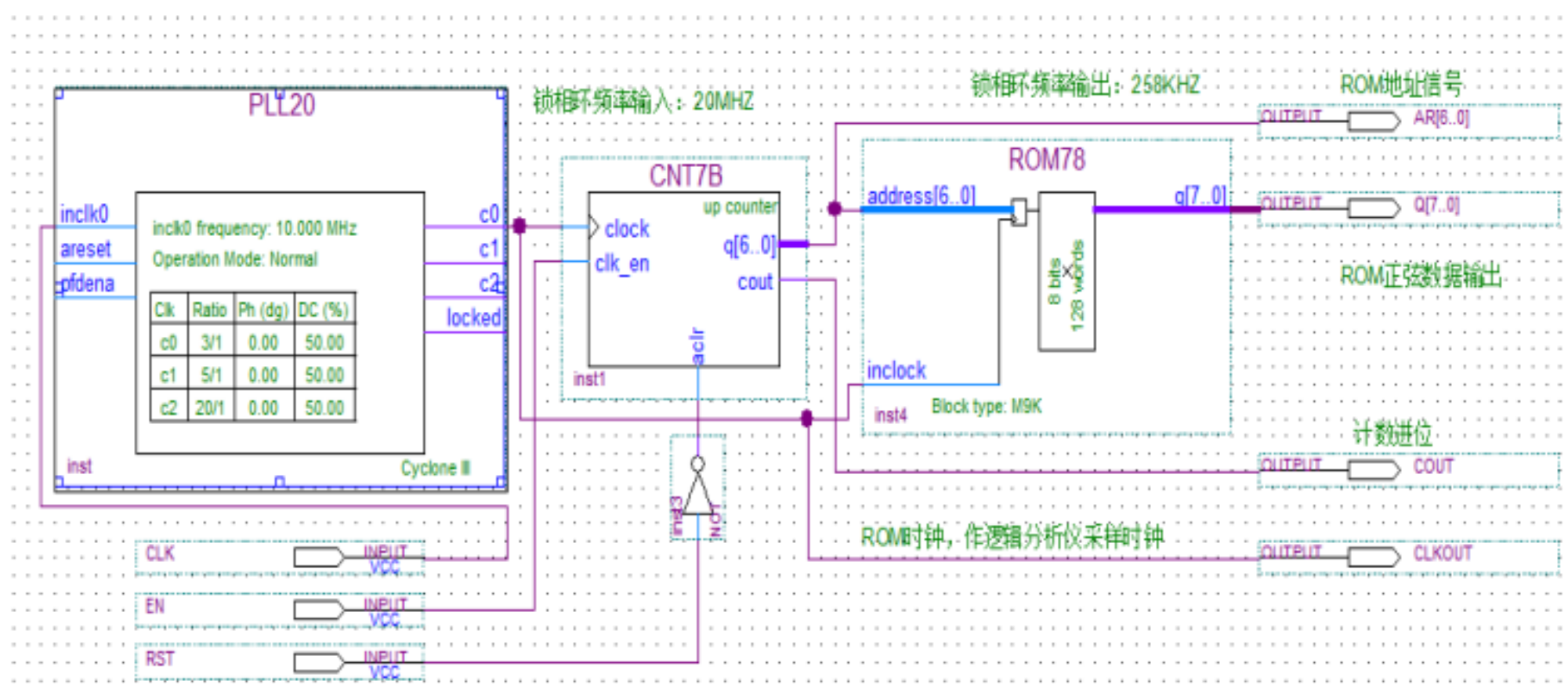


图二 FIFO 的仿真波形

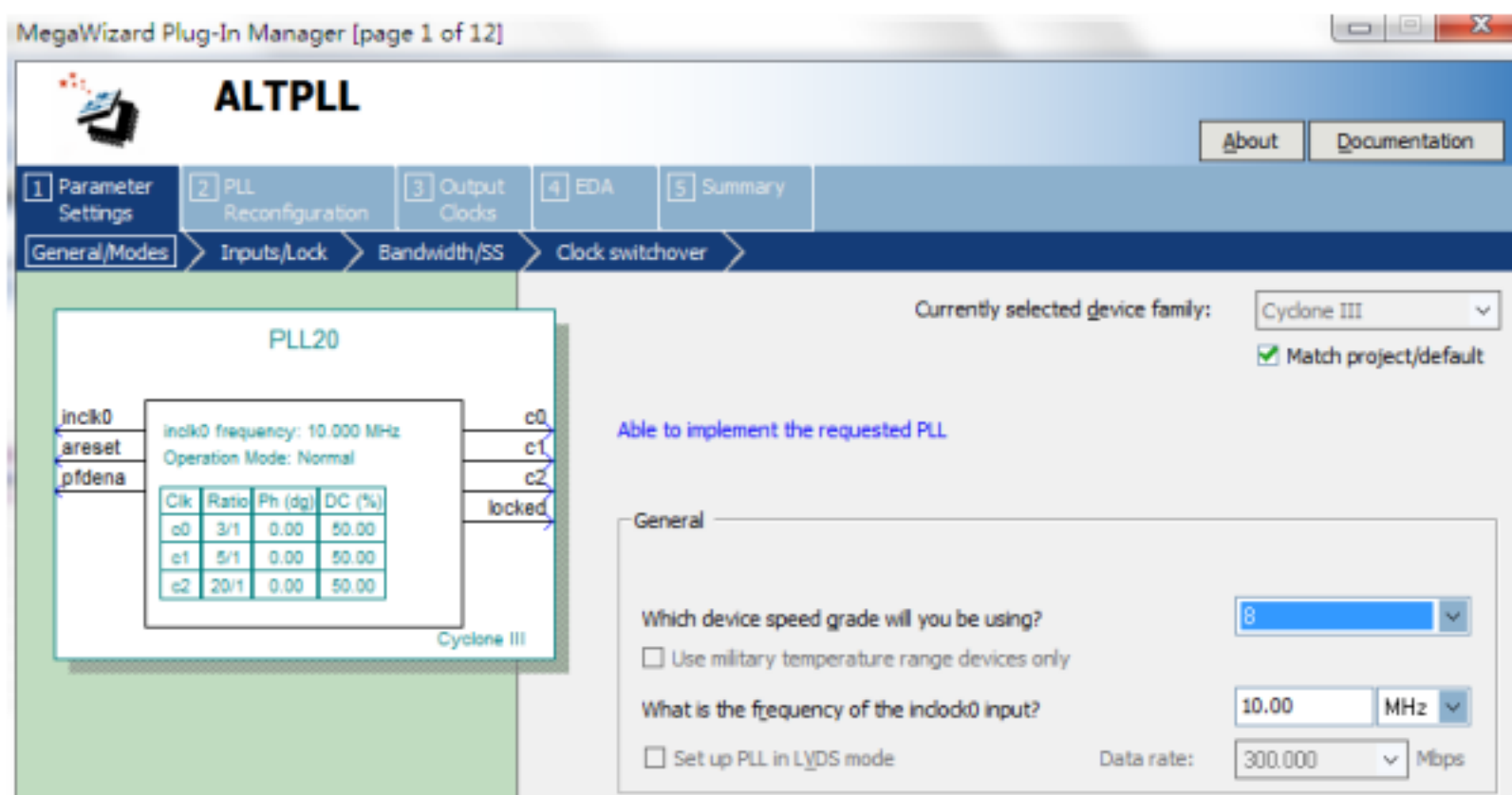
从波形中可以看出, 当写入请求 wrreq 为高电平时, 在 clock 的每一个上升沿将 data 上的数据写入 FIFO 中; 而在 wrreq 为低电平和读出请求 rdreq 为高电平时, clock 的每一个上升沿, 按照先进先出的顺序将 FIFO 中存入的数据读出, 在这个过程中, usedw[7..0]的数据也随之变化。

## 实验十四、LPM 嵌入式锁相环调用

### 一、采用嵌入式锁相环作时钟的正弦信号发生器电路图



图一 电路原理图



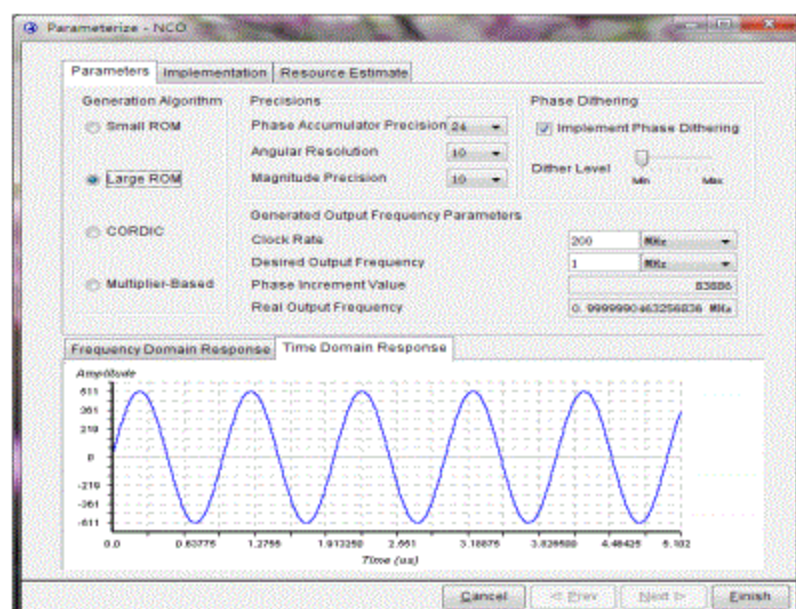
图二 选择输入参考时钟 inclk0 为 10MHZ

## 实验十五、NCO 核数控振荡器使用方法

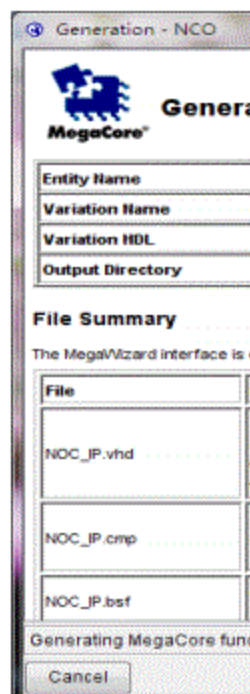
### 一、NCO 核数控振荡器使用方法



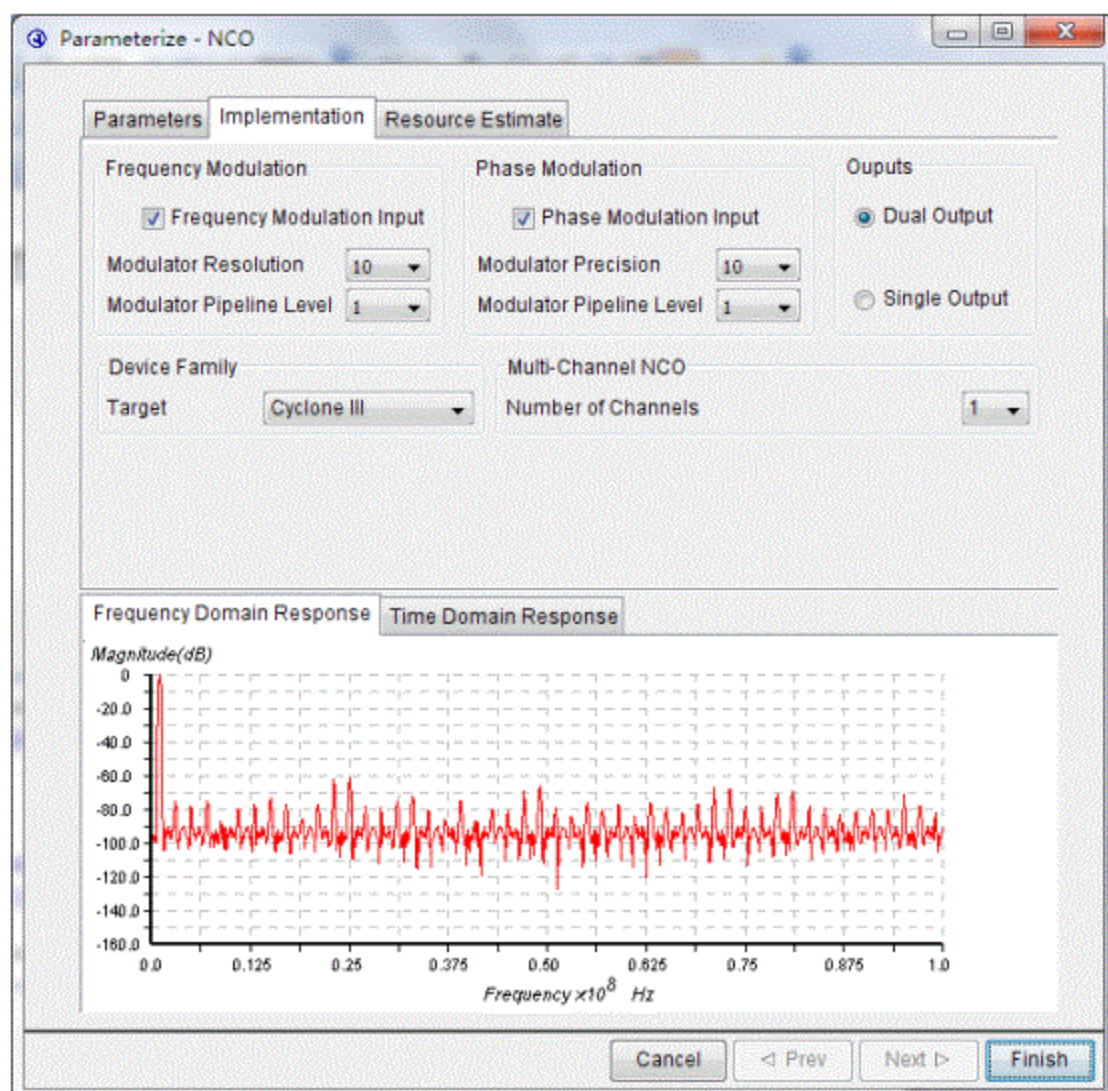
图一 开始进入 Core 文件生成选择窗口



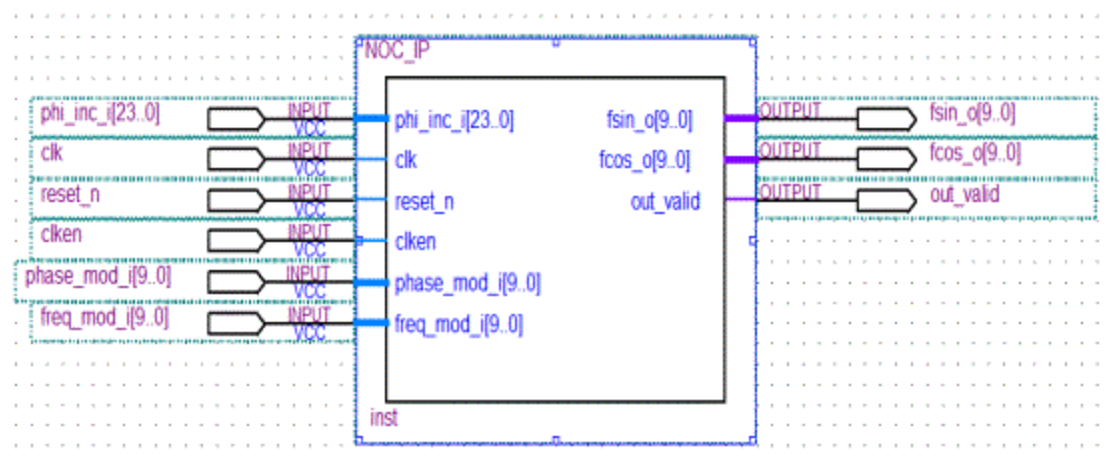
图二 设置 NCO 参数



图三 完成 NCO 参数设置并生成设计文件后的信息窗口



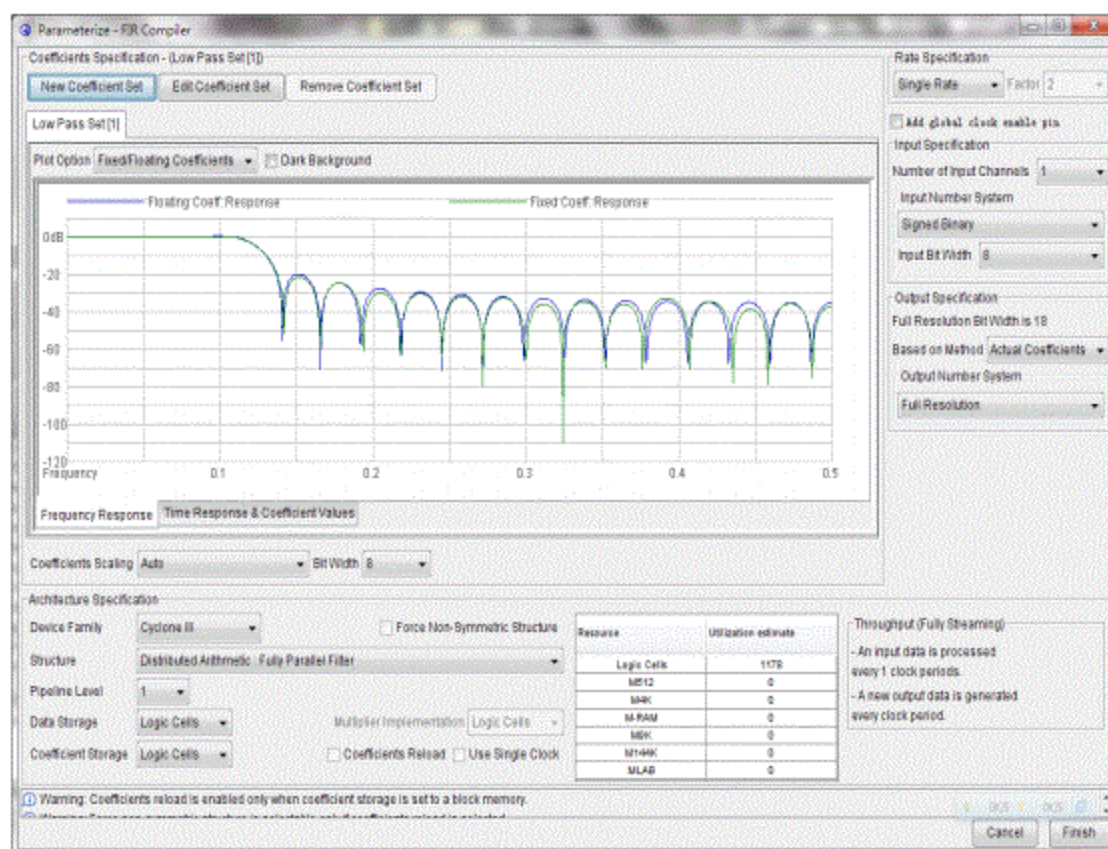
图四 设置 NCO 参数



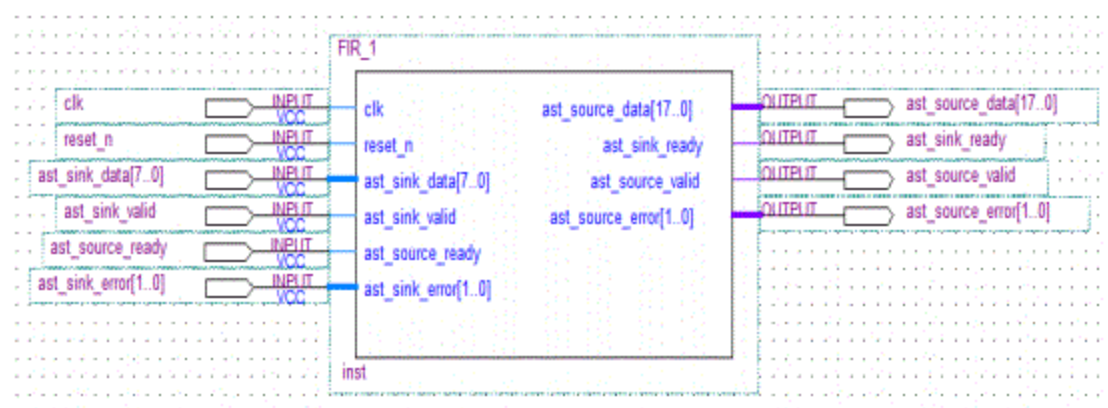
图五 测试 NCO 的电路

## 实验十六、使用 IP CORE 设计 FIR 滤波器

### 一、使用 IP Core 设计 FIR 滤波器



图一 FIR 滤波器系数确定



图二 测试电路图

## 实验十七、数字时钟

### 一、文本设计输入 (VHDL) 法

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

entity count24 is
    port(en1,en2,res,clk:in std_logic;
         ca:out std_logic;
         a,b:out std_logic_vector(3 downto 0));
end count24;

architecture rtl of count24 is
    signal aout,bout:std_logic_vector(3 downto 0);
    signal cout,cl:std_logic;
begin
    cl <= clk when en2 = '0' else
        en1;

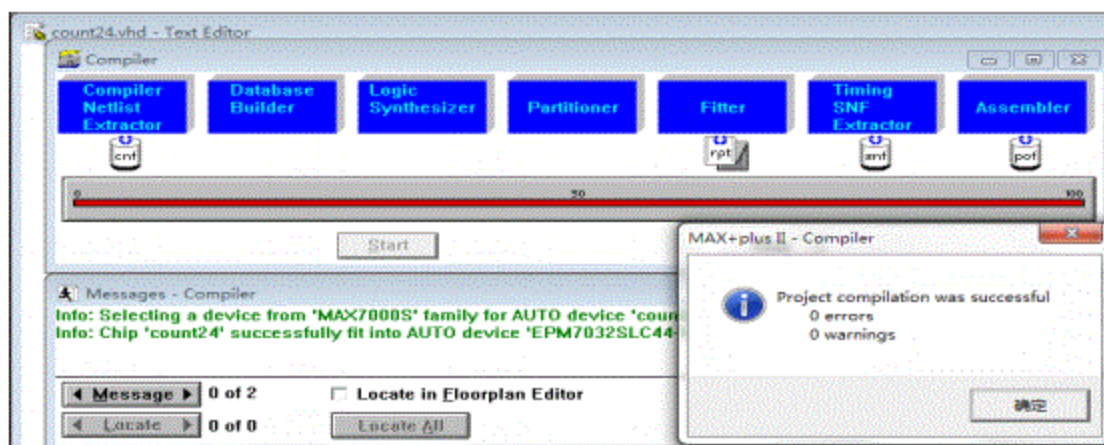
    process(en1,en2,cl,res)
    begin
        if(res='0')then
            aout<="0000";
            bout<="0000";
            cout<='0';
        elsif(cl'event and cl='1')then
            if(bout>1)then
                if(aout>2)then
                    aout<="0000";
                    bout<="0000";
                    cout<='1';
                else
                    aout<=aout+1;
                end if;
            else
                cout<='0';
            end if;
        end if;
    end process;
end architecture;
```

```

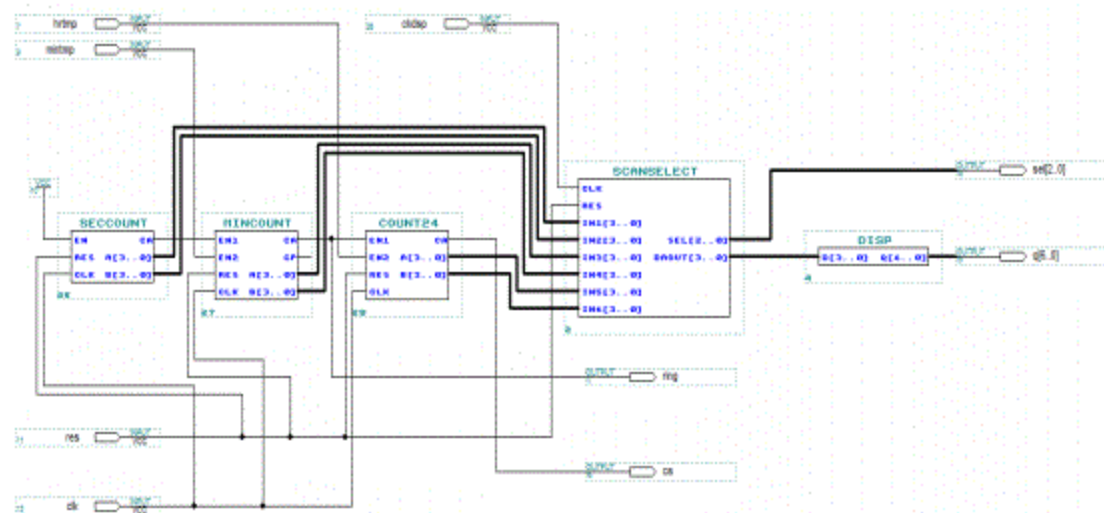
        if(aout=9) then
            aout<="0000";
            bout<=bout+1;
        else
            aout<=aout+1;
        end if;
    end if;
end if;
end process;

a<=aout;
b<=bout;
ca<=cout;
end rtl;

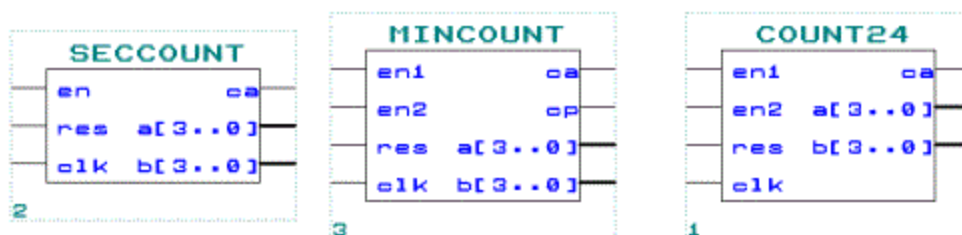
```



图一 仿真结果



图二 数字时钟电路原理图



图三 数字时钟实体模块

## 二、数字时钟 seccount 模块——秒计时

```

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

entity seccount is
    port(en,res,clk:in std_logic;
          ca:out std_logic;
          a,b:out std_logic_vector(3 downto 0));
end seccount;

architecture rtl of seccount is
    signal aout,bout:std_logic_vector(3 downto 0);
    signal cout:std_logic;
begin
    process(en,clk,res)
    begin
        if(res='0') then
            aout<="0000";
            bout<="0000";
            cout<='0';
        elsif(clk'event and clk='1') then
            if(en='1') then
                if(bout>4) then
                    if(aout>9) then
                        aout<="0000";
                        bout<="0000";
                        cout<='1';
                    else
                        aout<=aout+1;
                    end if;
                else
                    if(aout=9) then

```

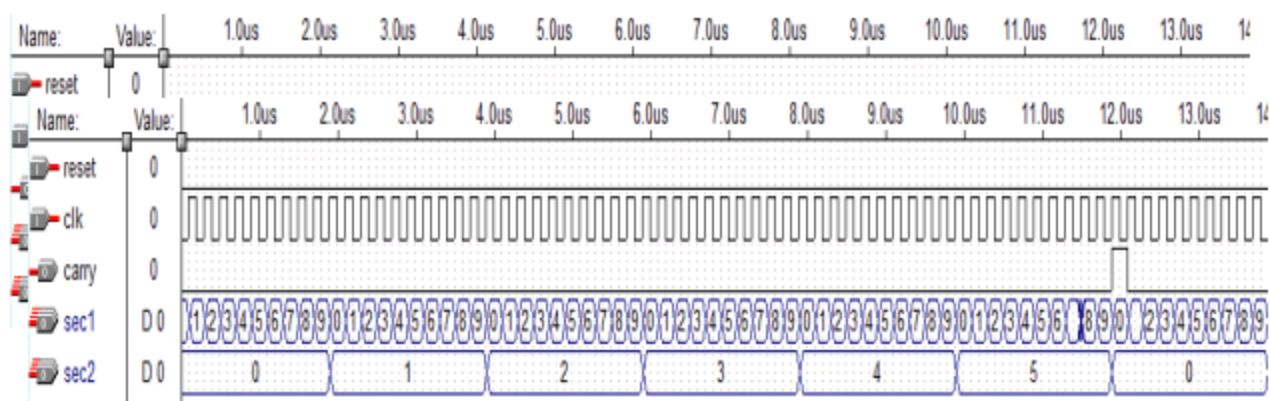
```

        aout<="0000";
        bout<=bout+1;
    else
        aout<=aout+1;
        cout<='0';
    end if;
end if;
end if;
end if;
end process;

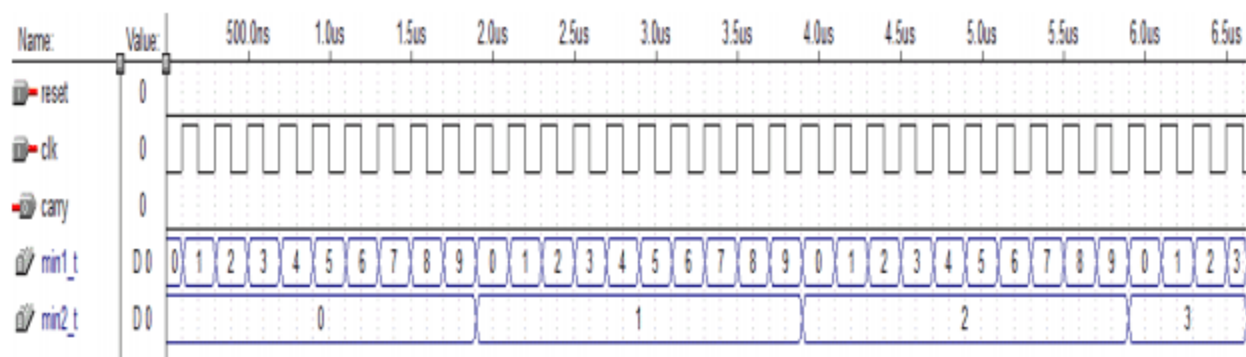
a<=aout;
b<=bout;
ca<=cout;
end rtl;

```

图四 全加器数字时钟 seccount 模块一秒计时



图五 秒计数波形仿真图



图六 分计数波形仿真图

## 实验十八、交通灯

### 一、文本设计输入（VHDL）法

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;--加载库文件

entity traffic_ctrl is--定义实体
    generic (
        green1_cnt:integer:=25;--定义主通道绿灯亮的时间，这里为25秒
        yellow1_cnt:integer:=5; --定义主通道黄灯亮的时间，这里为5秒
        green2_cnt:integer:=15;--定义支路绿灯亮的时间，这里为15秒
        yellow2_cnt:integer:=5 --定义支路黄灯亮的时间，这里为5秒
    );
    port(
        clk:      in std_logic;--时钟信号|
        rst:      in std_logic;--复位信号
        lgt1_red:  out std_logic;--主通道红灯控制信号
        lgt1_yellow: out std_logic;--主通道黄灯控制信号
        lgt1_green: out std_logic;--主通道绿灯控制信号
        lgt2_red:  out std_logic;--支路红灯控制信号
        lgt2_yellow: out std_logic;--支路黄灯控制信号
        lgt2_green: out std_logic;--支路绿灯控制信号
    );
end entity traffic_ctrl;

architecture rtl of traffic_ctrl is--定义结构体
    type states is (st0,st1,st2,st3);--定义控制器各种状态
    signal state:states:=st0;--初始化状态
    signal cnt : integer range 0 to 30:=1;--定义计数器
    signal cnt_enb :std_logic:='0';--初始化计数器使能信号
begin
    process(clk,rst)
    begin
        if(rst='1') then--复位信号为高则执行复位操作
            state<=st0;
            cnt<=1;
        elsif(rising_edge(clk)) then--时钟上升沿
            if(cnt_enb='1') then--计数器计数
                cnt<=cnt+1;
            else
                cnt<=1;
            end if;
        end if;
    end process;
end architecture;
```

```

end if;
case state is
  when st0=>--主通道绿灯亮了一段时间时转换状态st1
    if(cnt=green1_cnt) then
      state<=st1;
    else
      state<=st0;
    end if;
  when st1=>--主通道黄灯亮了一段时间时转换状态st2
    if(cnt=yellow1_cnt) then
      state<=st2;
    else
      state<=st1;
    end if;
  when st2=>--支路绿灯亮了一段时间时转换状态st3
    if(cnt=green2_cnt) then
      state<=st3;
    else
      state<=st2;
    end if;
  when st3=>--支路黄灯亮了一段时间时转换状态st0
    if(cnt=yellow2_cnt) then
      state<=st0;
    else
      state<=st3;
    end if;
end case;
end if;
end process;
process(state)

```

begin

case state is

when st0=>--St0表示主路绿灯亮、支路红灯亮

```

  lgt1_red<='0';
  lgt1_yellow<='0';
  lgt1_green<='1';
  lgt2_red<='1';
  lgt2_yellow<='0';
  lgt2_green<='0';
  cnt_enb<='1';
  if(cnt=green1_cnt) then
    cnt_enb<='0';
  end if;

```

when st1=>--St1表示主路黄灯亮、支路红灯亮

```

  lgt1_red<='0';
  lgt1_yellow<='1';
  lgt1_green<='0';
  lgt2_red<='1';
  lgt2_yellow<='0';

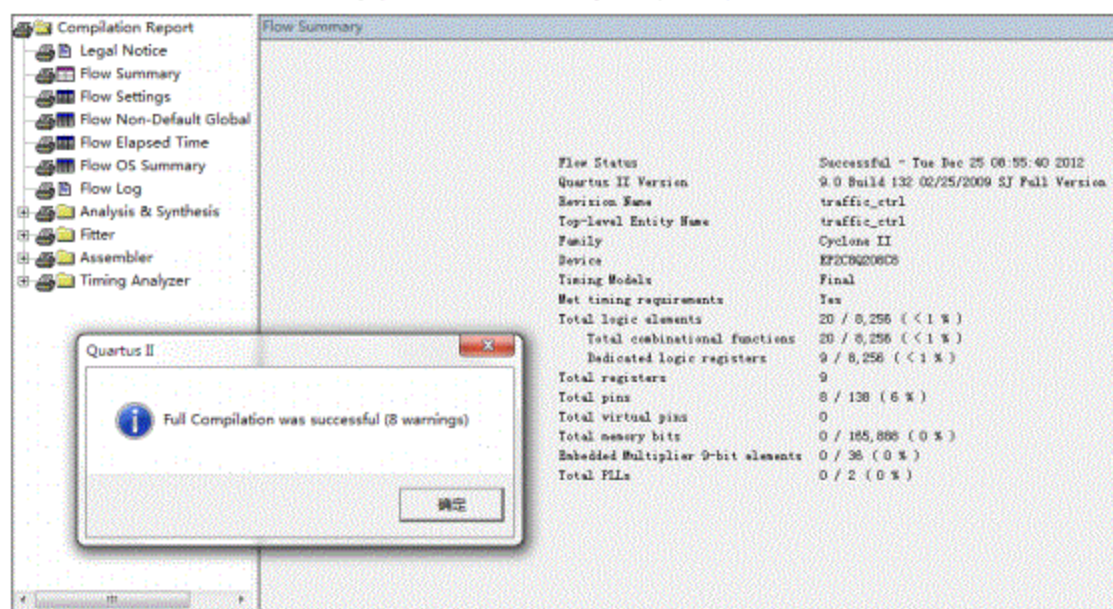
```

```

lgt2_green<='0';
cnt_enb<='1';
if(cnt=yellow1_cnt) then
    cnt_enb<='0';
end if;
when st2=>--St2表示主路红灯亮、支路绿灯亮
    lgt1_red<='1';
    lgt1_yellow<='0';
    lgt1_green<='0';
    lgt2_red<='0';
    lgt2_yellow<='0';
    lgt2_green<='1';
    cnt_enb<='1';
    if(cnt=green2_cnt) then
        cnt_enb<='0';
    end if;
when st3=>--St3表示主路红灯亮、支路黄灯亮
    lgt1_red<='1';
    lgt1_yellow<='0';
    lgt1_green<='0';
    lgt2_red<='0';
    lgt2_yellow<='1';
    lgt2_green<='0';
    cnt_enb<='1';
    if(cnt=yellow2_cnt) then
        cnt_enb<='0';
    end if;
end case;
end process;
end rtl;

```

图一 交通灯程序文本设计输入



图二 仿真结果