# EDA 技术实用教程

## 第 11 章

## 优化和时序分析

# 11.1 资源优化

## 11.1.1 资源共享

【例11-1】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;
ENTITY multmux IS
    PORT (A0, A1, B  : IN  std_logic_vector(3 downto 0);
                sel  : IN  std_logic;
             Result  : OUT std_logic_vector(7 downto 0));
END multmux;
ARCHITECTURE rtl OF multmux IS
BEGIN
    process(A0,A1,B,sel)
    begin
        if(sel = '0') then   Result <= A0 * B;
                      else   Result <= A1 * B;
        end if;
    end process;
END rtl;
```

# 11.1 资源优化

## 11.1.1 资源共享

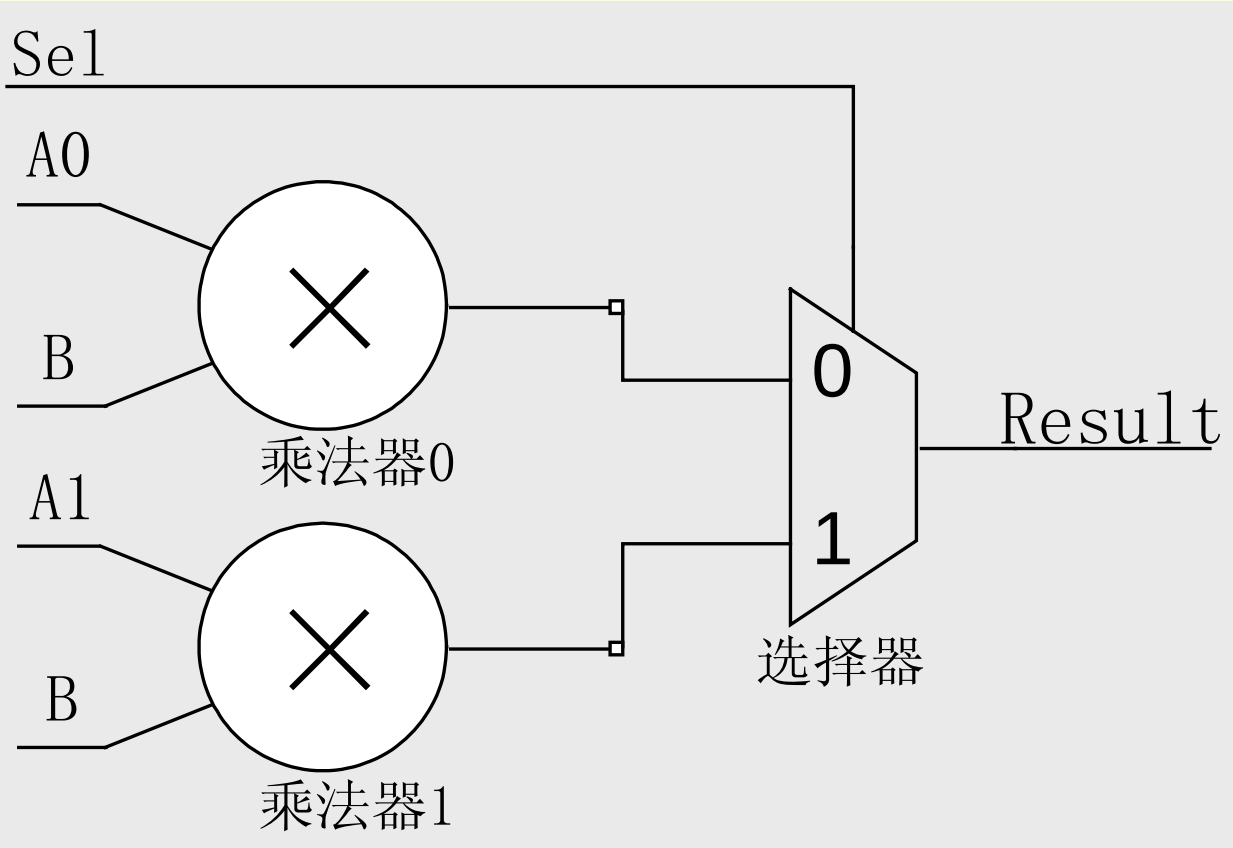



图11-1 先乘后选择的设计方法RTL结构

# 11.1 资源优化

## 11.1.1 资源共享

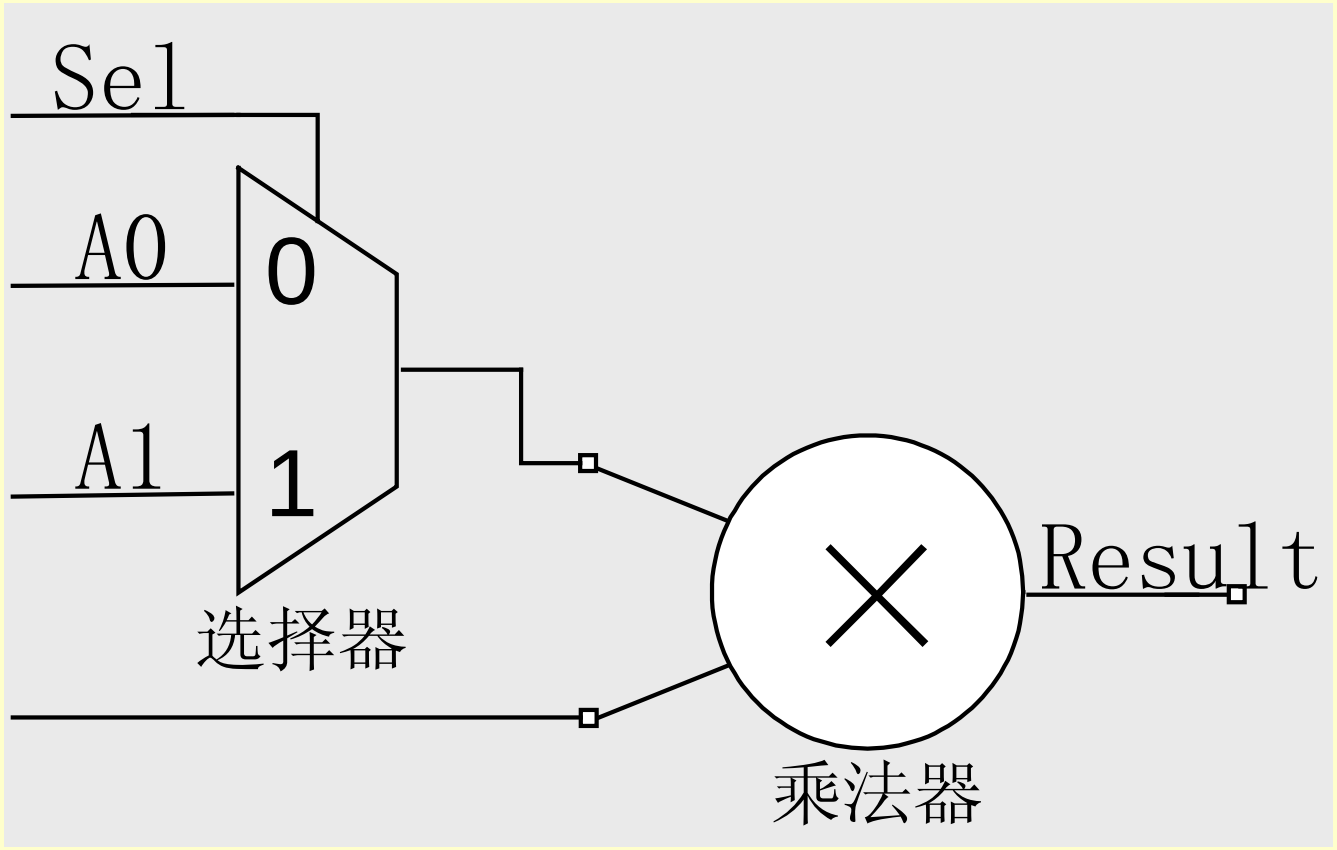



图11-2 先选择后乘设计方法RTL结构

# 11.1 资源优化

## 11.1.1 资源共享

【例11-2】

```
ARCHITECTURE rtl OF muxmult IS
    signal temp : std_logic_vector(3 downto 0);
BEGIN
    process(A0,A1,B,sel)
    begin
        if(sel = '0') then    temp <= A0; else  temp <= A1;
        end if;
        result <= temp * B;
    end process;
END rtl;
```

# 11.1 资源优化

## 11.1.1 资源共享

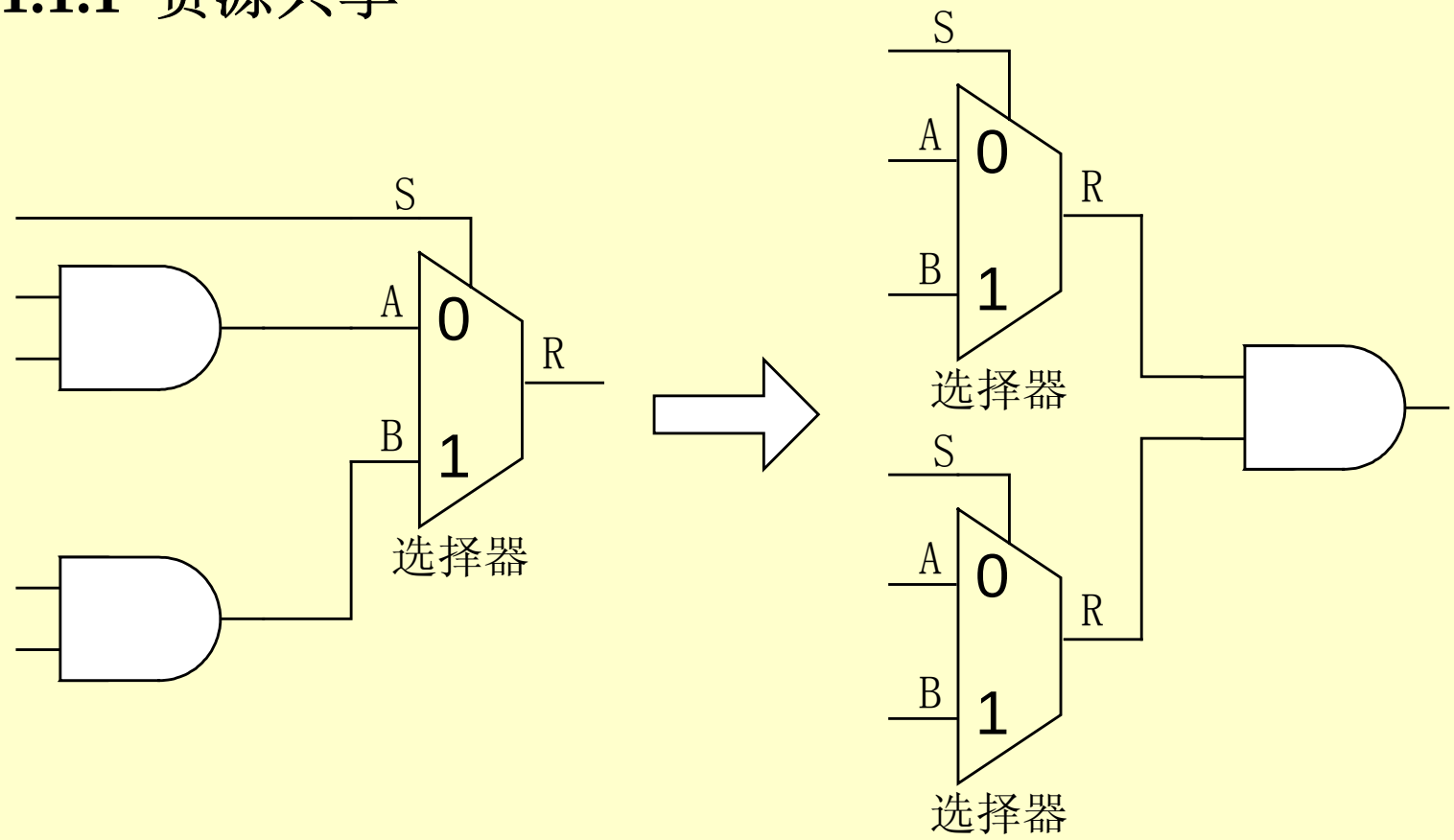



图11-3 资源共享反例

# 11.1 资源优化

## 11.1.2 逻辑优化

【例11-3】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY mult1 IS
    PORT(clk : in std_logic;
           ma : In std_logic_vector(11 downto 0);
           mc : out std_logic_vector(23 downto 0));
END mult1;
ARCHITECTURE rtl OF mult1 IS
    signal ta, tb : std_logic_vector(11 downto 0);
BEGIN
process(clk) begin
    if(clk'event and clk = '1') then
        ta <= ma;  tb <= "100110111001";   mc <= ta * tb;
    end if;
end process;
END rtl;
```

# 11.1 资源优化

## 11.1.2 逻辑优化

【例11-4】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY mult2 IS
    PORT(clk : in std_logic;
         ma : In std_logic_vector(11 downto 0);
         mc : out std_logic_vector(23 downto 0));
END mult2;
ARCHITECTURE rtl OF mult2 IS
    signal ta : std_logic_vector(11 downto 0);
    constant tb : std_logic_vector(11 downto 0) := "100110111001";
BEGIN
process(clk) begin
    if(clk'event and clk = '1') then  ta<=ma; mc<=ta * tb;
    end if;
end process;
END rtl;
```

# 11.1 资源优化

## 11.1.3 串行化

$$yout = a_0 \times b_0 + a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3$$

【例11-5】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY pmultadd IS
    PORT(clk : in std_logic;
         a0,a1,a2,a3 : in std_logic_vector(7 downto 0);
         b0,b1,b2,b3 : in std_logic_vector(7 downto 0);
         yout : out std_logic_vector(15 downto 0));
END pmultadd;
ARCHITECTURE p_arch OF pmultadd IS
BEGIN
process(clk) begin
    if(clk'event and clk = '1') then
        yout <= ((a0*b0)+(a1*b1))+((a2*b2)+(a3*b3)); end if;
end process;
END p_arch;
```

# 11.1 资源优化

## 11.1.3 串行化

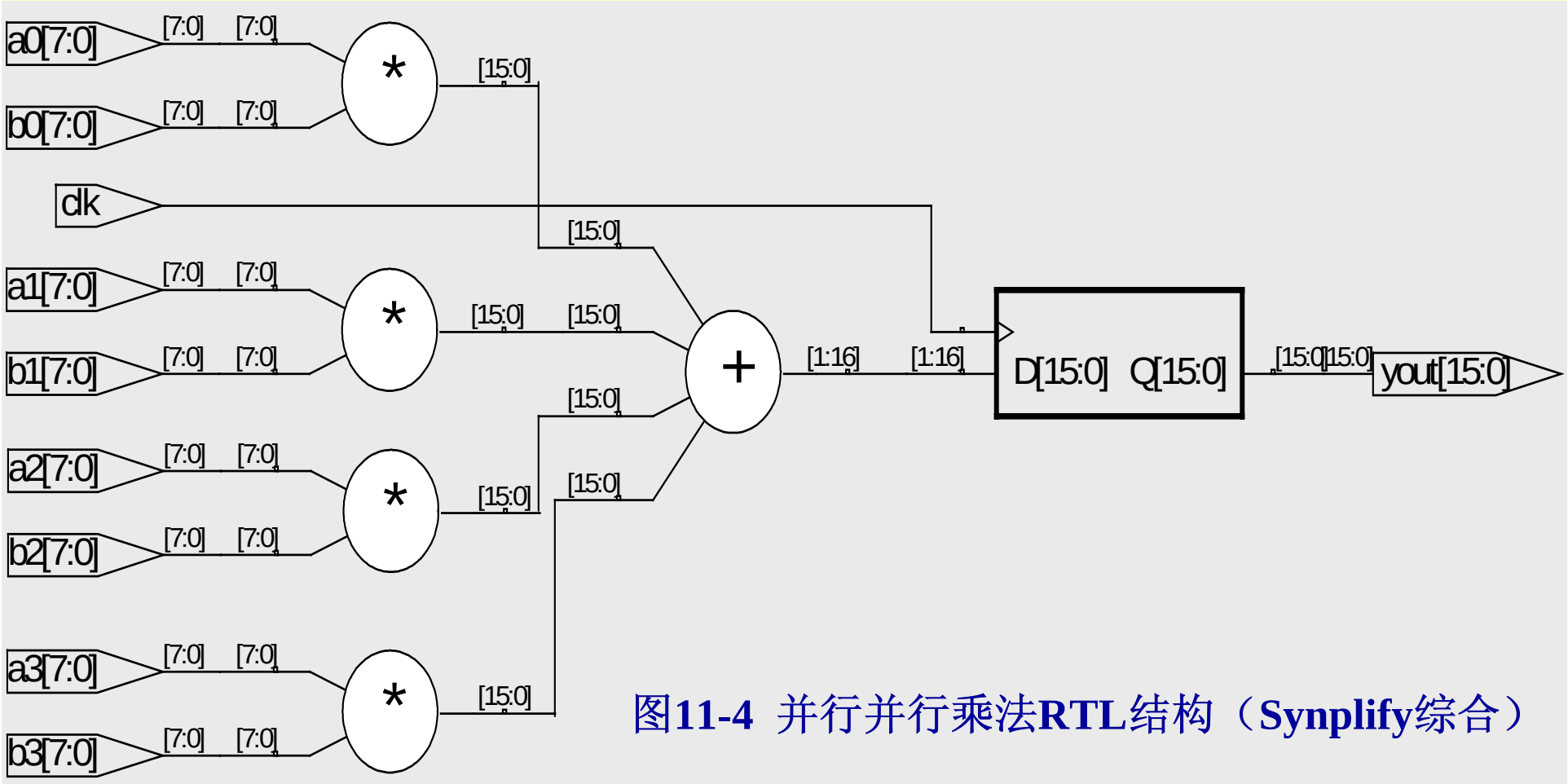



图11-4 并行并行乘法RTL结构（Synplify综合）

【例11-6】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY smultadd IS
    PORT(clk,  start : in std_logic;
         a0,a1,a2,a3 : In std_logic_vector(7 downto 0);
         b0,b1,b2,b3 : In std_logic_vector(7 downto 0);
         yout : out std_logic_vector(15 downto 0));
END smultadd;
ARCHITECTURE s_arch OF smultadd IS
    signal cnt : std_logic_vector(2 downto 0);
    signal tmpa, tmpb : std_logic_vector(7 downto 0);
    signal  tmp,  ytmp : std_logic_vector(15 downto 0);
  BEGIN
tmpa <= a0 when cnt = 0 else
         a1 when cnt = 1 else
         a2 when cnt = 2 else
         a3 when cnt = 3 else
         a0;
tmpb <= b0 when cnt = 0 else
         b1 when cnt = 1 else
         b2 when cnt = 2 else
         b3 when cnt = 3 else
         b0;
tmp <= tmpa * tmpb;
process(clk) begin
    if(clk'event and clk = '1') then
        if(start = '1') then  cnt <= "000"; ytmp <= (others=>'0');
        elsif (cnt<4) then  cnt <= cnt + 1; ytmp <= ytmp + tmp;
        elsif (cnt = 4) then  yout <= ytmp;
        end if;
    end if;
end process;
END s_arch;
```

# 11.2 速度优化

## 11.2.1 流水线设计

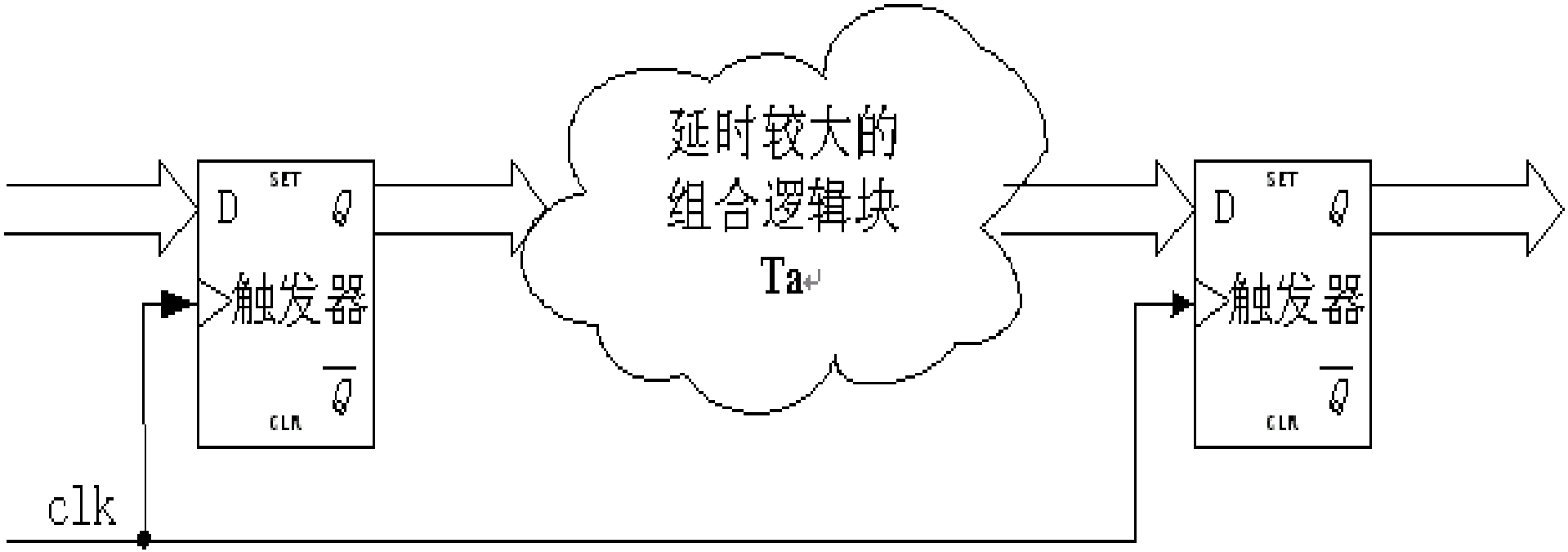



图11-5 未使用流水线

# 11.2 速度优化

## 11.2.1 流水线设计

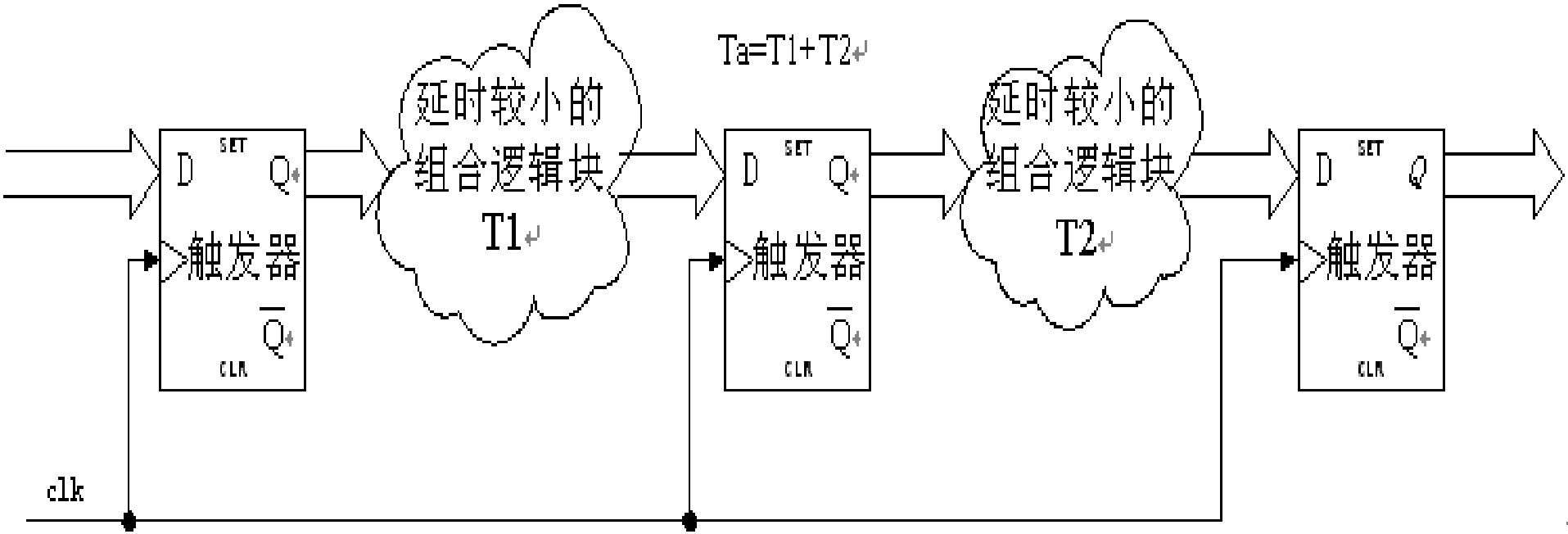



图11-6 使用流水线

# 11.2 速度优化

## 11.2.1 流水线设计


图11-7 流水线工作图示

【例11-7】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY adder4 IS
    PORT(clk : in std_logic;
         a0,a1,a2,a3 : in std_logic_vector(7 downto 0);
         yout : out std_logic_vector(9 downto 0));
END adder4;
ARCHITECTURE normal_arch OF adder4 IS
    signal t0,t1,t2,t3 : std_logic_vector(7 downto 0);
    signal addtmp0,addtmp1 : std_logic_vector(8 downto 0);
BEGIN
process(clk) begin
    if(clk'event and clk='1') then
        t0 <= a0;  t1 <= a1;  t2 <= a2;   t3 <= a3;
    end if;
end process;
addtmp0 <= '0'&t0 + t1;
addtmp1 <= '0'&t2 + t3;
process(clk) begin
    if(clk'event and clk = '1') then yout <= '0'&addtmp0 + addtmp1;
    end if;
end process;
END normal_arch;
```

【例11-8】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY pipeadd IS
    PORT(clk : in std_logic;
         a0,a1,a2,a3 : in std_logic_vector(7 downto 0);
         yout : out std_logic_vector(9 downto 0));
END pipeadd;
ARCHITECTURE pipelining_arch OF pipeadd IS
    signal t0,t1,t2,t3 : std_logic_vector(7 downto 0);
    signal addtmp0,addtmp1 : std_logic_vector(8 downto 0);
BEGIN
process(clk) begin
    if(clk'event and clk='1') then
        t0 <= a0; t1 <= a1;  t2 <= a2;  t3 <= a3;
    end if;
end process;
process(clk) begin
    if(clk'event and clk = '1') then
        addtmp0 <= '0'&t0 + t1;
        addtmp1 <= '0'&t2 + t3;
yout <= '0'&addtmp0 + addtmp1;
    end if;
end process;
END pipelining_arch;
```

# 11.2 速度优化

## 11.2.2 寄存器配平

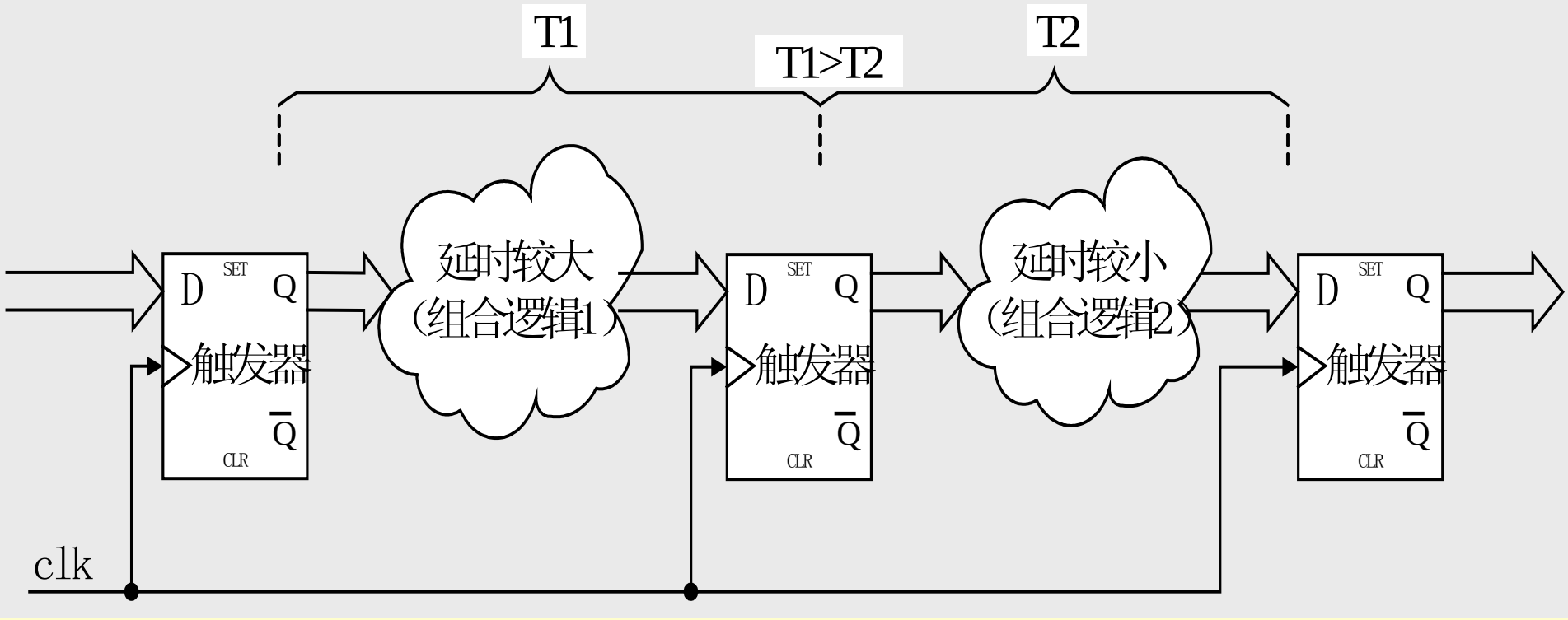



图11-8 不合理的结构

# 11.2 速度优化

## 11.2.2 寄存器配平

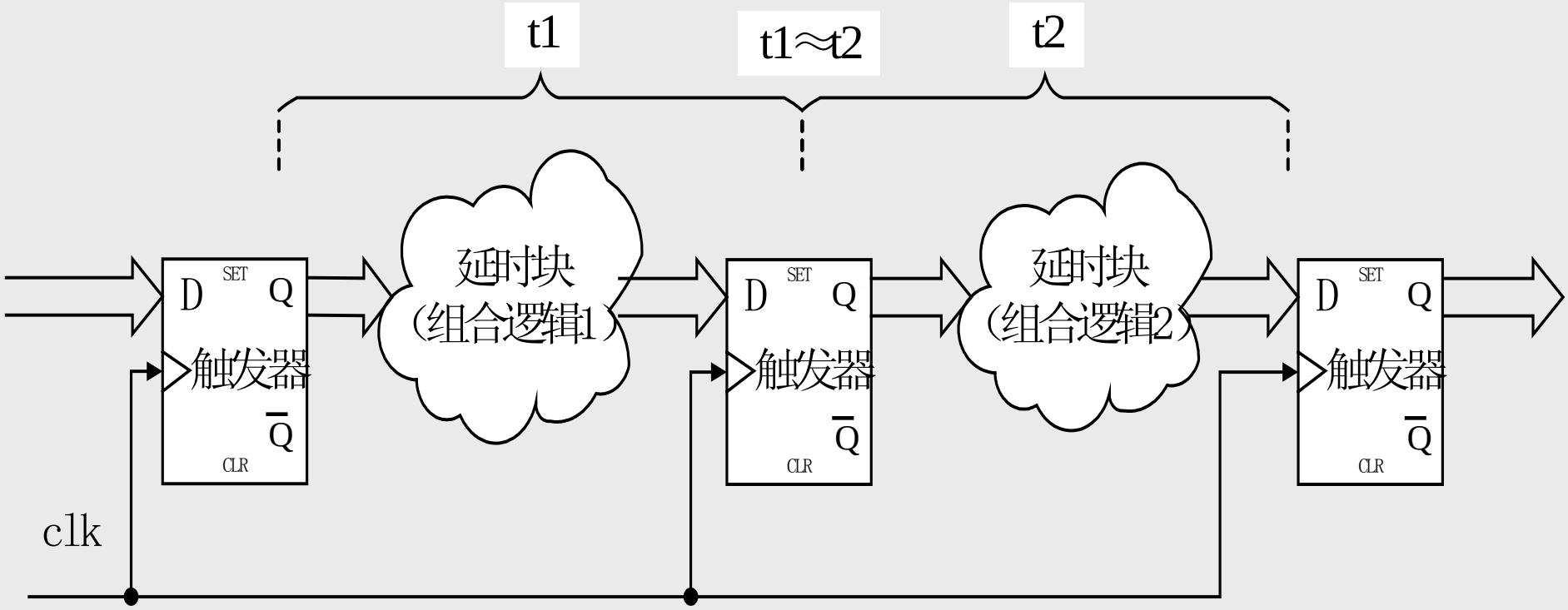



图11-9 寄存器配平的结构

# 11.2 速度优化

## 11.2.3 关键路径法

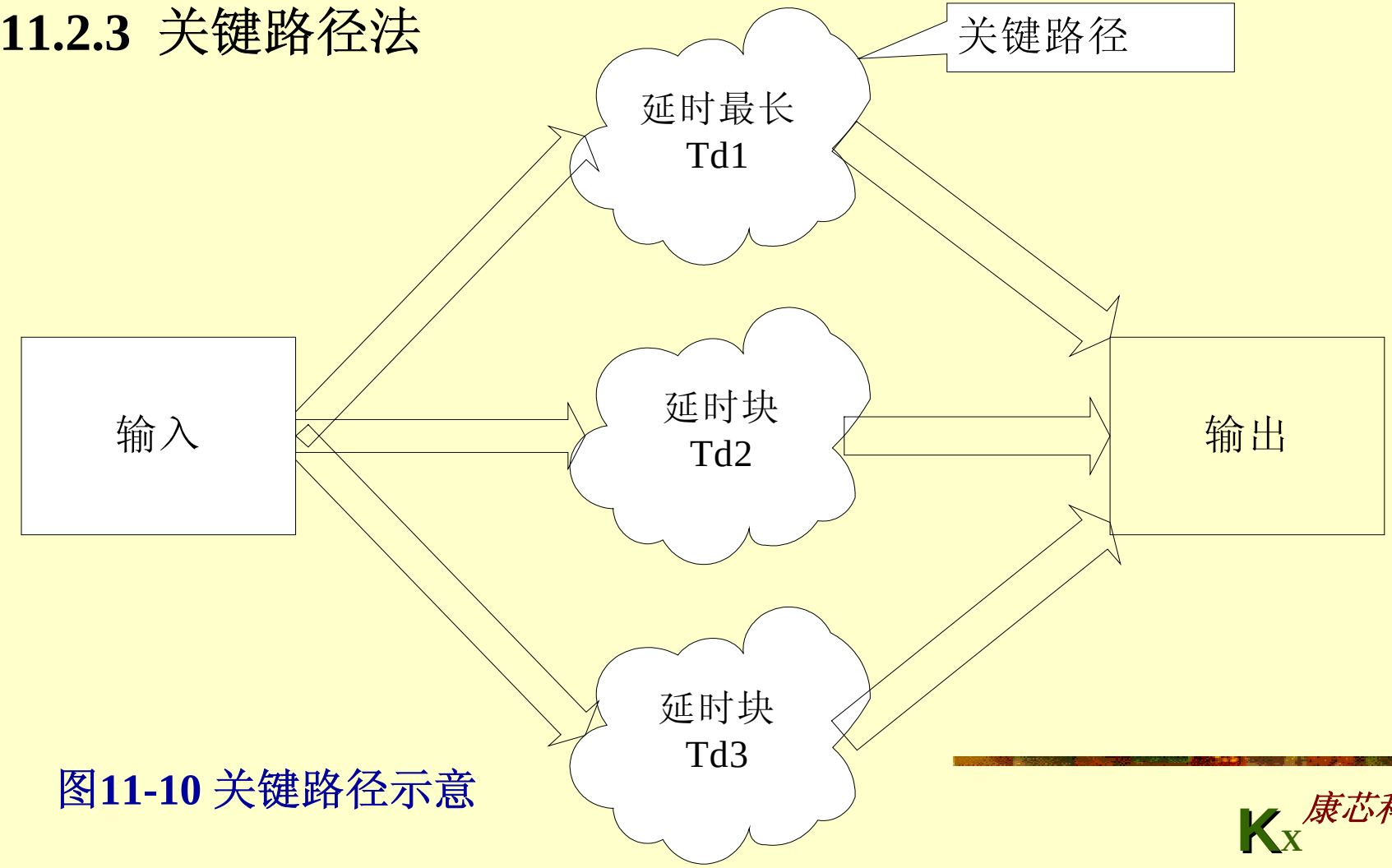



图11-10 关键路径示意

# 11.3 优化设置与时序分析

**11.3.1 Settings设置**

**11.3.2 HDL版本设置及Analysis & Synthesis功能**

**11.3.3 Analysis & Synthesis的优化设置**

**11.3.4 适配器Fitter设置**

# 11.3 优化设置与时序分析

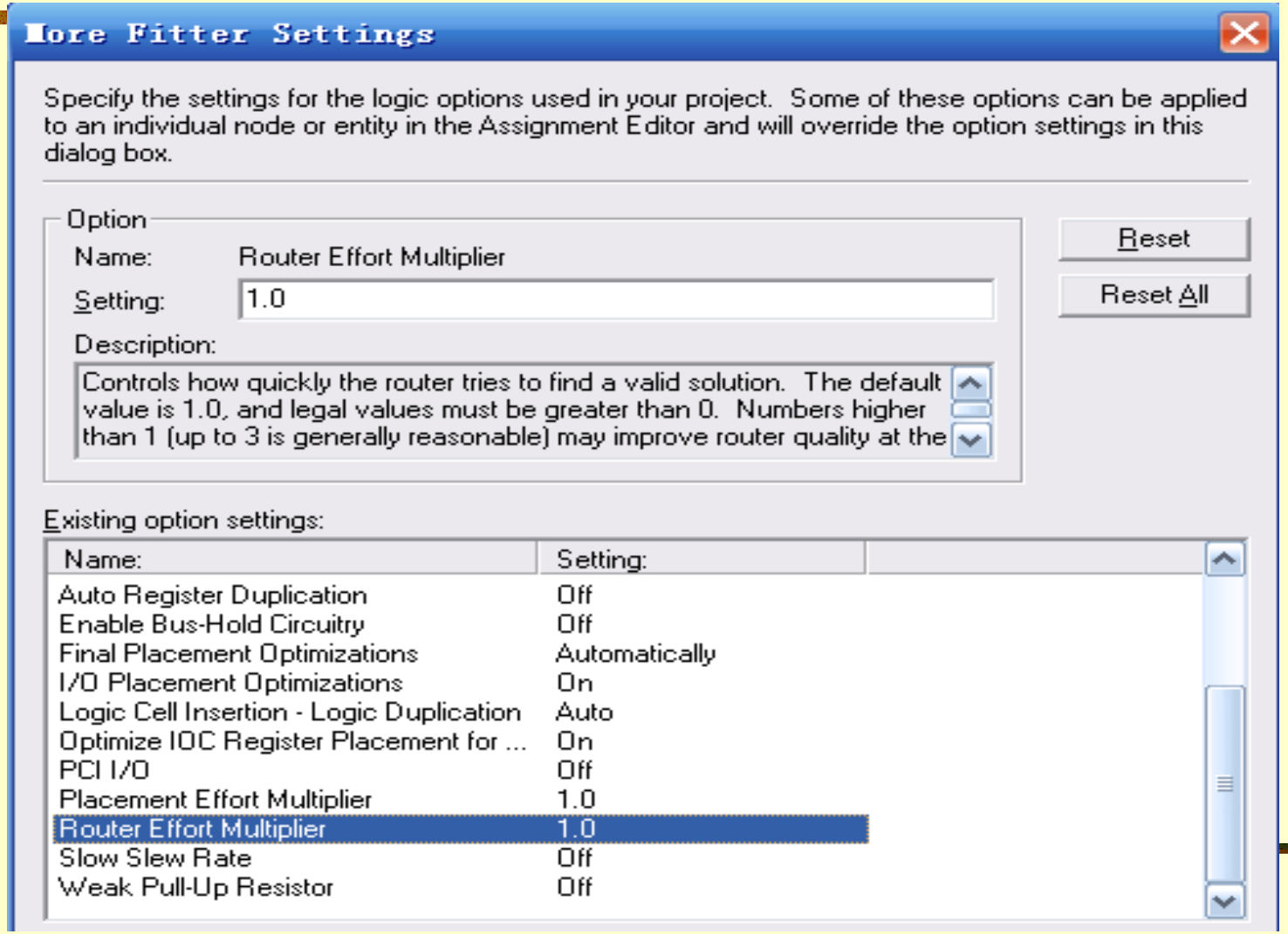



图9-11 布线倍增器优化程度指数选择

# 11.3 优化设置与时序分析

## 11.3.5 增量布局布线控制设置

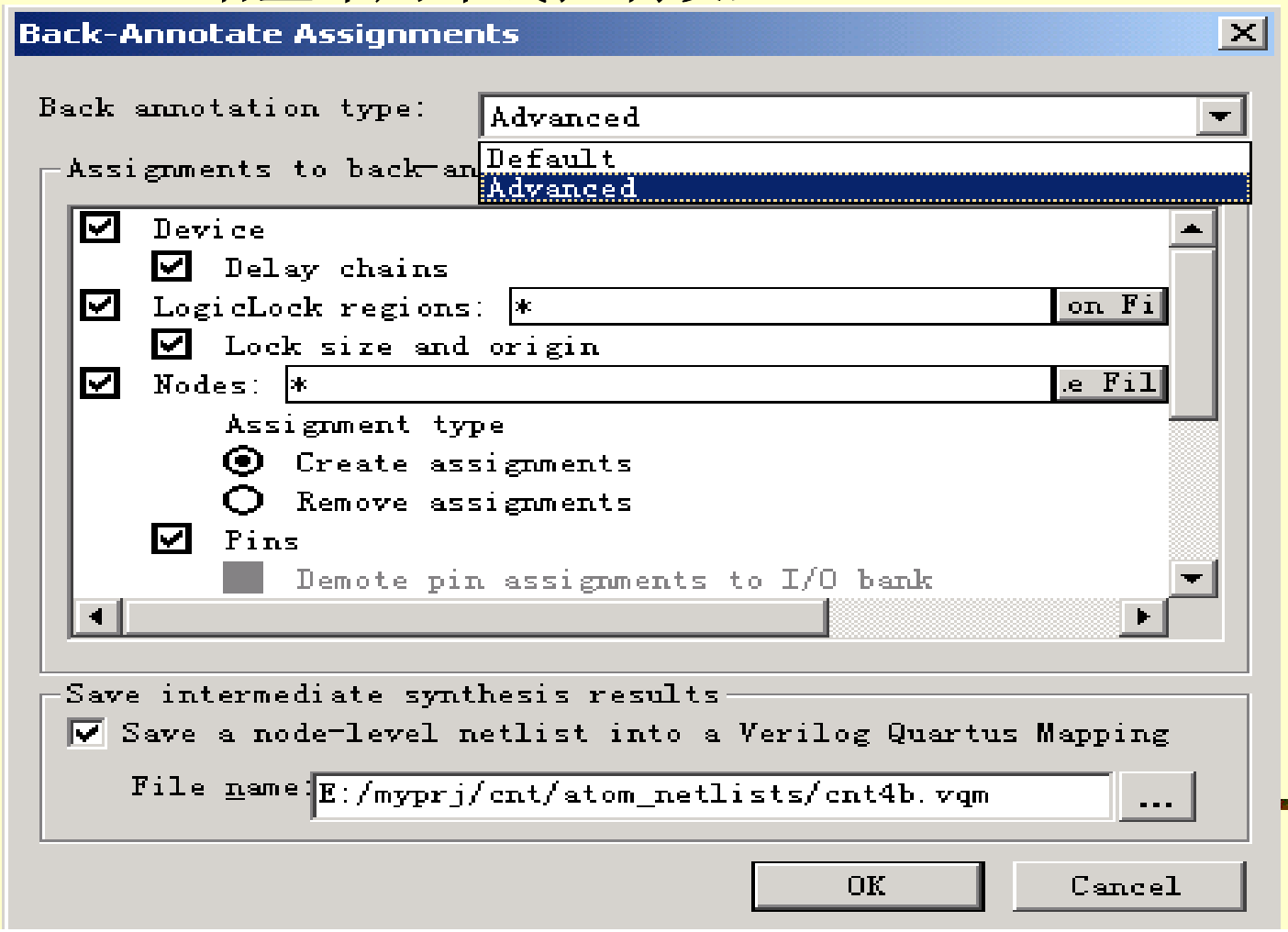



图11-12 反标设置

# 11.3 优化设置与时序分析

## 11.3.6 使用Design Assistant检查设计可靠性

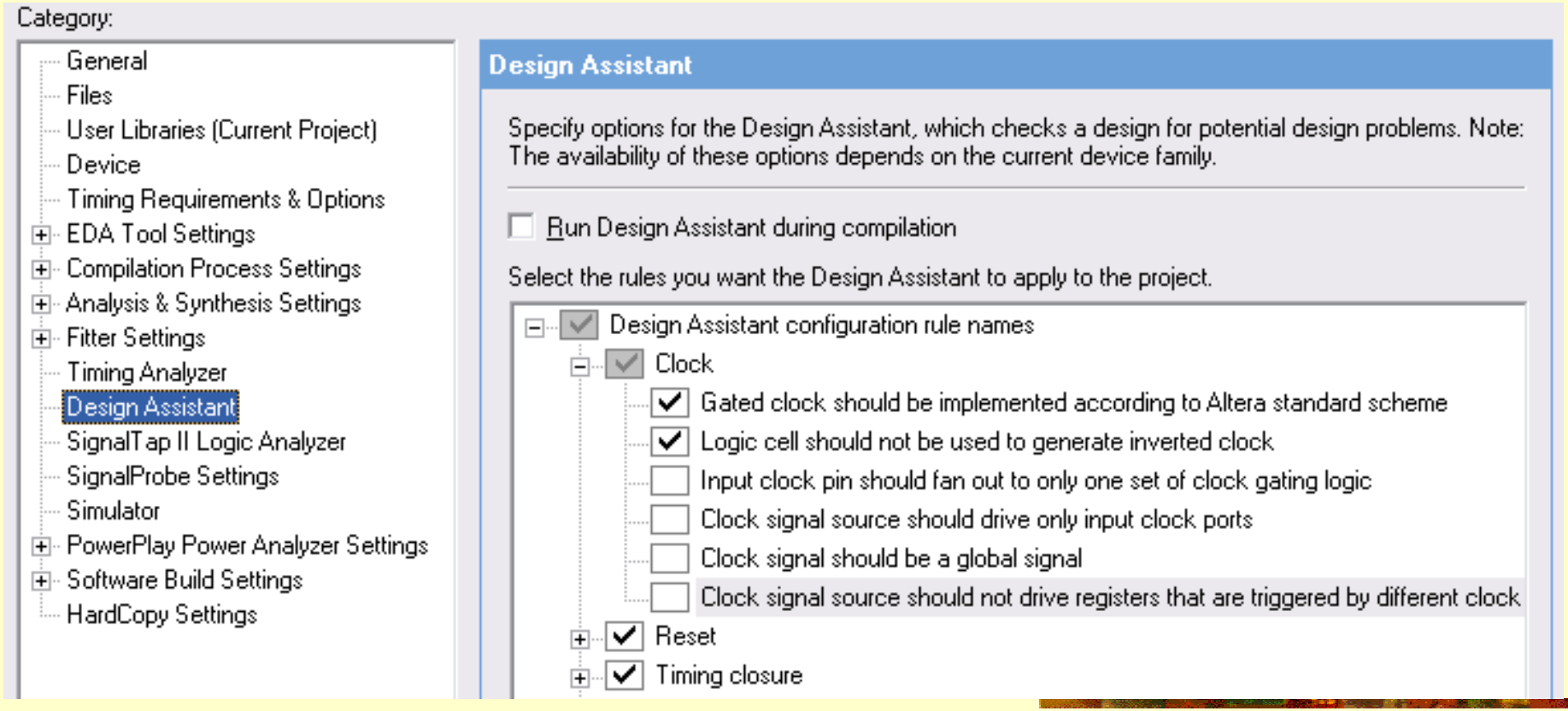



图11-13 Design Assistant设置

# 11.3 优化设置与时序分析

## 11.3.7 时序设置与分析

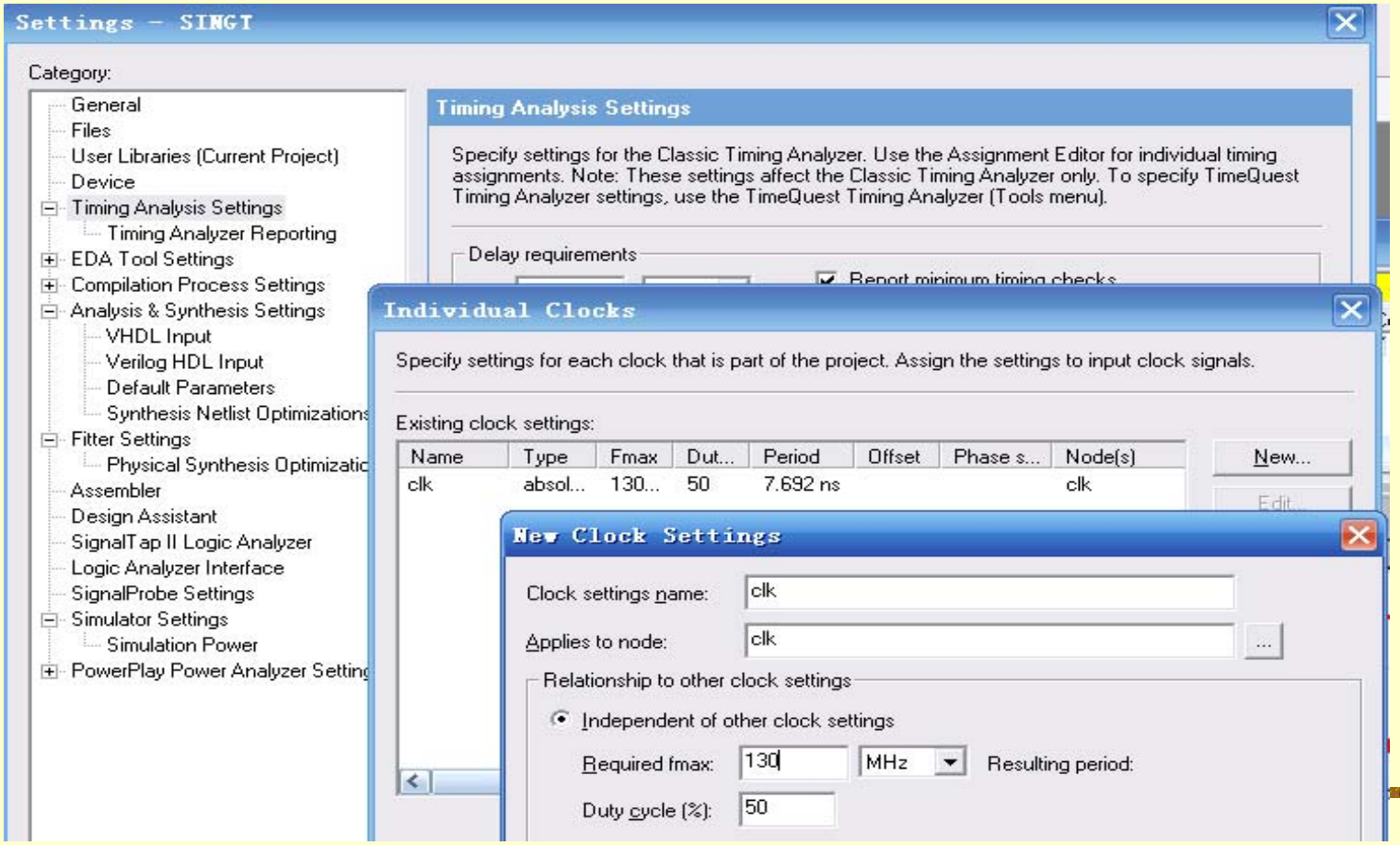



图11-14 全编译前时序条件设置（设置时钟信号CLK不低于130MHz）

# 11.3 优化设置与时序分析

## 11.3.7 时序设置与分析

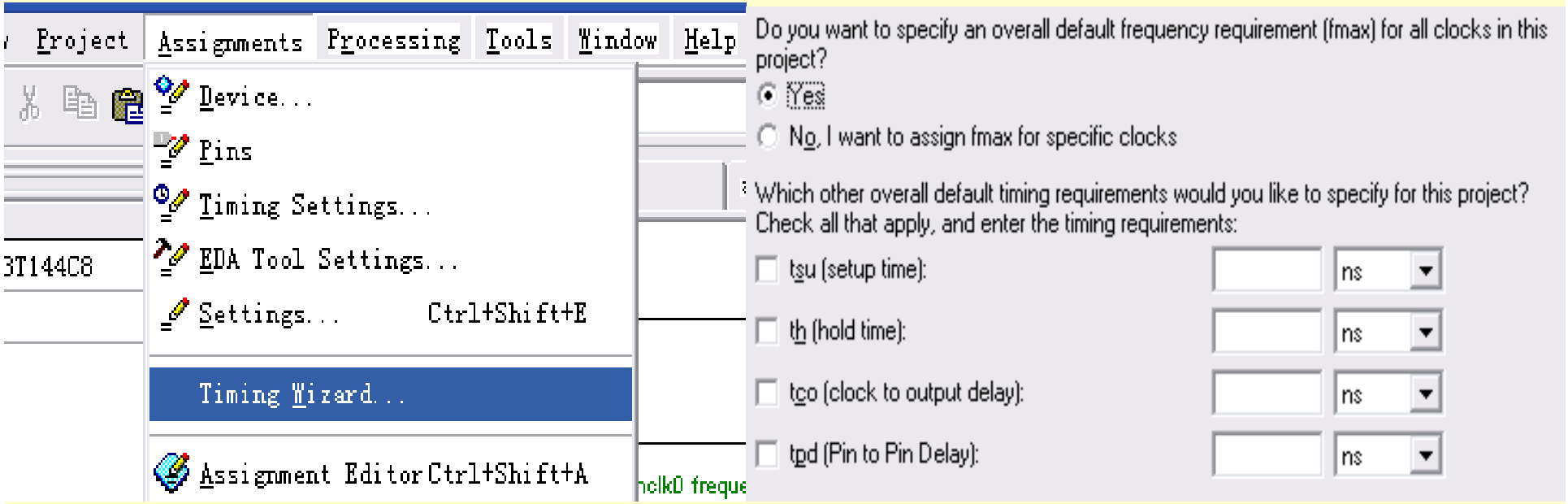
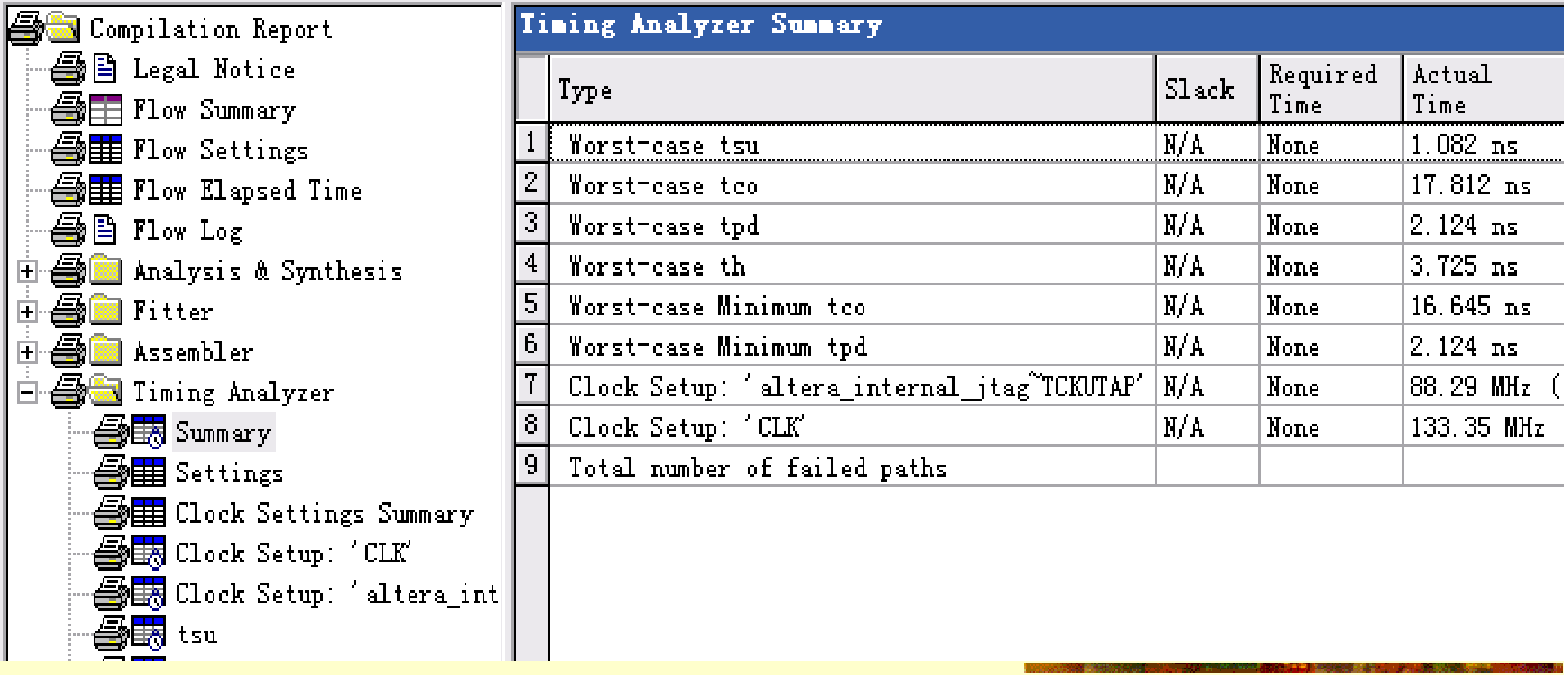



图11-15 由Timing Wizard窗口设置时序条件

# 11.3 优化设置与时序分析

## 11.3.8 查看时序分析结果




**Timing Analyzer Summary**

| | Type | Slack | Required Time | Actual Time |
|---|---|---|---|---|
| 1 | Worst-case tsu | N/A | None | 1.082 ns |
| 2 | Worst-case tco | N/A | None | 17.812 ns |
| 3 | Worst-case tpd | N/A | None | 2.124 ns |
| 4 | Worst-case th | N/A | None | 3.725 ns |
| 5 | Worst-case Minimum tco | N/A | None | 16.645 ns |
| 6 | Worst-case Minimum tpd | N/A | None | 2.124 ns |
| 7 | Clock Setup: 'altera_internal_jtag~TCKUTAP' | N/A | None | 88.29 MHz ( |
| 8 | Clock Setup: 'CLK' | N/A | None | 133.35 MHz |
| 9 | Total number of failed paths | | | |

图11-16 时序分析报告窗

# 11.3 优化设置与时序分析

## 11.3.8 查看时序分析结果

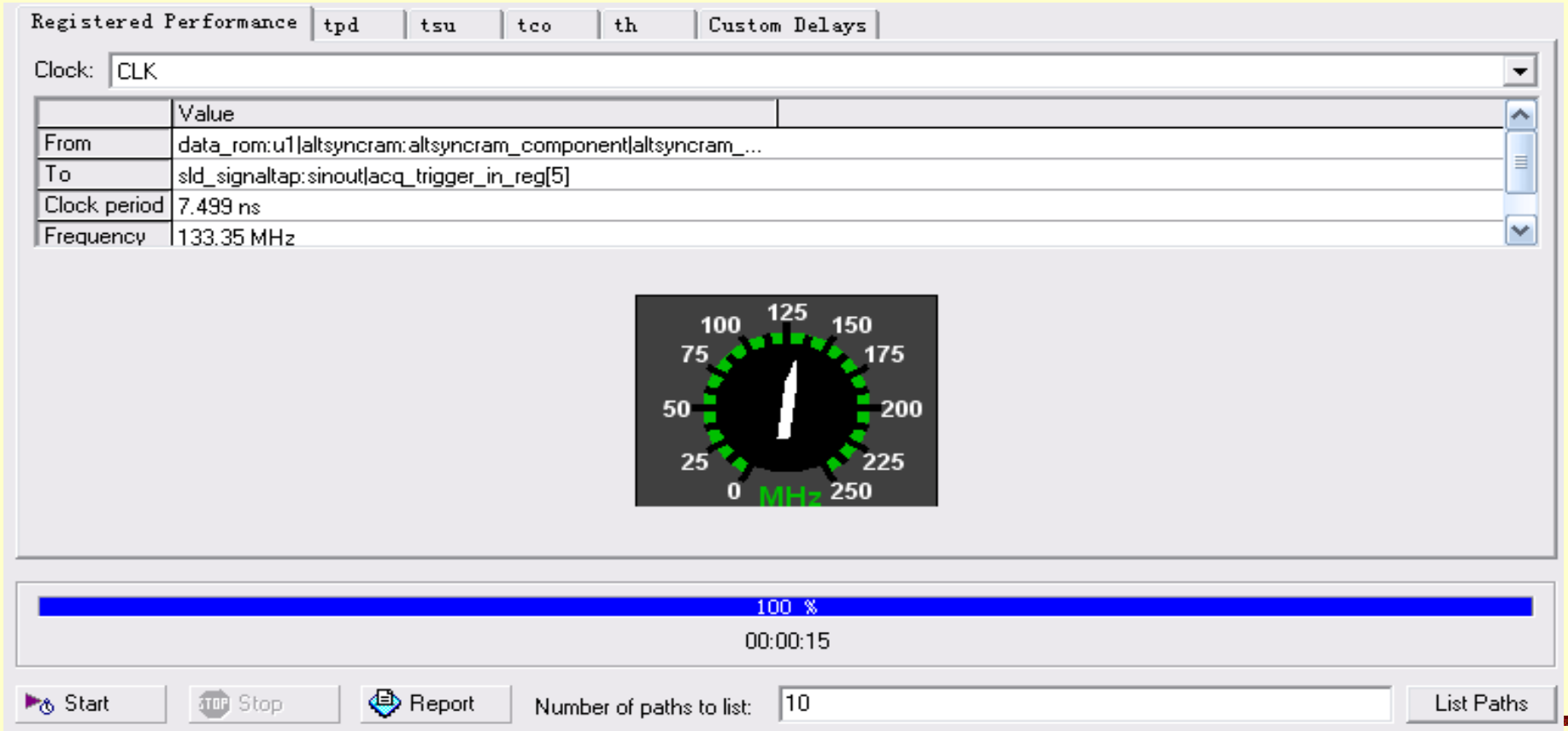



图11-17 Timing Analyzer Tool 项进入的时序分析报告窗

# 11.3 优化设置与时序分析

## 11.3.9 适配优化设置示例

| | |
|---|---|
| Family | APEX20KE |
| Device | EP20K300EQC240-3 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 51 / 11,520 ( < 1 % ) |
| Total pins | 9 / 152 ( 5 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 147,456 ( 0 % ) |
| Total PLLs | 0 |

图11-18 未用乘积项前的编译报告

【例11-9】 用CASE语句设计的正弦信号发生器

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SINGT IS
    PORT ( CLK      : IN STD_LOGIC;
           DOUT     : OUT INTEGER RANGE 255 DOWNTO 0      );
END;
ARCHITECTURE DACC OF SINGT IS
    SIGNAL Q        : INTEGER RANGE 63 DOWNTO 0 ;
    SIGNAL D        : INTEGER RANGE 255 DOWNTO 0 ;
BEGIN
PROCESS(CLK)
   BEGIN
   IF CLK'EVENT AND CLK = '1' THEN
IF Q < 63 THEN   Q <= Q + 1; ELSE   Q <= 0 ; END IF;  END IF;
END PROCESS;
PROCESS(Q)
   BEGIN
CASE Q IS
```

(接下页)

```
WHEN 00=> D<=255; WHEN 01=> D<=254; WHEN 02=> D<=252; WHEN 03=> D<=249;
WHEN 04=> D<=245; WHEN 05=> D<=239; WHEN 06=> D<=233; WHEN 07=> D<=225;
WHEN 08=> D<=217; WHEN 09=> D<=207; WHEN 10=> D<=197; WHEN 11=> D<=186;
WHEN 12=> D<=174; WHEN 13=> D<=162; WHEN 14=> D<=150; WHEN 15=> D<=137;
WHEN 16=> D<=124; WHEN 17=> D<=112; WHEN 18=> D<= 99; WHEN 19=> D<= 87;
WHEN 20=> D<= 75; WHEN 21=> D<= 64; WHEN 22=> D<= 53; WHEN 23=> D<= 43;
WHEN 24=> D<= 34; WHEN 25=> D<= 26; WHEN 26=> D<= 19; WHEN 27=> D<= 13;
WHEN 28=> D<=  8; WHEN 29=> D<=  4; WHEN 30=> D<=  1; WHEN 31=> D<=  0;
WHEN 32=> D<=  0; WHEN 33=> D<=  1; WHEN 34=> D<=  4; WHEN 35=> D<=  8;
WHEN 36=> D<= 13; WHEN 37=> D<= 19; WHEN 38=> D<= 26; WHEN 39=> D<= 34;
WHEN 40=> D<= 43; WHEN 41=> D<= 53; WHEN 42=> D<= 64; WHEN 43=> D<= 75;
WHEN 44=> D<= 87; WHEN 45=> D<= 99; WHEN 46=> D<=112; WHEN 47=> D<=124;
WHEN 48=> D<=137; WHEN 49=> D<=150; WHEN 50=> D<=162; WHEN 51=> D<=174;
WHEN 52=> D<=186; WHEN 53=> D<=197; WHEN 54=> D<=207; WHEN 55=> D<=217;
WHEN 56=> D<=225; WHEN 57=> D<=233; WHEN 58=> D<=239; WHEN 59=> D<=245;
WHEN 60=> D<=249; WHEN 61=> D<=252; WHEN 62=> D<=254; WHEN 63=> D<=255;
WHEN OTHERS => NULL ;
END CASE;  END PROCESS;
  DOUT <= D ;
END;
```

# 11.3 优化设置与时序分析

## 11.3.9 适配优化设置示例

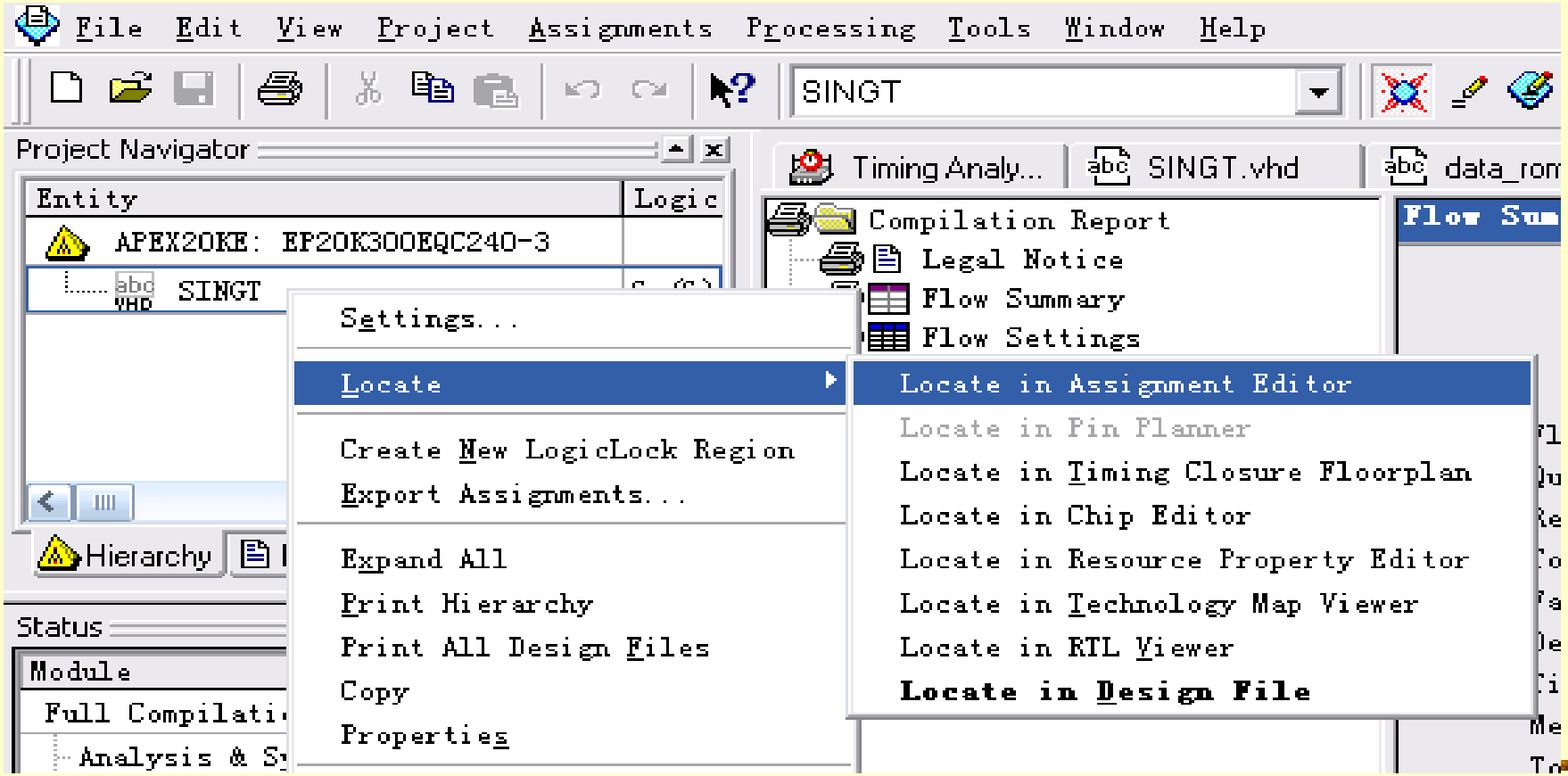



图11-19 针对工程选择Locate in Assignment Editor

# 11.3 优化设置与时序分析

## 11.3.9 适配优化设置示例

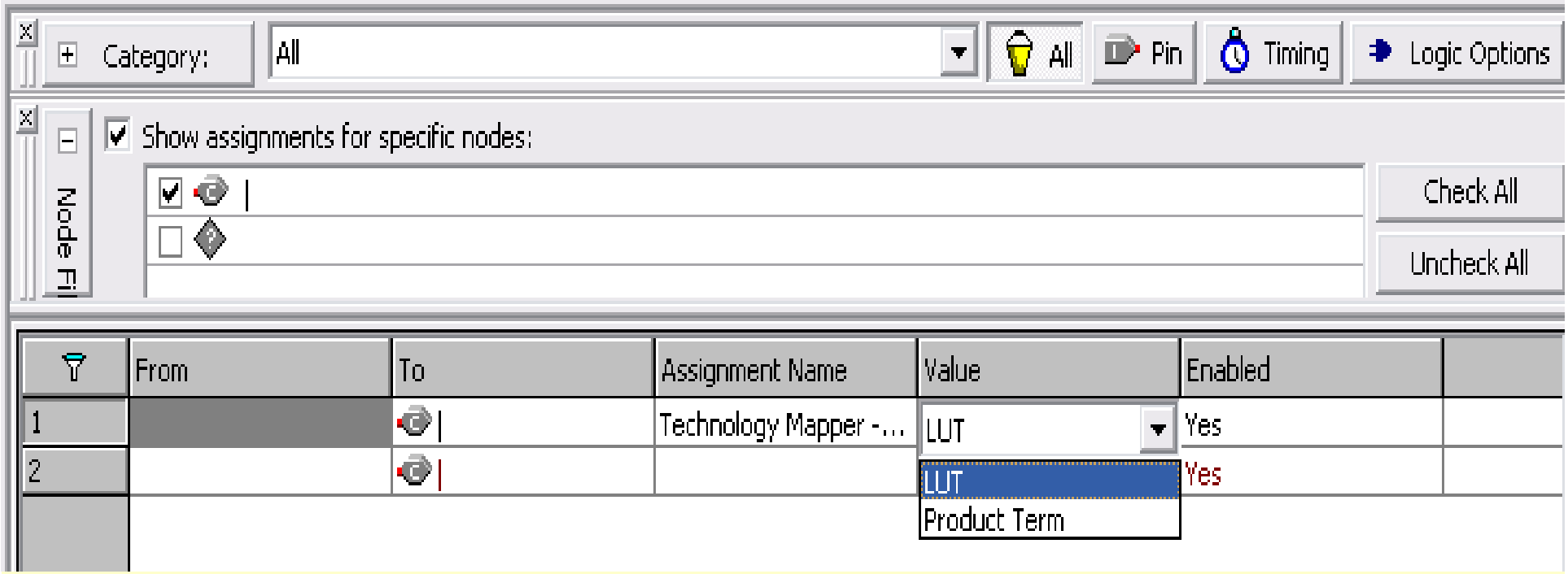



图11-20 选用乘积项逻辑优化

# 11.3 优化设置与时序分析

## 11.3.9 适配优化设置示例

图11-21在floorplan中可以看到使用了32个ESB

# 11.3 优化设置与时序分析

## 11.3.9 适配优化设置示例

| | |
|---|---|
| Family | APEX20KE |
| Device | EP20K300EQC240-3 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 6 / 11,520 ( < 1 % ) |
| Total pins | 9 / 152 ( 5 % ) |
| Total virtual pins | 0 |
| Number of product terms | 48 |
| Total PLLs | 0 |
| Total memory bits | 6,144 / 147,456 ( 4 % ) |

图11-22使用了乘积项的编译报告

# 11.3 优化设置与时序分析

## 11.3.10 Slow Slew Rate设置

Existing option settings:

| Name: | Setting: |
|---|---|
| Auto Register Duplication | Off |
| Enable Bus-Hold Circuitry | Off |
| Final Placement Optimizations | Automatically |
| I/O Placement Optimizations | On |
| Logic Cell Insertion - Logic Duplication | Auto |
| Optimize IOC Register Placement for ... | On |
| PCI I/O | Off |
| Placement Effort Multiplier | 1.0 |
| Router Effort Multiplier | 1.0 |
| Slow Slew Rate | On |
| Weak Pull-Up Resistor | Off |

图11-23 Slow Slew Rate选择

# 11.3 优化设置与时序分析

## 11.3.11 LogicLock优化技术

大规模系统开发中，应用逻辑锁定技术可以优化设计，合理分配硬件资料，提高系统的工作速度和可靠性。**QuartusII**支持逻辑锁定技术的**FPGA**器件系列有**APEX20K**、**APEXII**、**Excalibur**、**Cyclone/II**和**Stratix/II**等。

# 11.4 Chip Editor应用

## 11.4.1 Chip Editor应用实例

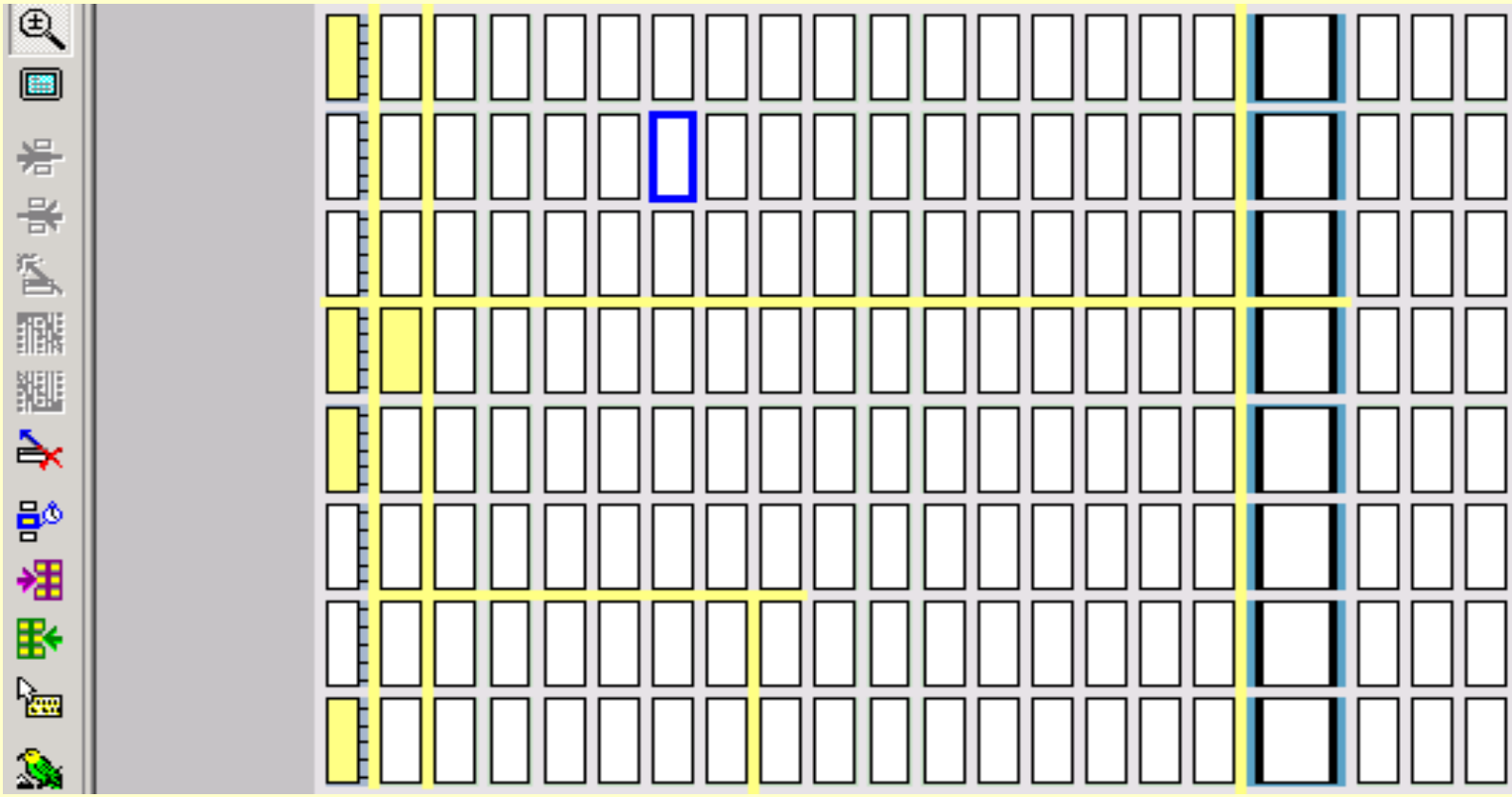
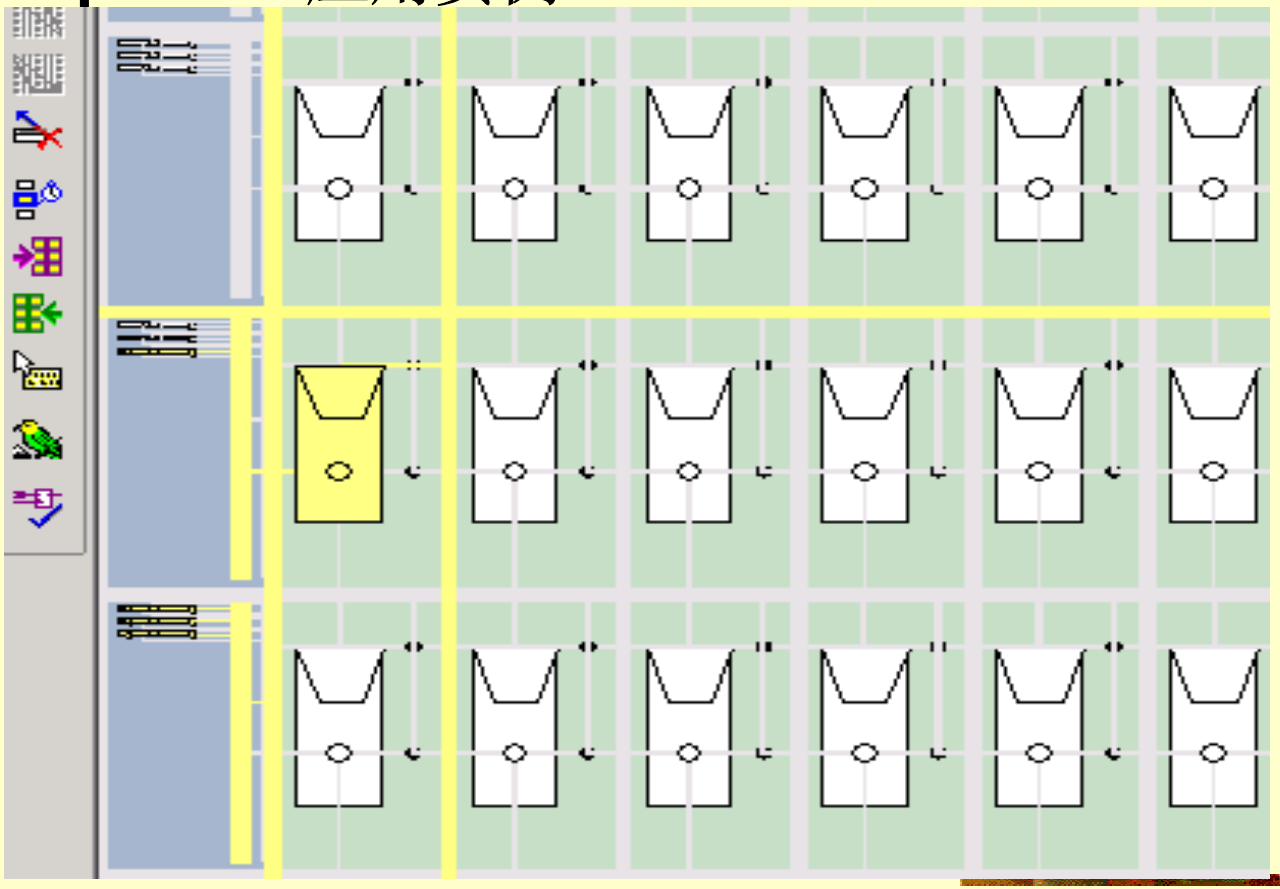


图9-24 最左侧是CNT4B占用的LAB

# 11.4 Chip Editor应用

## 11.4.1 Chip Editor应用实例



9-25 放大后的LAB分布

# 11.4 Chip Editor应用

## 11.4.1 Chip Editor应用实例

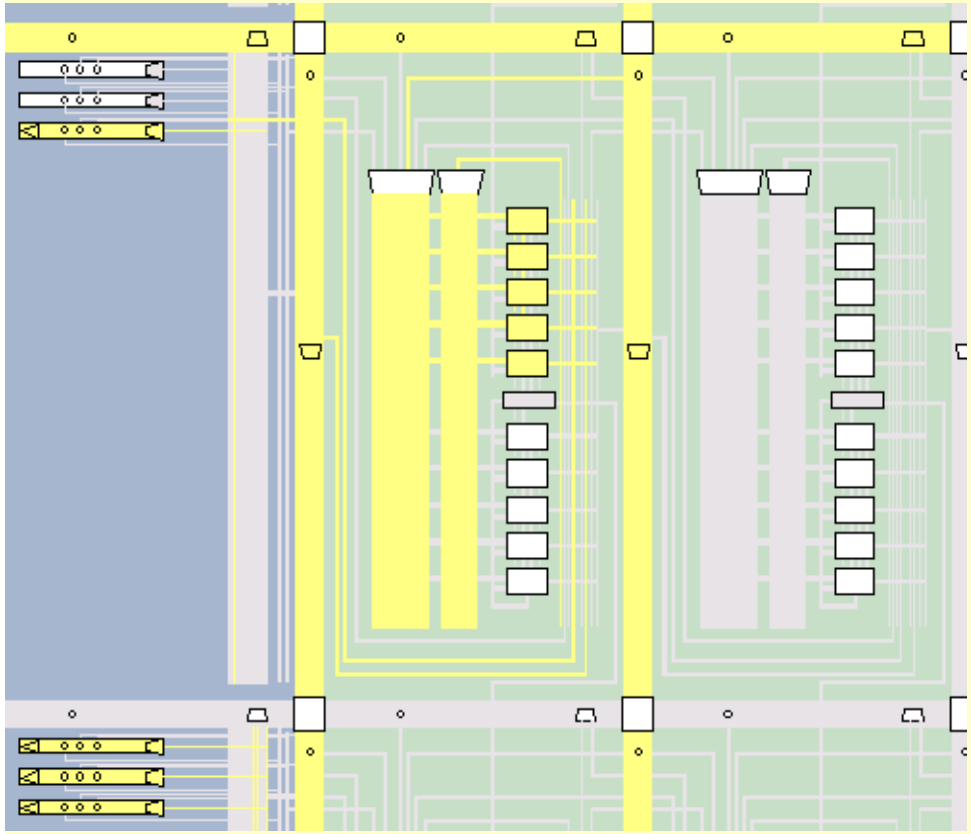


图11-26 被占用的LAB

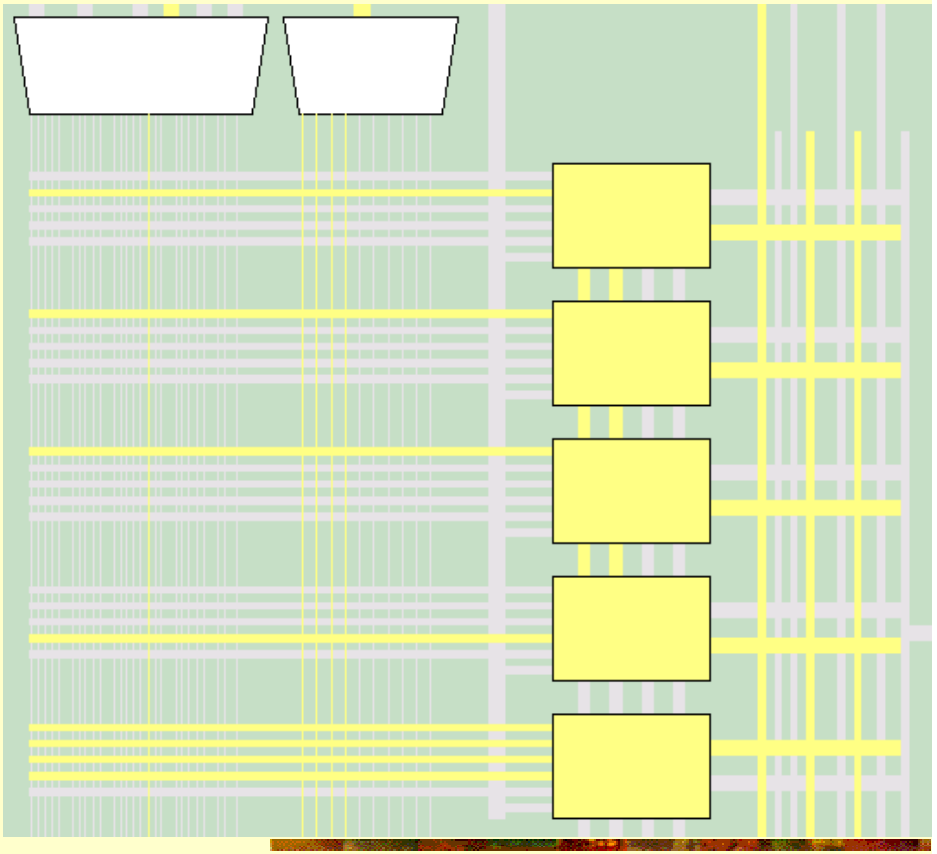


图11-27 LAB中被占用的5个LCs

# 11.4 Chip Editor应用

## 11.4.1 Chip Editor应用实例

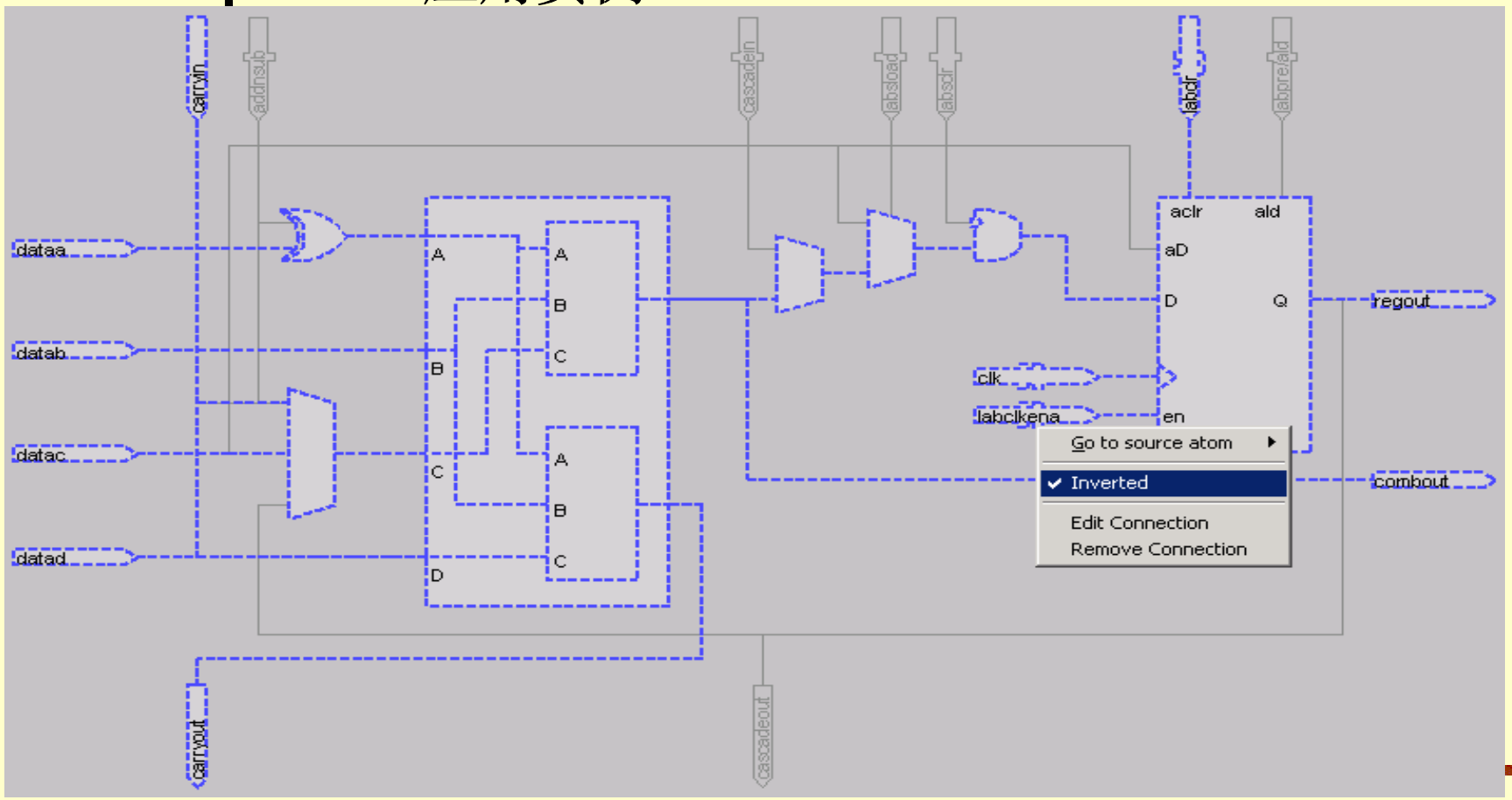



图11-28 Resource Property Editor的门级原理图编辑窗

# 11.4 Chip Editor应用

## 11.4.1 Chip Editor应用实例

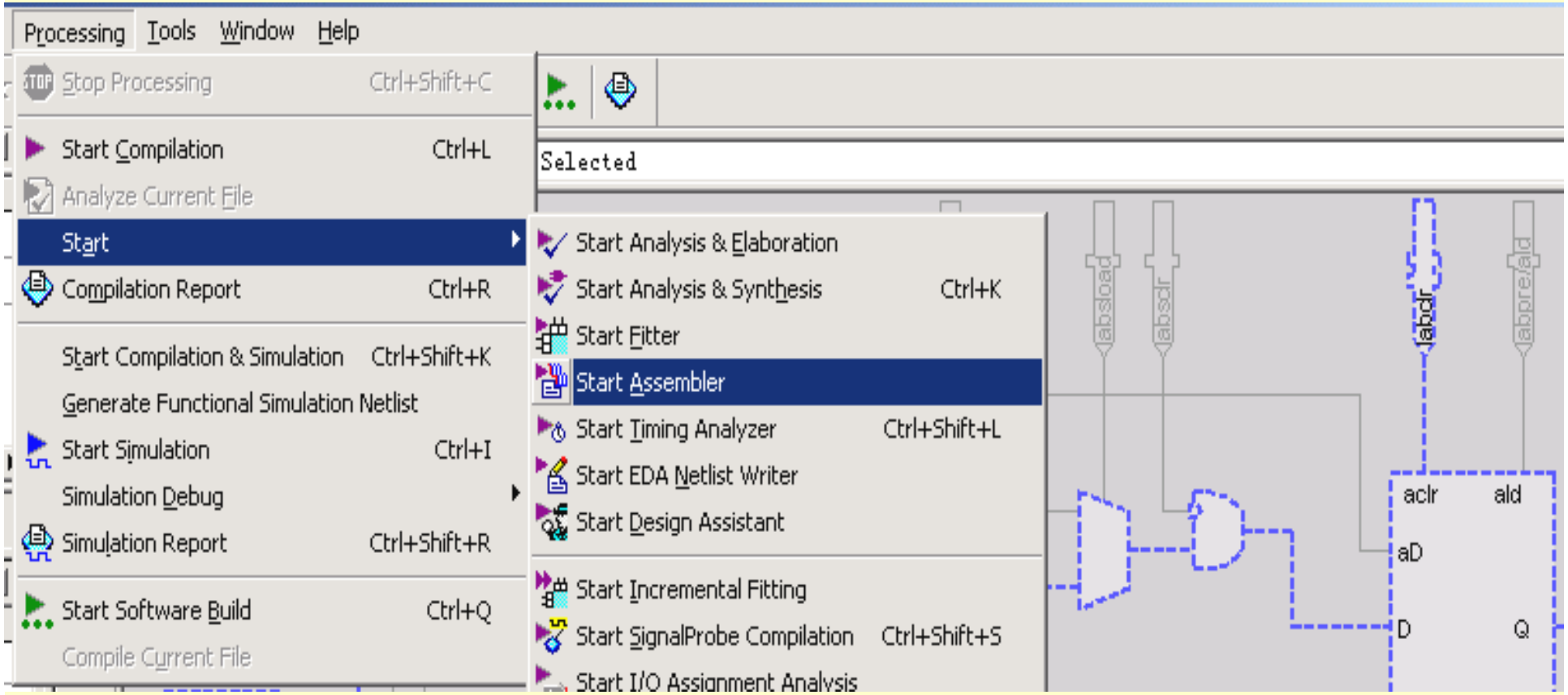



图11-29 的时序分析报告窗图

康芯科技

# 11.4 Chip Editor应用

## 11.4.2 Chip Editor功能说明

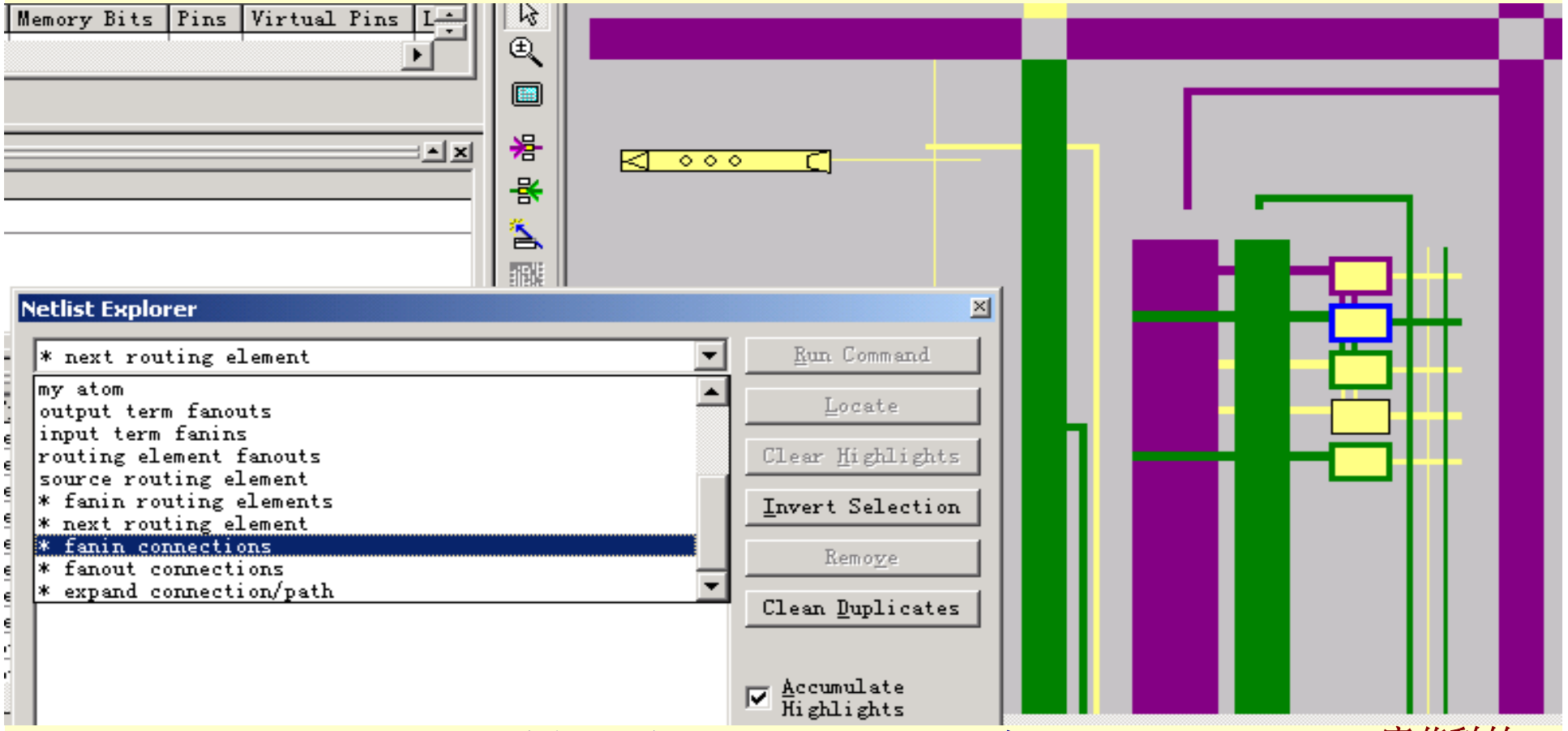



图9-30 打开Netlist Explorer窗

# 11.4 Chip Editor应用

## 11.4.2 Chip Editor功能说明

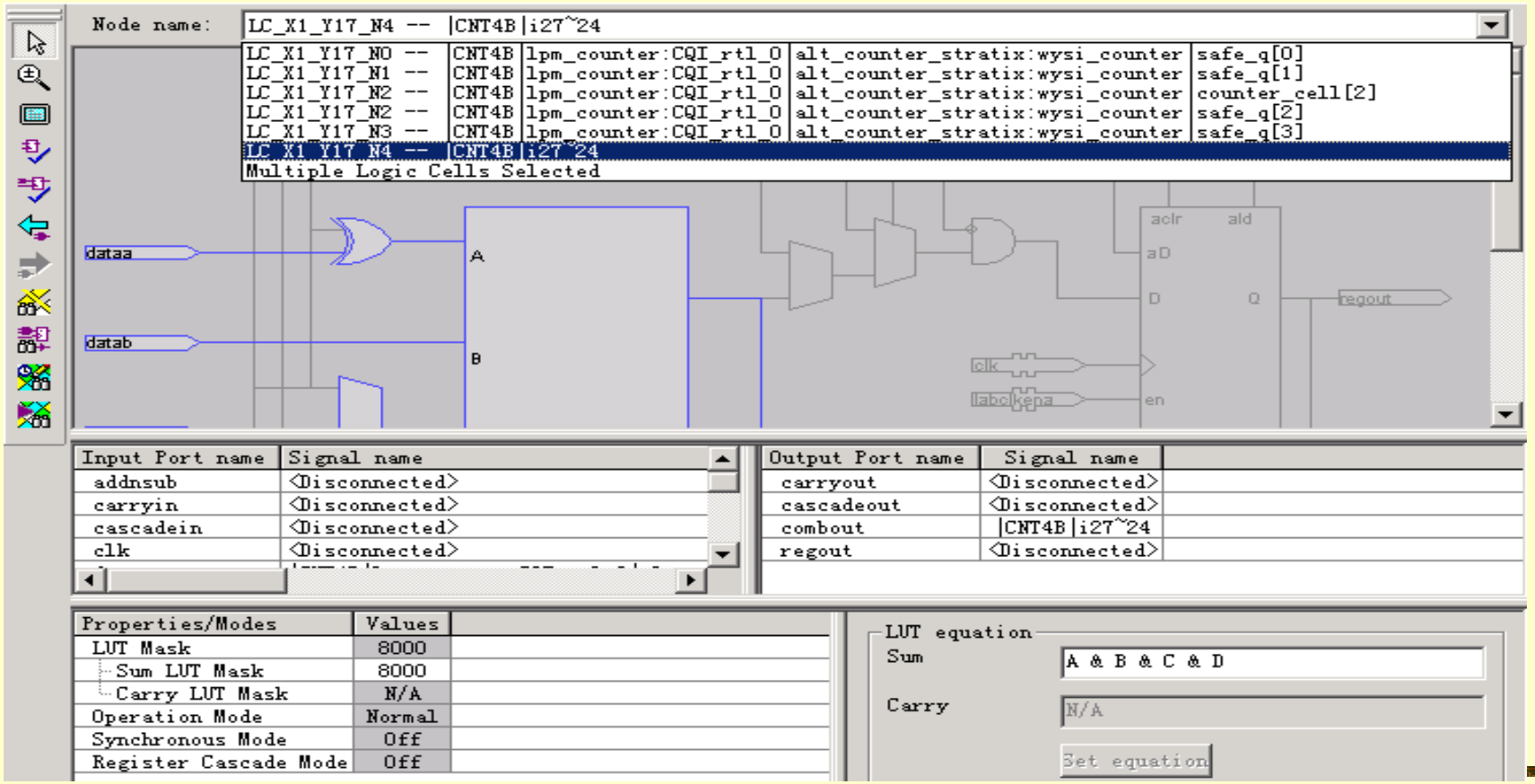



图11-31 打开属性和端口连接窗

# 11.4 Chip Editor应用

## 11.4.3 利用Change Manager检测底层逻辑

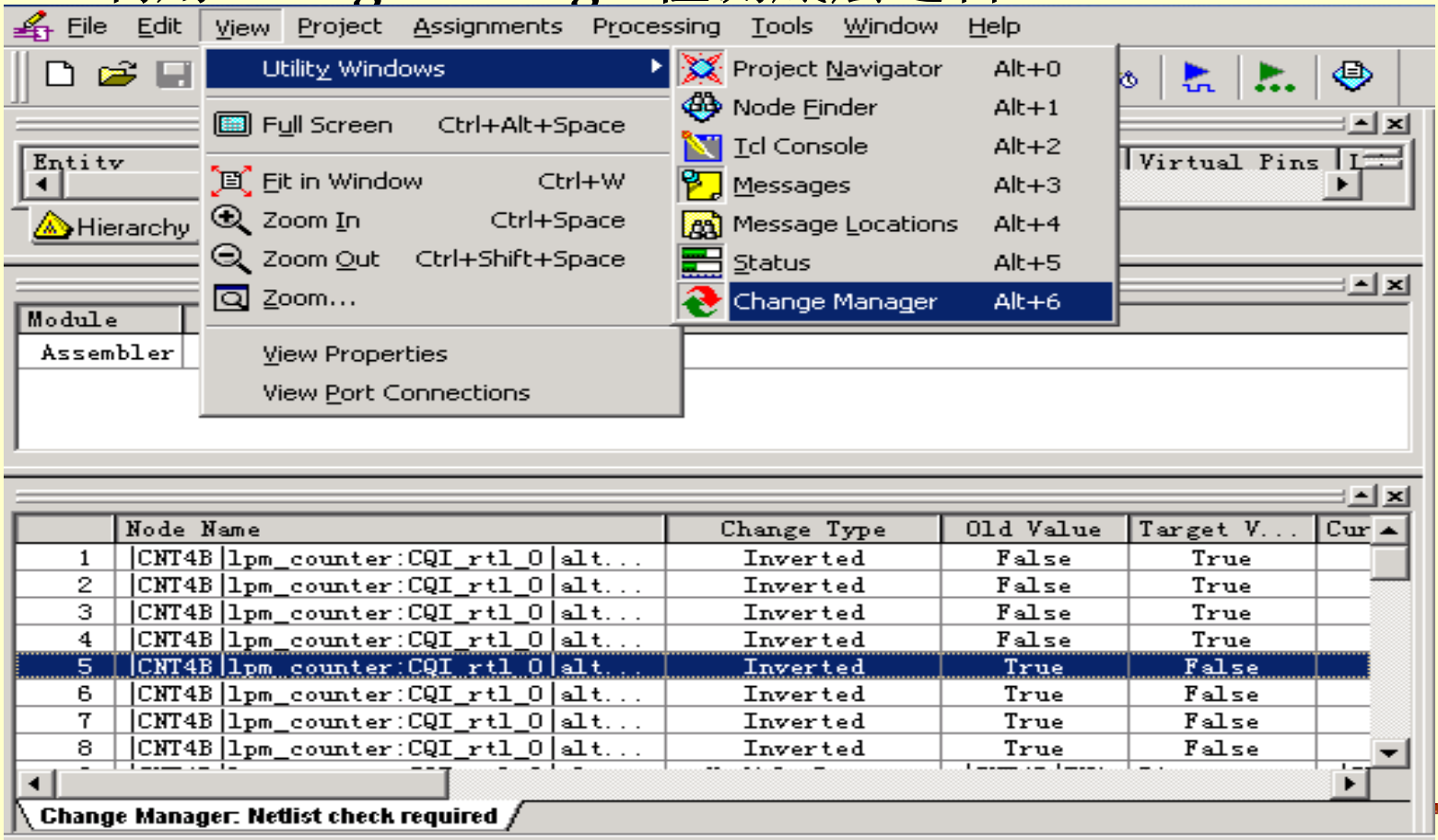



图11-32 打开Change Manager窗

# 习 题

**11-1.** 利用资源共享的面积优化方法对下面程序进行优化(仅要求在面积上优化)。习题程序如下：

【例11-10】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;
ENTITY addmux IS
    PORT (A,B,C,D   : IN  std_logic_vector(7 downto 0);
          sel       : IN  std_logic;
          Result    : OUT std_logic_vector(7 downto 0));
END addmux;
ARCHITECTURE rtl OF addmux IS
BEGIN
    process(A,B,C,D,sel)
    begin
        if(sel = '0') then  Result <= A + B;
                      else   Result <= C + D;
        end if;
    end process;
END rtl;
```

# 习 题

**11-2.** 试通过优化逻辑的方式对图**11-33**中所示的结构进行改进，给出**VHDL**代码和结构图。

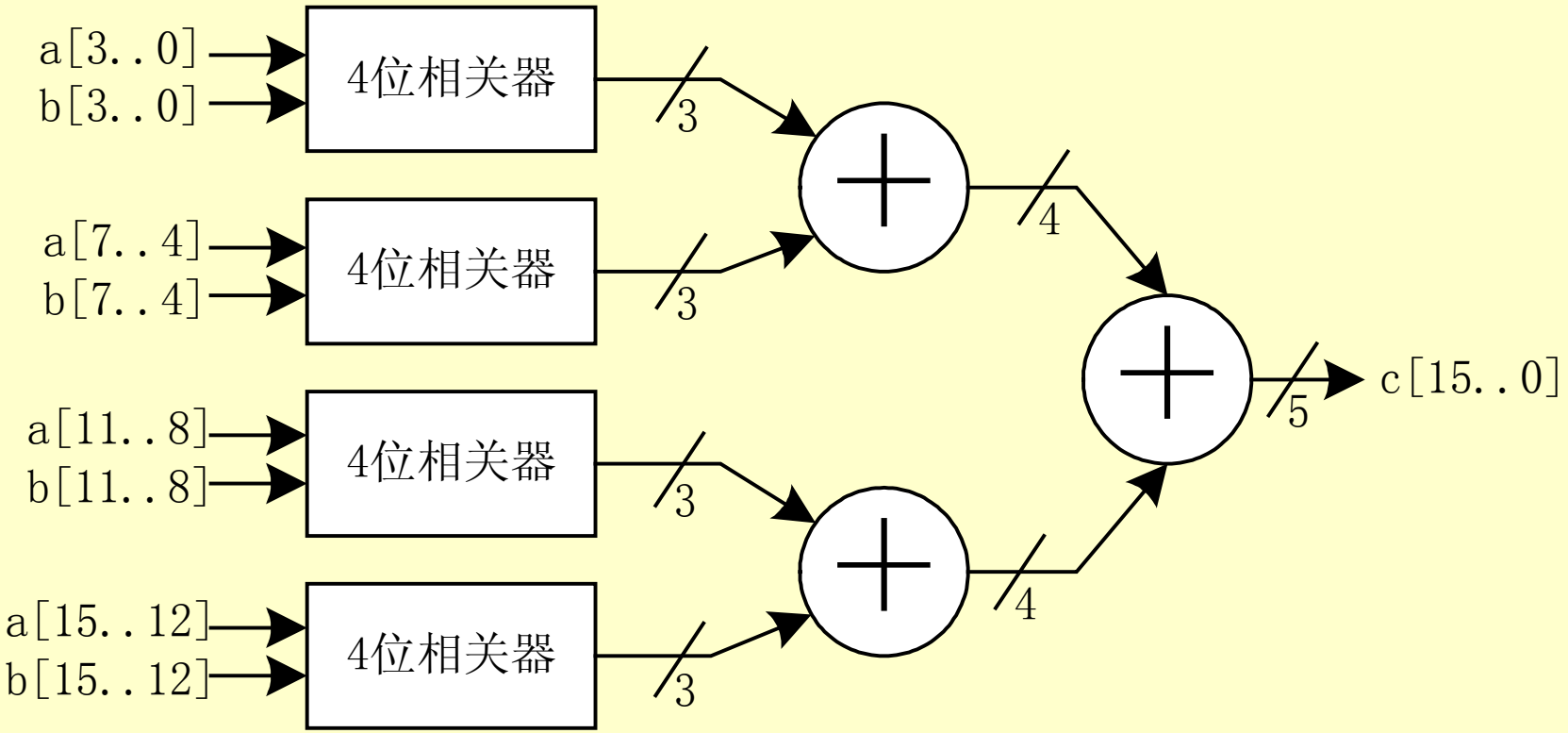



图11-33 习题11-2图

# 习 题

**11-3.** 已知4阶直接型FIR滤波器节的数学表达式如下：

$$y(n)=x(n)h(0)+x(n-1)+x(n-2)h(2)+x(n-3)h(3)$$

$x(n)$与$x(n-m)$，$m$=0，1，2，3是延迟关系，$m$表示延迟的clk数。$x(n-m)$与$h(m)$的位宽均为8位，$y(n)$为10位，其中$h(m)$在模块例化后为常数。该模块的输入为$x(n)$、clk，输出为$y(n)$，试实现该逻辑。

**11-4.** 对习题11-3中的FIR滤波器节在速度上进行优化(在$h(m)$固定的情况下)，试采用流水线技术。

**11-5.** 利用FPGA的LUT结构，构建资源占用较小的常数乘法器，改进习题11-3、11-4的设计，减少模块的资源使用。

11-6. 若对速度要求不高，但目标芯片的容量较小，试把习题11-3中的FIR滤波器用串行化的方式实现。

# 习 题

**11-7.** 设计一个连续乘法器，输入为**a0**，**a1**，**a2**，**a3**，位宽各为**8**位，输出**rout**为**32**位，完成**rout=a0*a1*a2*a3**，试实现之。

**11-8.** 对**11-7**进行优化，判断以下实现方法，那种方法更好？

**rout=((a0*a1)*a2)*a3**

**rout=(a0*a1)*(a2*a3)**

**11-9.** 为提高速度，对习题**11-8**中的前一种方法加上流水线技术进行实现。

**11-10.** 试对以上的习题解答通过设置**QuartusII**相关选项的方式，提高速度，减小面积。

# 实 验 与 设 计

## 11-1 采用流水线技术设计高速数字相关器

**(1) 实验目的：**设计一个在数字通信系统中常见的数字相关器，并利用流水线技术提高其工作速度，对其进行仿真和硬件测试。

**(2) 实验原理：**数字相关器用于检测等长度的两个数字序列间相等的位数，实现序列间的相关运算。

一位相关器，即异或门，异或的结果可以表示两个1位数据的相关程度。异或为0表示数据位相同；异或为1表示数据位不同。多位数字相关器可以由多个一位相关器构成，如N位的数字相关器由N个异或门和N个1位相关结果统计电路构成。

**(3) 实验内容1：**根据上述原理设计一个并行4位数字相关器(例9-17是示例程序)。

提示：利用CASE语句完成4个1位相关结果的统计。

# 实 验 与 设 计

## 11-1 采用流水线技术设计高速数字相关器

【例11-11】

```
stemp <= a XOR b;
PROCESS(stemp) BEGIN
        CASE stemp IS
         WHEN "0000" => c <= "100";
        --4
         WHEN "0001"|"0010"|"0100"|"1000" => c <= "011";
        --3
         WHEN "0011"|"0101"|"1001"|"0110"|"1010"|"1100" => c <= "010";      --2
                  WHEN "0111"|"1011"|"1101"|"1110" => c <= "001";            --1
                  WHEN "1111" => c <= "000";          -- 0;
                  WHEN OTHERS => c <= "000";
        END CASE;
END PROCESS;
```

# 实 验 与 设 计

## 11-1 采用流水线技术设计高速数字相关器

**(4) 实验内容2：**利用实验内容1中的4位数字相关器设计并行16位数字相关器。使用QuartusII估计最大延时，并计算可能运行的最高频率。

**(5) 实验内容3：**在以上实验的基础上，利用设计完成的4位数字相关器设计并行16位数字相关器，其结构框图可参考图11-33，并利用QuartusII计算运行速度。

**(6) 实验内容4：**上面的16位数字相关器是用3级组合逻辑实现的，在实际使用时，对其有高速的要求，试使用流水线技术改善其运行速度。在输入、输出及每一级组合逻辑的结果处加入流水线寄存器，提高速度，可参照本章内容进行设计。

**(7) 实验思考题：**考虑采用流水线后的运行速度与时钟clock的关系，测定输出与输入的总延迟。若输入序列是串行化的，数字相关器的结构如何设计？如何利用流水线技术提高其运行速度？

**(8) 实验报告：**根据以上的实验内容写出实验报告，包括设计原理、程序设计、程序分析、仿真分析、硬件测试和详细实验过程。

# 实 验 与 设 计

## 11-2 线性反馈移位寄存器设计

**(1) 实验目的：**学习用VHDL设计LFSR，掌握利用FPGA的特殊结构中高效实现LFSR的方法。

**(2) 实验原理：**LFSR即Linear Feedback Shift Register 线性反馈移位寄存器，是一种十分有用的时序逻辑结构，广泛用于伪随机序列发生、可编程分频器、CRC校验码生成、PN码等等。图11-34是典型的LFSR结构。由图中可以看出LFSR由移位寄存器加上xor构成，不同的xor决定了不同的生成多项式。图11-34中的生成多项式为$X^3+X^2+X^0$。

**(3) 实验内容：**依据图11-34设计一个LFSR，其生成多项式为$X^4+X^3+X^0$。试在FPGA上加以实现，并利用本章中提及的QuartusII优化选项，使之达到最高运行速度，并在GW48 EDA开发系统上，对其产生的码序列进行观察。

# 实 验 与 设 计

## 11-2 线性反馈移位寄存器设计

**(4) 实验思考题1：**另有一种LFSR的结构，见图11-35，试分析与图11-34中LFSR结构的异同点。

**(5) 实验思考题2：**对图11-35结构的LFSR电路进行改进，设计成串行CRC校验码发生器(提示：反馈线上加入xor，xor的一个输入端接待编码串行有效信息输入)。

**(6) 实验报告：**作出本项实验设计的完整电路图，详细说明其工作原理，完成测试实验内容，对实验中的码序列进行记录，写出电路可达到的最高运行速度及设置的QuartusII选项。

# 实 验 与 设 计

## 11-2 线性反馈移位寄存器设计

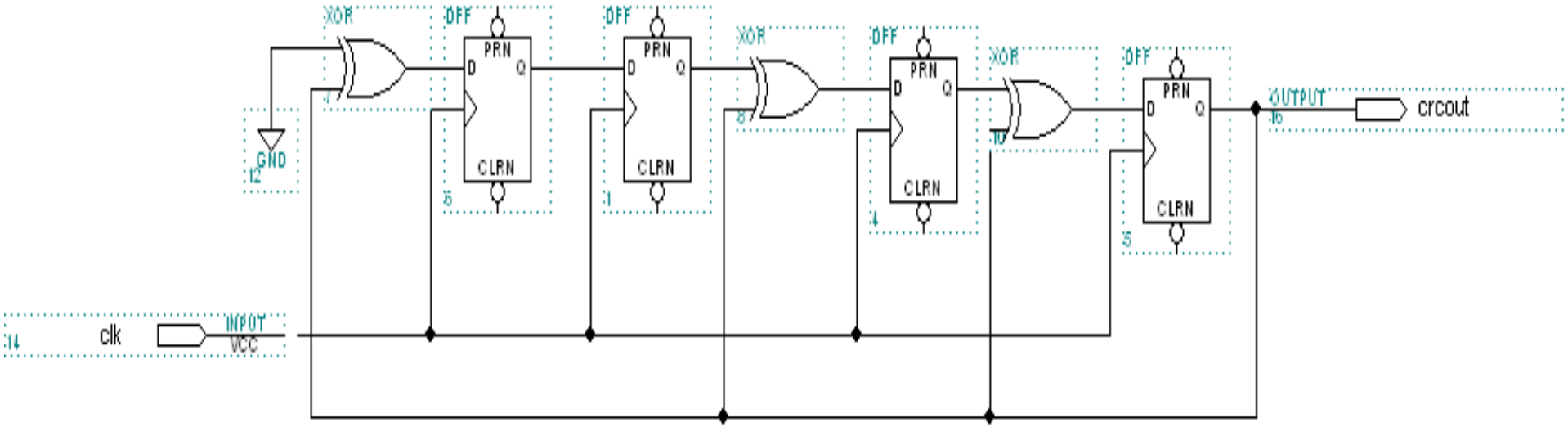



图11-34 LFSR举例

# 实 验 与 设 计

## 11-2 线性反馈移位寄存器设计

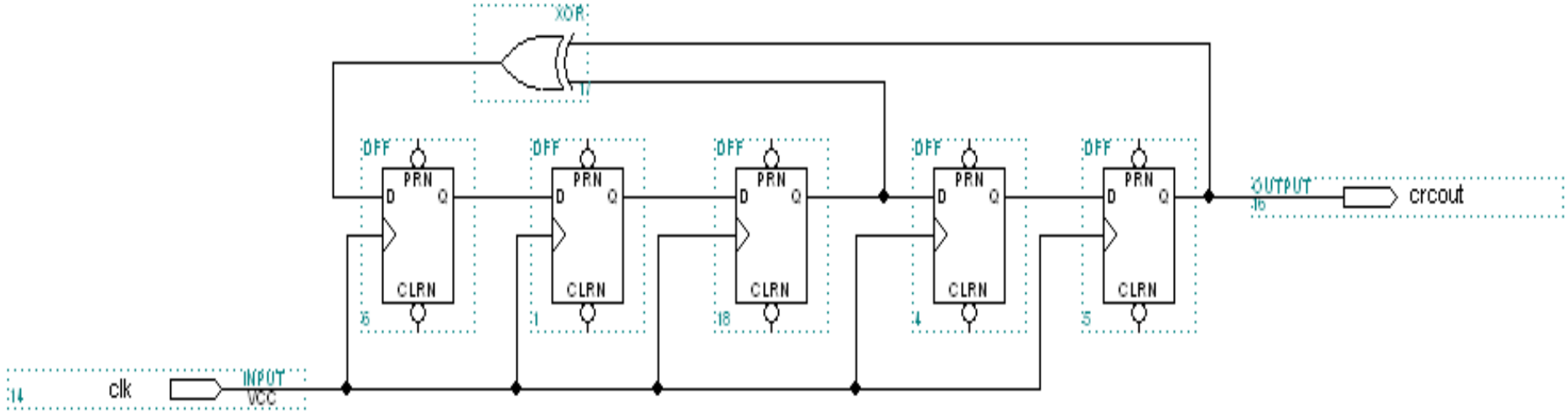



图11-35 另一种LFSR结构

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

**(1) 实验目的：**学习利用EDA技术和FPGA实现直接数字频率综合器DDS的设计。

**(2) 实验原理：**直接数字频率综合技术，即DDS技术，是一种新型的频率合成技术和信号产生方法。其电路系统具有较高的频率分辨率，可以实现快速的频率切换，并且在改变时能够保持相位的连续，很容易实现频率、相位和幅度的数控调制。

$$f_{SIN} = M（f_{clk}/2^n）$$ 11-1

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

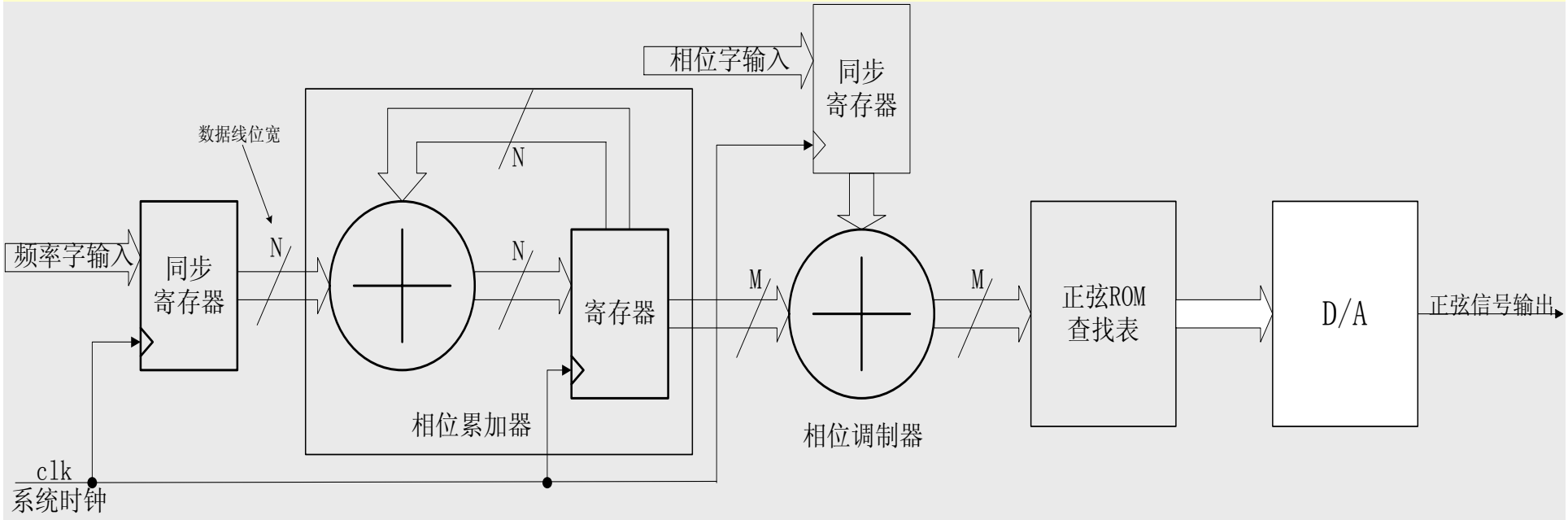
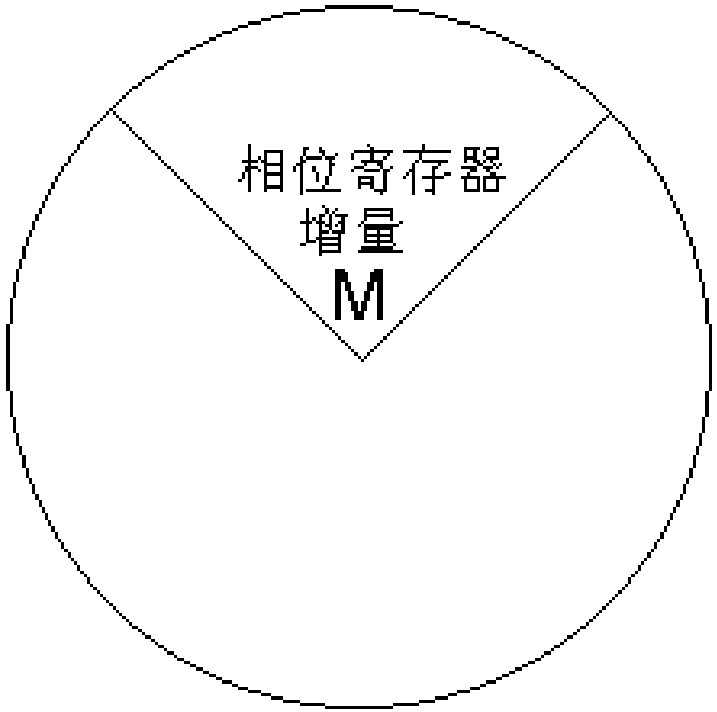



图11-36 DDS基本结构

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验




| 相位累加器宽 | 对应采样点数 |
|---|---|
| 8 | 256 |
| 12 | 4096 |
| 16 | 65536 |
| 20 | 1048576 |
| 24 | 16777216 |
| 28 | 268435456 |
| 32 | 4294967296 |

图11-37 相位累加器位宽和采样点关系

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

【例11-12】

```
LIBRARY ieee; --波形数据ROM
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
ENTITY data_rom IS
        PORT
        (       address         : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
                inclock         : IN STD_LOGIC ;
                q               : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)     );
END data_rom;
 . . .
                init_file => "./data/LUT10X10.mif", --波形数据初始化文件路径
                lpm_hint => "ENABLE_RUNTIME_MOD=YES, INSTANCE_NAME=rom2",
. . .
END;
```

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

【例11-13】

```
LIBRARY IEEE;  --32位加法器模块
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY ADDER32B IS
    PORT (  A : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            S : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)     );
END ADDER32B;
ARCHITECTURE behav OF ADDER32B IS
    BEGIN
     S <= A + B;
END behav;
```

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

【例11-14】--32位寄存器模块

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY REG32B IS
    PORT (  Load : IN STD_LOGIC;
             DIN : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            DOUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END REG32B;
ARCHITECTURE behav OF REG32B IS
BEGIN
    PROCESS(Load, DIN)
   BEGIN
   IF Load'EVENT AND Load = '1' THEN     -- 时钟到来时，锁存输入数据
            DOUT <= DIN;
        END IF;
    END PROCESS;
END behav;
```

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

【例11-15】rom_data.mif 10位正弦波数据文件，读者可用MATLAB/DSP Builder生成

```
WIDTH=10;
DEPTH=1024;
ADDRESS_RADIX=DEC;
DATA_RADIX=DEC;
CONTENT BEGIN
  0 : 513; 1 : 515; 2 : 518; 3 : 521; 4 : 524; 5 : 527;  6 : 530; 7 : 533;
 8 : 537; 9 : 540; 10 : 543; 11 : 546; 13 : 549; 13 : 552; 14 : 555;
  . . . . . （略去部分数据）
  1018 : 493; 1019 : 496; 1020 : 499; 1021 : 502; 1022 : 505; 1023 : 508;
END;
```

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

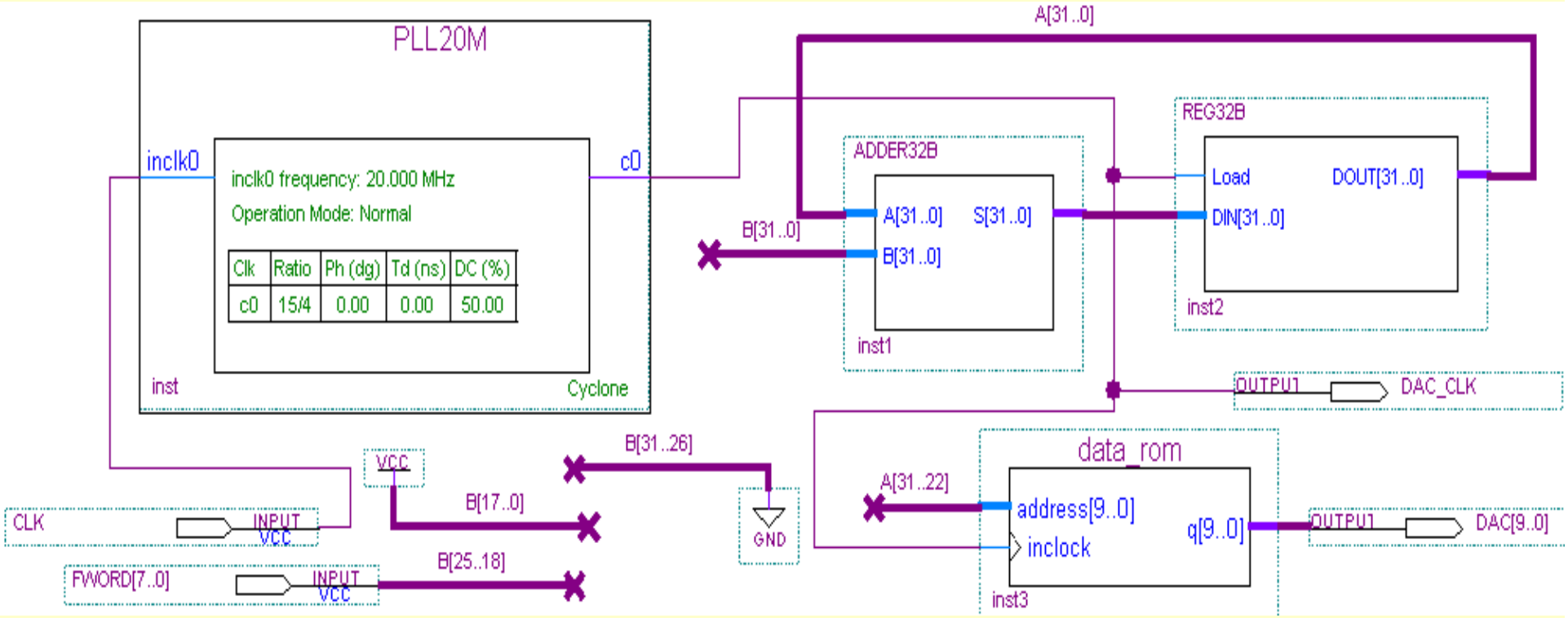



图11-38 DDS.vhd顶层原理图

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

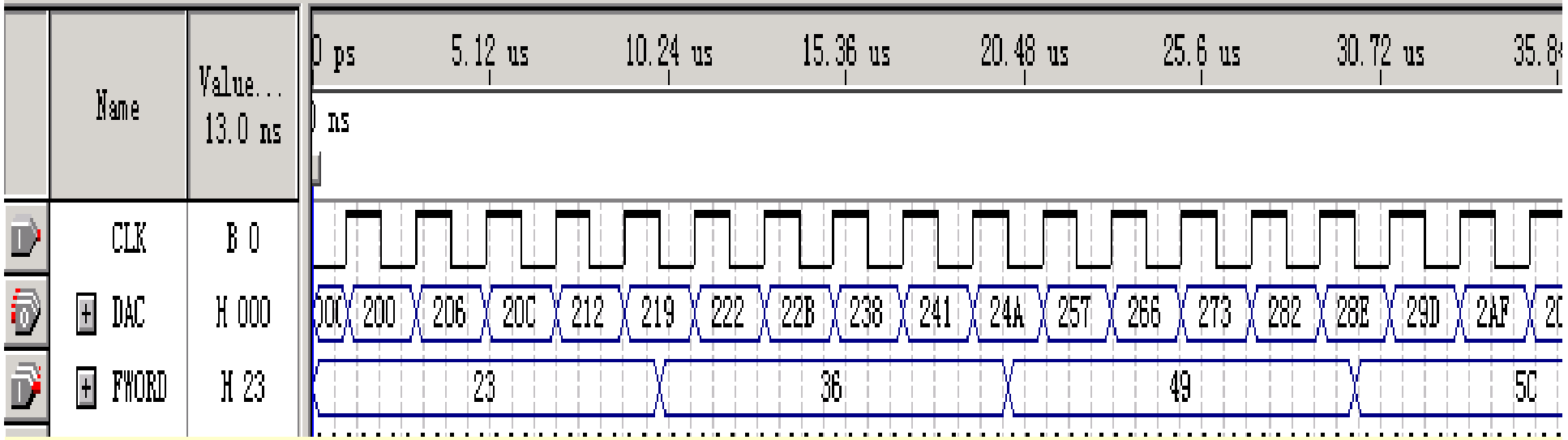



图11-39 DDS.vhd仿真波形

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

**(3) 实验内容1：** 详细述叙DDS的工作原理，依据例11-12至例11-15完成仿真，并由仿真结果进一步说明DDS的原理。完成编译和下载。选择模式1；键2、键1输入8位频率字FWORD；利用GW48系统ADDA板上的10位D/A5651输出波形，用示波器观察输出波形。

**(4) 实验内容2：** 根据图11-36，在原设计（图11-38）中加入相位控制电路，用键4、键3输入8位相位字PWORD；重复实验1的内容。

**(5) 实验内容3：** 将图11-38的顶层原理图表述为VHDL，重复实验1的内容。

**(6) 实验内容4：** 在图11-38的设计中增加一些元件，设计成扫频信号源，扫频速率、扫频频域、扫频步幅可设置。

# 实 验 与 设 计

## 11-3 直接数字式频率合成器(DDS)设计实验

**(7) 实验内容5：** 例11-12后的程序将32位频率字和10位相位字作了截断，都是8位。如果不作截断，修改其中的程序，并设法在GW48实验系统上完成实验（提示，增加2个锁存器与单片机通信）。

**(8) 实验内容6：** 将上例改成频率可数控的正交信号发生器，即使电路输出两路信号，且相互正交，一路为正弦(sin)信号，一路为余弦(cos)信号（此电路可用于正交方式的信号调制解调）。

**(9) 实验内容7：** 利用上例设计一个FSK信号发生器，并硬件实现之。

**(10) 思考题：** 如果不作截断，此例的频率精度和相位精度分别是多少？

# 实 验 与 设 计

**11-4 基于DDS的数字移相信号发生器设计实验**

**(1) 实验原理：**移相信号发生器是2003年大学生电子设计竞赛题中的一个设计项目。图11-40是基于DDS模型的数字移相信号发生器的电路模型图，示例程序如例11-16所示。

**（2）实验内容1：**完成10位输出数据宽度的移相信号发生器的设计，其中包括设计正弦波形数据MIF文件（数据深度1024、数据类型是十进制数）；给出仿真波形。最后进行硬件测试，对于GW48系统，推荐选择模式1：CLK接clock0，接13MHz；用键4、3控制相位字PWORD输入，键2、1控制频率字FWORD输入。观察它们的图形和李萨如图形。

**（3）实验内容2：**修改设计，增加幅度控制电路（如可以用一乘法器控制输出幅度）。

**（4）实验内容3：**将此信号发生器改成具有扫频功能的波形发生器，扫速可数控，点频扫频可控。

**（5）实验思考题：**如果频率控制字宽度直接用32位，相位控制字宽度直接用10位，输出仍为10位，时钟为20MHz，计算频率、相位和幅度三者分别的步进精度是多少，给出输出频率的上下限。

**（6）实验报告：**根据以上的实验要求、实验内容和实验思考题写出实验报告。

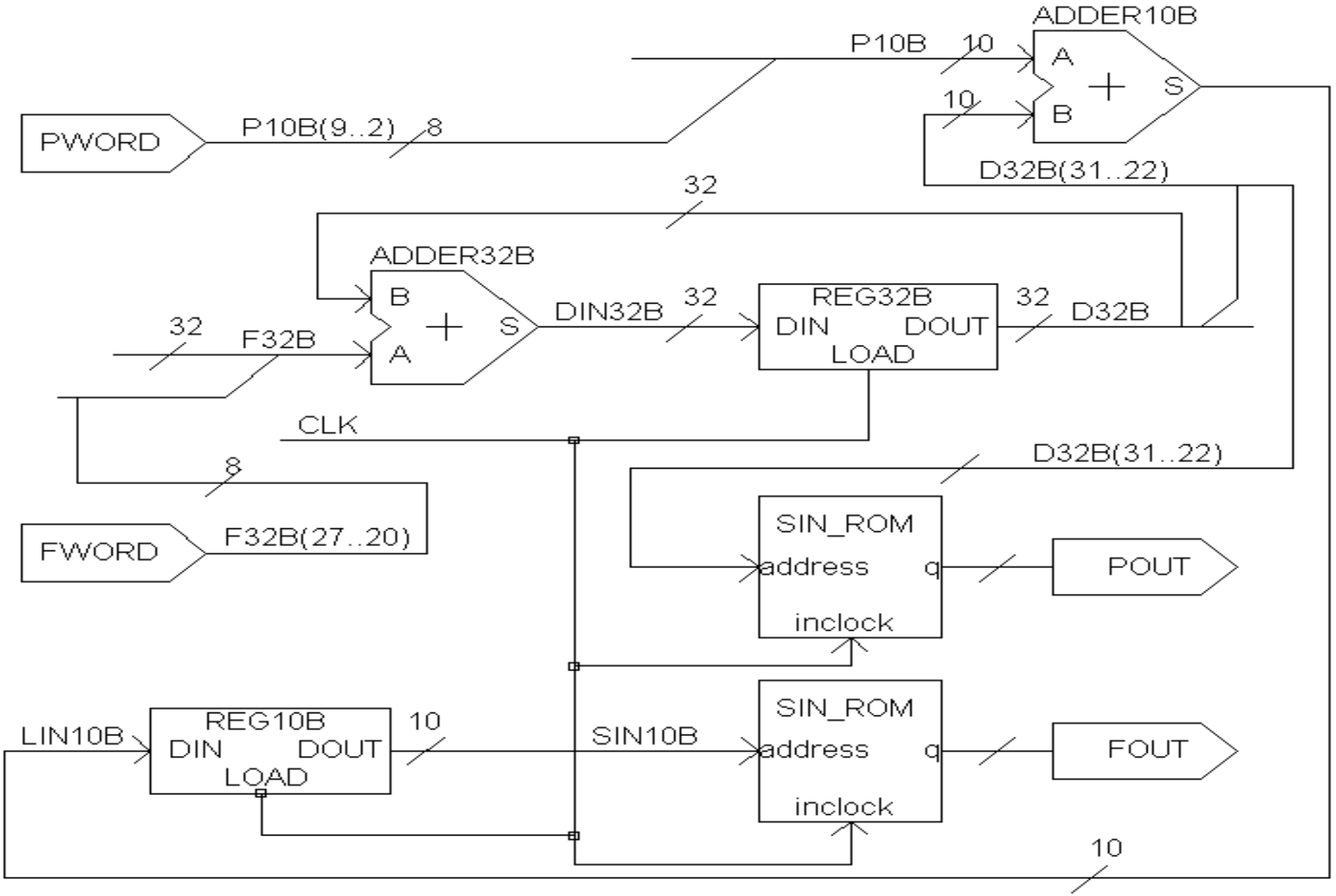



图11-40 数字移相信号发生器电路模型图

【例11-16】数字移相信号发生器顶层设计文件，元件连接结构参考图11-40。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DDS_VHDL IS                                    -- 顶层设计
   PORT (  CLK : IN  STD_LOGIC;  --系统时钟
         FWORD : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);    --频率控制字
         PWORD : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);    --相位控制字
         FOUT  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);    --可移相正弦信号输出
         POUT  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)   ); --参考信号输出
 END;
ARCHITECTURE one OF DDS_VHDL IS
    COMPONENT REG32B                --32位锁存器
      PORT (  LOAD : IN STD_LOGIC;
              DIN : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
             DOUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
   END COMPONENT;
    COMPONENT REG10B                --10位锁存器
      PORT (  LOAD :  IN STD_LOGIC;
              DIN :  IN STD_LOGIC_VECTOR(9 DOWNTO 0);
             DOUT : OUT STD_LOGIC_VECTOR(9 DOWNTO 0) );
   END COMPONENT;
   COMPONENT ADDER32B               --32位加法器
     PORT (  A : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
```

(接下页）

```
B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            S : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)     );
   END COMPONENT;
   COMPONENT ADDER10B                --10位加法器
     PORT (  A : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
             B : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
             S : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)     );
   END COMPONENT;
   COMPONENT SIN_ROM                 --10位地址10位数据正弦信号数据ROM
    PORT ( address   : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                       inclock              : IN STD_LOGIC ;
                    q             : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)        );
   END COMPONENT;
    SIGNAL F32B，D32B，DIN32B    : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL P10B，LIN10B，SIN10B  : STD_LOGIC_VECTOR( 9 DOWNTO 0);
 BEGIN
 F32B(27 DOWNTO 20)<=FWORD ;      F32B(31 DOWNTO 28)<="0000";
 F32B(19 DOWNTO 0)<="00000000000000000000" ;
 P10B( 9 DOWNTO 2)<=PWORD ;      P10B( 1 DOWNTO 0)<="00"  ;
 u1 : ADDER32B  PORT MAP( A=>F32B,B=>D32B, S=>DIN32B );
 u2 :   REG32B  PORT MAP( DOUT=>D32B,DIN=> DIN32B, LOAD=>CLK );
 u3 :  SIN_ROM  PORT MAP( address=>SIN10B, q=>FOUT, inclock=>CLK );
 u4 : ADDER10B  PORT MAP( A=>P10B,B=>D32B(31 DOWNTO 22),S=>LIN10B );
 u5 :   REG10B  PORT MAP( DOUT=>SIN10B,DIN=>LIN10B, LOAD=>CLK );
u6 :  SIN_ROM  PORT MAP( address=>D32B(31 DOWNTO 22), q=>POUT, inclock=>CLK );
END;
```

# 实 验 与 设 计

## 1-5 基于DDS的幅度调制AM信号发生器设计

（1）实验原理：幅度调制信号发生器是2005年大学生电子设计竞赛题中的一个设计项目。

（2）实验内容1：利用MATLAB和DSP Builder，根据图11-41，完成电路模型设计，使产生图11-42的波形，最后在FPGA上硬件实现，并于示波器上验证图11-42的波形。

（3）实验内容2：将图11-41模型完全用VHDL表达出来（顶层设计可以用原理图），直接用QuartusII实现硬件设计，仿真和FPGA实现，比较两种方法的异同点（如输出频率的高低，资源利用等）。

# 实 验 与 设 计

## 1-5 基于DDS的幅度调制AM信号发生器设计

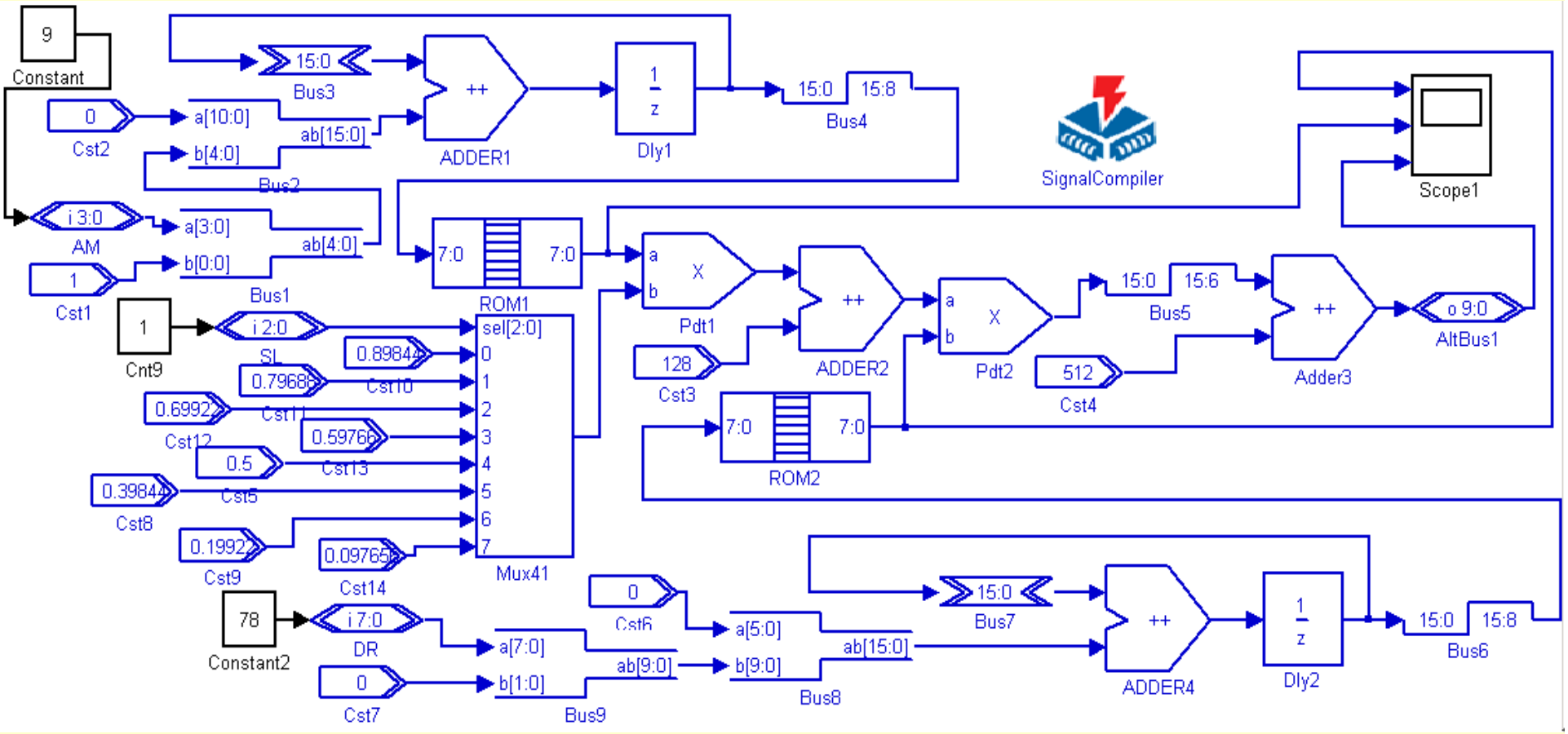



图11-41 AM发生器模型

康芯科技

# 实 验 与 设 计

## 1-5 基于DDS的幅度调制AM信号发生器设计

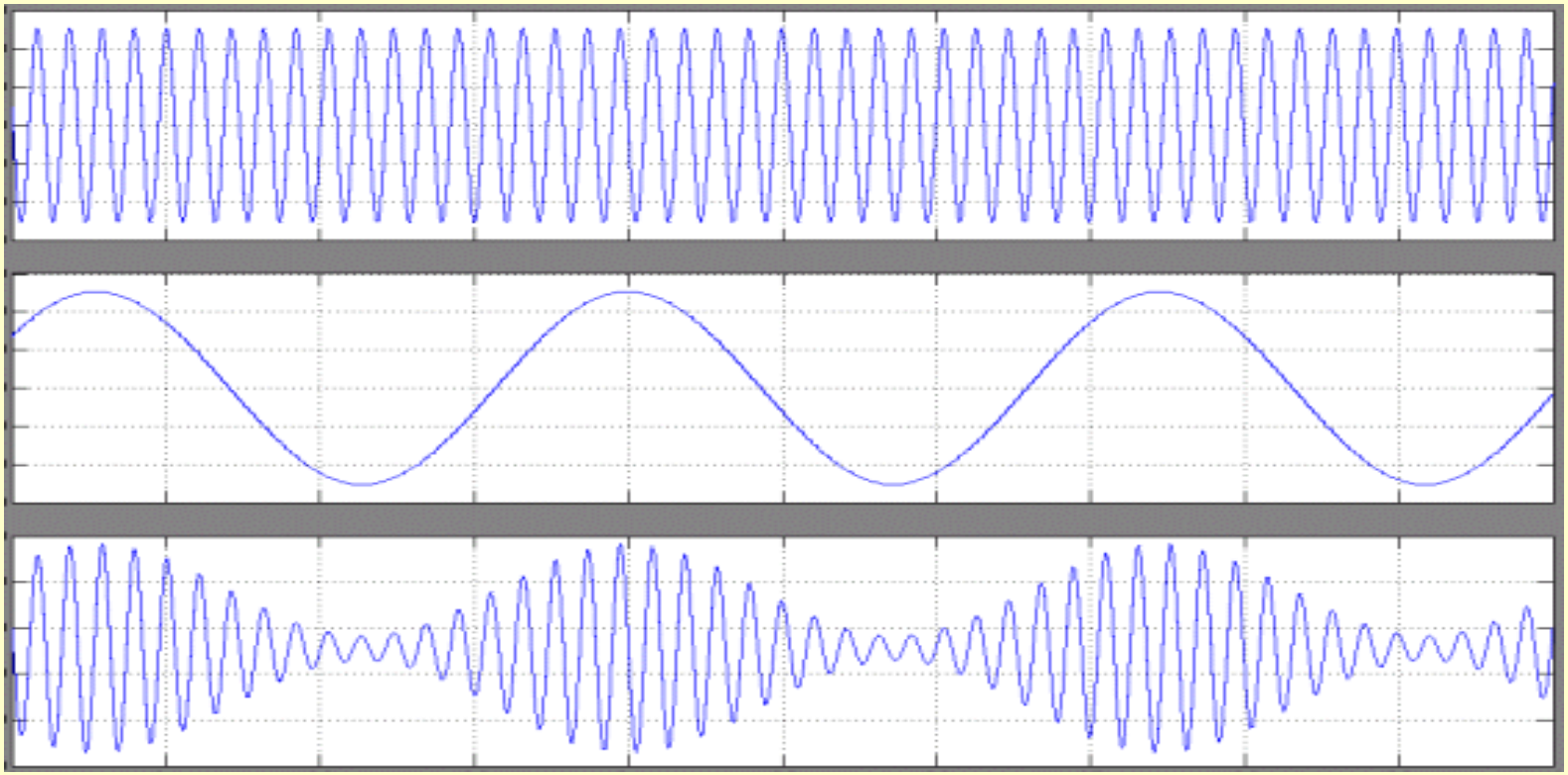


图11-42 AM模型仿真波形

## 11-6. 频率调制FM信号发生器设计

实验内容：根据实验11-15和图11-43，用VHDL设计频率调制信号发生器。要求载波频率、调制波频率和调制度可数控。

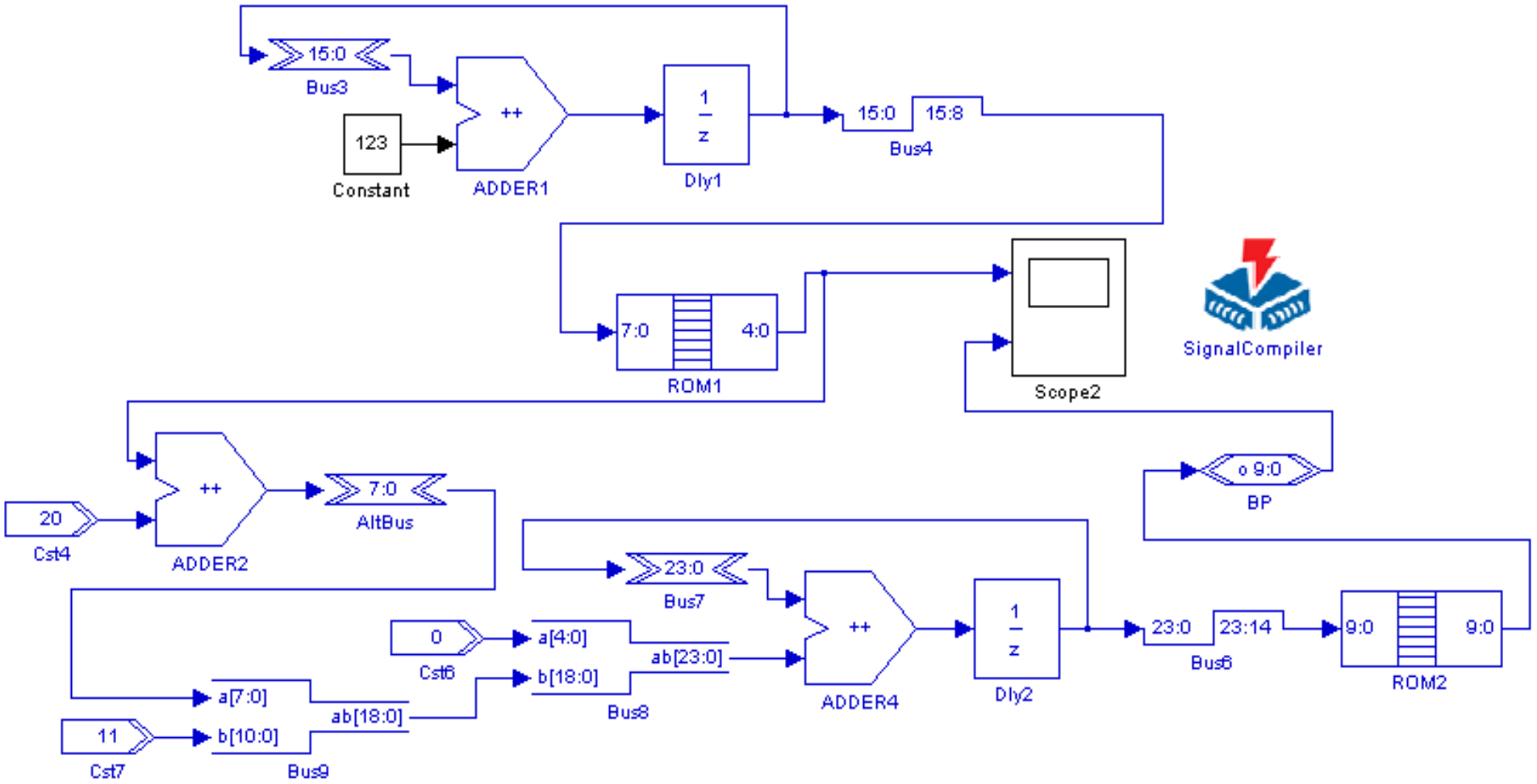



图11-43 频率调制信号发生器MATLAB模型