

VHDL 语言简介及 Quartus II 使用介绍

2006 年秋季学期

数字电子技术基础

目 录

VHDL 语言简介及 Quartus II 使用介绍.....	0
第一章 VHDL 简介.....	1
1.1 VHDL 的特点.....	1
1.2 VHDL 的设计流程.....	1
1.3 VHDL 的基本语法.....	2
1.3.1 VHDL 程序的结构.....	2
1.3.2 数据类型.....	4
1.3.3 数据操作.....	5
1.3.4 并行赋值语句.....	5
1.3.5 进程语句.....	6
1.3.6 元件例化.....	7
1.3.7 注释.....	7
1.4 结构体描述的三种方法.....	7
1.4.1 行为描述法设计举例.....	8
1.4.2 数据流描述法设计举例.....	9
1.4.3 结构描述法设计举例.....	10
1.5 VHDL 的电路设计举例.....	13
1.5.1 组合逻辑电路.....	13
1.5.2 触发器（D 触发器）.....	13
1.5.3 分频器电路.....	14
1.5.4 锁存器（Latch）.....	16
1.5.5 RAM/ROM 的设计.....	16
1.6 状态机设计.....	17
1.6.1 概述.....	17
1.6.2 程序举例.....	18
第二章 Quartus II 的使用.....	23
2.1 Quartus II 概述.....	23
2.2 Quartus II 的 VHDL 输入设计流程.....	24
2.2.1 新建工程.....	24
2.2.2 新建 VHDL 设计文件.....	26
2.2.3 功能仿真.....	28
2.2.4 编译前的一些设置及全编译.....	32
2.2.5 时序仿真.....	35
2.2.6 引脚锁定和下载.....	36
2.3 Quartus II 的原理图输入设计流程.....	39
2.3.1 新建工程和生成元件符号.....	39
2.3.2 新建原理图设计文件.....	39

2.3.3 全编译和时序仿真	42
附录 实验装置介绍	43
1.1 简介	43
1.2 GWAC6L 适配板	43
1.3 GW48-PK2 主板	43
1.3.1 GW48-PK2 主板结构	43
1.3.2 GW48-PK2 主板上部分多模式电路介绍	45
1.4 GW48 系统使用注意事项	46

第一章 VHDL 简介

1.1 VHDL 的特点

随着电子设计技术的高速发展，电路的复杂度越来越高，产品的更新速度越来越快，原理图输入的方法已经不能满足工业界对设计能力的要求。VHDL（Very High Speed Integrated Circuit Hardware Description Language）是美国国防部 1983 年提出的一种硬件描述语言，它可以描述硬件的结构和行为，通过采用 EDA 工具自动综合出电路结构，极大地提高了设计能力。VHDL 设计方法有以下几个优点：

1. 可以直接描述电路的行为，由 EDA 工具综合出电路，设计速度快。
2. 工艺无关性。设计人员不必过多关心具体的工艺，由 EDA 工具自动针对具体的工艺综合出电路。同时设计具有非常高的可移植性，这是原理图输入法不可比拟的。
3. 设计文件可读性好

1.2 VHDL 的设计流程

CPLD/FPGA 的 VHDL 设计流程如图 1.2.1 所示。

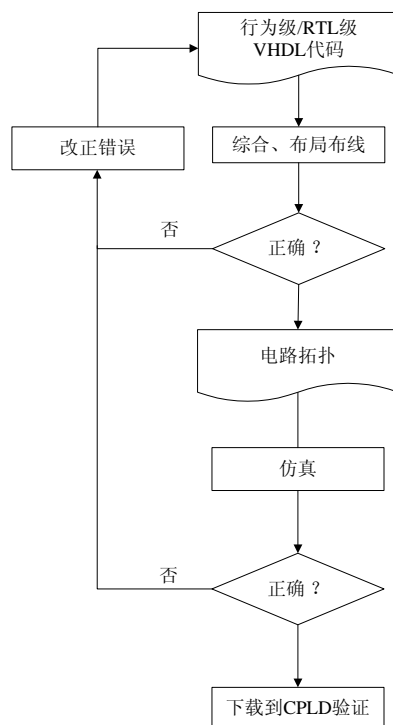


图 1.2.1 CPLD/FPGA 设计流程

设计流程主要包括以下几步：

1. 写设计文件。按照自顶向下的方法将系统分解为不同的模块，采用行为描述或结构描述的方法设计各个模块。
2. 综合、布局布线。由 EDA 工具根据具体的 CPLD 工艺，编译设计文件，产生电路结构，并完成布局布线，最后产生可下载到 CPLD 的数据文件。
3. 仿真。在计算机上对 EDA 工具产生的电路进行模拟，验证电路的功能、时序是否达到设计要求。
4. 下载到 CPLD 验证。仿真验证了设计的功能正确后，最后下载到 CPLD 芯片中，配合外围电路验证整个系统的功能。

1.3 VHDL 的基本语法

VHDL 语言是一种比较复杂的语言，它可以在不同的抽象层次描述一个电路，这里仅介绍实验所需要的最基本的语法。

1.3.1 VHDL 程序的结构

程序举例：

```
LIBRARY ieee;           -- 库定义
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
ENTITY counter IS       -- 实体定义
    PORT (
        A,B,CLK : in    std_logic;
        Q : out   std_logic );
END;

ARCHITECTURE behav OF counter IS
-- 结构体定义
    SIGNAL    D, E: std_logic;
BEGIN
    E <= A and B;
    PROCESS (CLK)
    BEGIN
        IF CLK'event and CLK='1' THEN
            D <= E;
        END IF;
        Q <= D;
    END PROCESS;
END behav ;
```

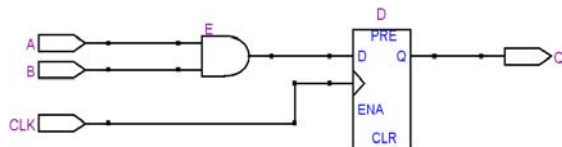


图 1.3.1 VHDL 程序对应原理图

上例是一个基本的 VHDL 程序,它包括三个基本部分:

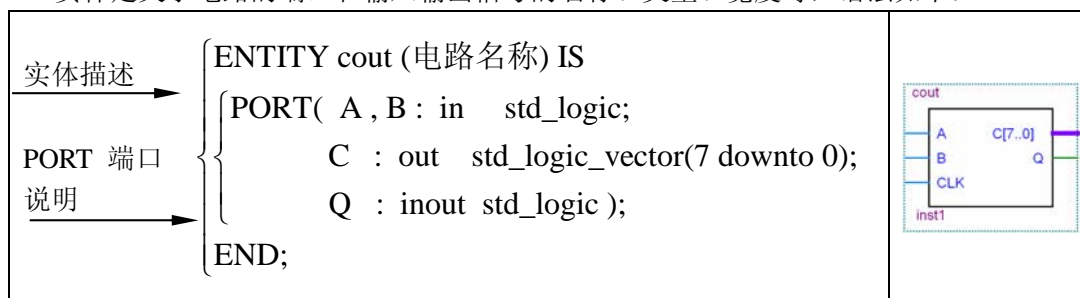
1. 库

库类似 C 语言的头文件，在库里定义了一些常用的数据类型、函数等，一般使用以下的库就够了：

```
LIBRARY ieee;           --标准库资源
use ieee.std_logic_1164.all;  --标准逻辑程序包
use ieee.std_logic_unsigned.all;
```

2. 实体

实体定义了电路的端口和输入输出信号的名称、类型、宽度等，语法如下：



关键词解释：

- in: 输入端口。
- out: 输出端口。
- inout: 输入输出双向端口。
- std_logic: 标准逻辑位数据类型。
- std_logic_vector: 标准逻辑矢量数据类型。括号内的语句是定义位宽用的，推荐将位宽写成 (M downto N) 的方式，上例中信号 C 就是一个 8 位位宽的总线端口信号。

3. 结构体

结构体定义了电路的内部结构，包括电路内部的信号，各个模块的结构描述和行为描述，语法如下：

```
ARCHITECTURE behave (结构名) OF cout (实体名) IS
    SIGNAL    E (内部信号): std_logic;
BEGIN
    模块 1
    模块 2
    .....
END;
```

- 结构体名由设计者自由命名，是结构体的唯一名称，在 VHDL 设计程序中只使用一次；OF 后面的实体名称表明该结构体属于那个设计实体，在设计实体中可能包含多个结构体。设计时可以根据结构体的特色来为它命名。例如：

```
ARCHITECTURE behave OF cout IS      突出结构体的行为特色
ARCHITECTURE dataflow OF cout IS    突出结构体的数据流特色
ARCHITECTURE structural OF cout IS  突出结构体的组织结构特色
ARCHITECTURE bool OF cout IS        突出结构体的数学表达方式特色
.....
```

- 结构体中内部信号的定义方式和端口的定义方式一样。比如 1.3.1 节程序中的——
SIGNAL D, E : std_logic; 语句。
- 结构体中的模块，实现各种功能。例 1.3.1 节中程序是一个结构描述类型的与门模块，一个是行为描述类型的 D 触发器模块。

1.3.2 数据类型

VHDL 支持多种数据类型，常用类型有：

1. 9 值逻辑 std_logic

它把信号线上的电平描述成 '0', '1', 'U', 'X', 'Z', 'W', 'L', 'H', '-' 9 种数据。其中 '0' 表示逻辑 0; '1' 表示逻辑 1; 'Z' 表示高阻态; 'U' 表示未初始化; 'X' 表示未知的; '-' 表示忽略。

2. 矢量类型 std_logic_vector

该矢量类型也是 9 值逻辑，使用时必须指定矢量的宽度，一般从高到低，例如：

```
signal x: std_logic_vector(3 downto 0);
```

表示 x 是一个四位总线，由 x(3),x(2),x(1),x(0) 构成。

3. 整型 integer

整型信号主要用作状态信号、计数信号或数组的下标，使用时必须指定数值范围，例如：

```
signal y: integer range 0 to 15;
```

矢量信号可以通过转换函数 CONV_INTEGER 转换成整形，例如：

```
y <= CONV_INTEGER(x);
```

4. 自定义数据类型

VHDL 允许用户自行定义新的数据类型。自定义数据类型是用类型定义语句 TYPE 实现的。TYPE 的规范书写格式为：

```
TYPE 数据类型名 IS 数据类型定义 ;
```

1) 枚举类型

枚举类型就是把数据类型中的各个元素都列举出来，方便、直观提高了程序可阅读性。枚举类型书写格式如下：

```
TYPE 数据类型名 IS (元素, 元素, ……);
```

例状态机设计方式内的枚举类型：

```
TYPE STATE IS (S0,S1,S2,S3);
```

2) 整数类型、实数类型

整数类型、实数类型在 VHDL 语言标准中已定义，有时出于设计者的要求需要自己定义数据类型。整数类型和实数类型用户定义的书写格式为：

```
TYPE 数据类型名 IS 数据类型定义 约束范围 ;
```

例在七段数码管控制设计中，一位数码管的数据类型应写为：

```
TYPE digit IS INTEGER range 0 TO 9;
```

3) 数组类型

VHDL 程序设计中的数组类型是指将相同类型的数据集合在一起形成一个新的数据类型。数组类型的书写格式为：

```
TYPE 数据类型名 IS ARRAY (INTEGER 0 TO 9) OF STD_LOGIC;
```

例如在使用 RAM、ROM 或寄存器堆时需要定义数组，语法如下：

```
type instr is array (0 to 15) of std_logic_vector(7 downto 0);  
signal IIRAM: instr;
```

上例首先定义了 instr 数据类型是一个 8 位的矢量数组，数组容量是 16；然后用该数据类型定义了信号 IIRAM。因此 IIRAM 信号实际上是一个容量为 16 的 8 位数组，可以作为 RAM、ROM 或寄存器堆。

用户定义的数据类型还有记录类型、文件类型、文件类型、存取类型，因为不常使用，这里不作介绍。

1.3.3 数据操作

1. 赋值 (<=)

相同数据类型的信号可以赋值，结构体内的赋值语句相当于信号线的连接。例如：

```
SIGNAL D, E: std_logic;  
BEGIN  
    E <= A and B;  
  
    PROCESS (CLK)  
    BEGIN  
  
        IF CLK'event and CLK='1' THEN  
            D <= E;  
        END IF;  
        Q <= D;  
    END PROCESS;
```

上述语句表示了这些信号线之间的连接关系。可参见图 2-2b。

2. 逻辑操作 (AND、OR、NOT、NAND、NOR、XOR、XNOR)

逻辑操作可以描述信号的逻辑关系，例如：

```
y <= not (x(1) nand x(2) or x(3));
```

3. 比较操作 (=、/=、<、>、<=、>=)

比较操作常用于条件赋值语句和条件分支语句，在以后详细介绍。其中 /= 表示“不等于”。

4. 拼接操作 (&)

拼接操作可以把两个信号拼接成一个新的矢量信号，例如：

```
x(3 downto 2) <= x(1) & x(0);
```

1.3.4 并行赋值语句

当赋值语句直接出现在结构体中时表示并行赋值语句，它直接表示了信号的连接关系或描述了一个电路模块（组合逻辑电路）。所有的语句在仿真时都是并行执行的。语法如下：

```
赋值目标 <= 表达式 WHEN 赋值条件 ELSE  
            表达式 WHEN 赋值条件 ELSE  
            .....  
            表达式 ;
```


1.3.5 进程语句

进程语句定义了一个电路模块，进程内部是模块的行为描述。语法如下：

```
[进程名 :] PROCESS (信号 1, 信号 2, ...)  
BEGIN  
    行为描述语句;  
END PROCESS [进程名];
```

其中 **PROCESS** 语句后的信号列表指进程的敏感表，一般情况下把整个 **PROCESS** 用到的所有输入信号都列到敏感表中，除非该进程是一个边沿触发的进程。和并行赋值语句不同，在进程内部的行为描述语句在仿真时都是顺序执行的。这里仅介绍条件分支语句和 **CASE** 语句,它们的语法如下：

1. 条件分支语句的 3 种类型：

1) 开关控制的 IF 语句

书写格式： if 条件 then 语句; end if;	程序举例： IF CLK'event and CLK='1' THEN D <= E; END IF; Q <= D;
---------------------------------------	--

2) 二选一控制的 IF 语句

书写格式： If 条件 then 语句; else 语句; end if;	程序举例： if LOAD = '0' then Q <= Q+1; else Q <= QLOAD; End if;
--	--

3) 多选择控制的 IF 语句

书写格式： if 条件 1 then 语句; elsif 条件 2 then 语句; else 语句; end if;	程序举例： if (D(0) = '0') then Q <= "11"; elsif (D(1) = '0') then Q <= "10"; elsif (D(2) = '0') then Q <= "01"; elsif Q <= "00"; end if;
--	---

2. CASE 分支语句语法：

书写格式： case 信号 is when 值 1 => 语句;	程序举例： case OP is when "00" => Q <= Q+1; when "01" "10" =>
---	---

when 值 2 值 3 ... => 语句; when others => 语句; end case;	Q <= QLOAD; when others => Q <= "0000"; end case;
--	--

1.3.6 元件例化

元件例化就是引入一种连接关系，将预先设计好的设计实体定义为一个元件，然后利用特定的语句将此元件与当前的设计实体中的指定端口相连接，从而实现结构化设计。语法如下：

第一部分：元件定义语句

```

COMPONENT 元件名 IS
    PORT ( 信号 1, 信号 2 : in      std_logic;
           信号 3          : out    std_logic_vector(7 downto 0);
           信号 4, 信号 5 : inout  std_logic );
END COMPONENT;

```

第二部分：连接说明语句

例化名：元件名 PORT MAP (连接端口名 1, 连接端口名 2, ……);

语法举例：

```

COMPONENT or_gate
    PORT( a,b : IN std_logic;
           c : OUT std_logic);
END COMPONENT;

BEGIN
    U0: or_gate  PORT MAP((tmp3,tmp2,Co_F);

```

1.3.7 注释

VHDL 程序的注释行从“--”开始，可以位于程序的任何位置。

1.4 结构体描述的三种方法

1. 行为描述法：对设计实体的描述按算法的路径来描述，其抽象程度远远高于数据流描述方式和结构描述方式，在 EDA 工程中称为高层次描述或高级描述。

特点：只需要描述清楚输入与输出的行为，不需要关注设计功能的门级实现。

2. 数据流描述法：是结构描述方法之一。对设计实体的描述是按照从信号到信号的寄存器传输的路径形式来进行的，也称为寄存器传输描述方式。

特点：易于进行逻辑综合。但是需要对设计电路有较深入的了解。

3. 结构描述法：指在多层次的设计中，通过调用库中的元件或是已设计好的模块来完成设计实体功能的描述。

特点：结构清晰。

上述 3 种电路在设计电路时可以混合使用。

1.4.1 行为描述法设计举例

用行为描述法设计一个 4 选 1 数据选择器。通过给定不同的地址代码，即可从 4 个输入数据中选出所要的一个，并送至输出端 Y。输出逻辑式和真值表如下：

$$Y=A(\bar{S}_1\bar{S}_0)+B(\bar{S}_1S_0)+C(S_1\bar{S}_0)+D(S_1S_0) \quad (2.4.1)$$

真值表：

表 1.4.1 4 选 1 数据选择器真值表

输入		输出 Y
0	0	A
0	1	B
1	0	C
1	1	D

--VHDL 程序：

```

LIBRARY IEEE; --标准库资源
USE IEEE.STD_LOGIC_1164.ALL; --标准逻辑程序包
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY mux_4_1 IS
    PORT (
        S: IN STD_LOGIC_VECTOR( 1 DOWNT0 0 );--数据选择端口
        A, B, C, D: IN STD_LOGIC; --输入端口
        Y: OUT STD_LOGIC );--输出端口
END mux_4_1;
ARCHITECTURE mux_behave OF mux_4_1 IS
BEGIN
    Y<= A WHEN S="00" ELSE
        B WHEN S="01" ELSE
        C WHEN S="10" ELSE
        D WHEN S="11" ELSE
        '0';
END mux_behave;

```

使用行为描述法设计的 4 选 1 数据选择器的 RTL 电路图见图 1.4.1。

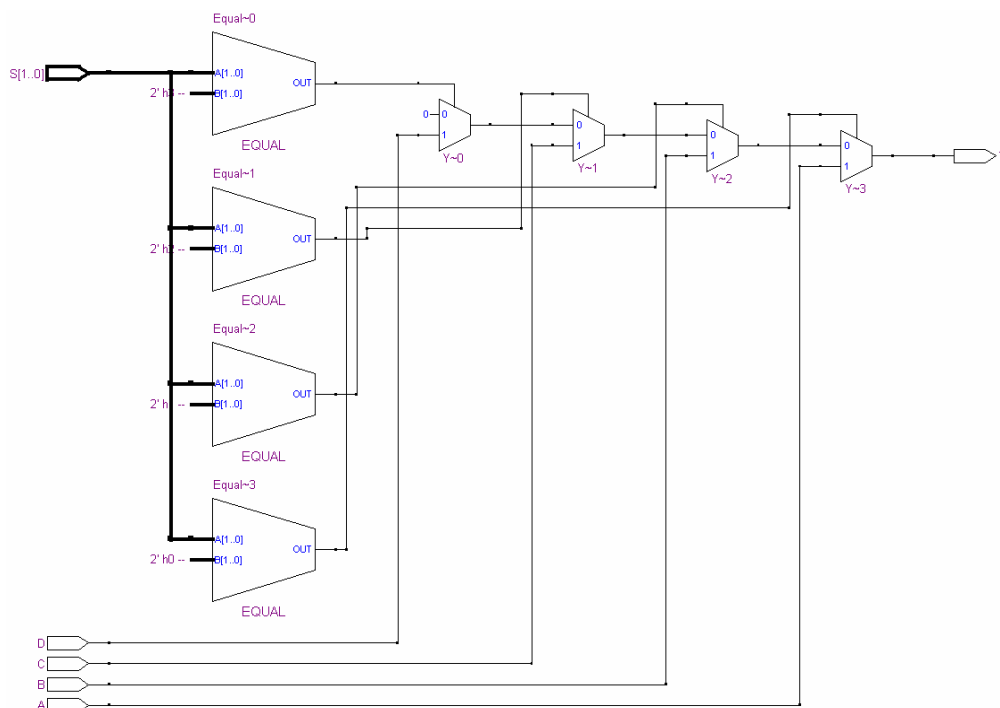


图 1.4.1 使用行为描述法设计的 4 选 1 数据选择器

试将上个程序中部分语句改为：

```
Y<= A WHEN S="00" ELSE
    B WHEN S="01" ELSE
    C WHEN S="10" ELSE
    '0' ;
```

后看 RTL 电路图结果，并与图 1.4.1 比较。

1.4.2 数据流描述法设计举例

用数据流描述法设计一个 4 选 1 数据选择器。电路要求见 1.4.1。

--VHDL 程序：

```
LIBRARY ieee;                --标准库资源
USE ieee.std_logic_1164.ALL;  --标准逻辑程序包
ENTITY mux_41 IS
    PORT(
        S      : IN   STD_LOGIC_VECTOR(1 DOWNTO 0); --数据选择端口
        A,B,C,D : IN   STD_LOGIC;                  --输入端口
        Y       : OUT  STD_LOGIC );                 --输出端口
END mux_41;
ARCHITECTURE mux_behave OF mux_41 IS
BEGIN
    Y <= ( NOT S(1) AND NOT S(0) AND A ) OR ( NOT S(1) AND S(0) AND B )
        OR ( S(1) AND NOT S(0) AND C ) OR ( S(1) AND S(0) AND D );
END mux_behave ;
```

数据流描述法设计的 4 选 1 数据选择器的 RTL 电路图见图 1.4.2。

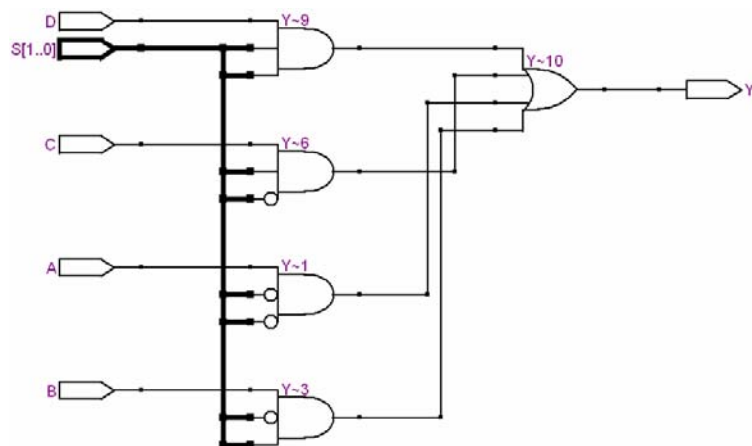


图 1.4.2 使用数据流描述法设计的 4 选 1 数据选择器

两种电路的时序分析结果见图 1.4.3 和图 1.4.4。

Registered Performance tpd tsu tco th Custom Delays						
	Slack	Required P2P Time	Actual P2P Time	From	To	
1	N/A	None	11.408 ns	B	Y	
2	N/A	None	11.323 ns	D	Y	
3	N/A	None	11.117 ns	S[0]	Y	
4	N/A	None	10.981 ns	S[1]	Y	
5	N/A	None	10.136 ns	A	Y	
6	N/A	None	9.950 ns	C	Y	

图 1.4.3 行为描述法设计的 4 选 1 数据选择器时序分析

Registered Performance tpd tsu tco th Custom Delays						
	Slack	Required P2P Time	Actual P2P Time	From	To	
1	N/A	None	11.408 ns	C	Y	
2	N/A	None	11.323 ns	D	Y	
3	N/A	None	11.117 ns	S[1]	Y	
4	N/A	None	10.981 ns	S[0]	Y	
5	N/A	None	10.136 ns	A	Y	
6	N/A	None	9.950 ns	B	Y	

图 1.4.4 数据流描述法设计的 4 选 1 数据选择器时序分析

1.4.3 结构描述法设计举例

用结构描述法设计一位全加器。要求，使用两个半加器和一个二输入或门实现。

首先，根据二进制加法运算规则列出半加器真值表见表 1.4.2。其中 A、B 是两个加数，S 是相加的和，CO 是向高位的进位。将 S、CO 和 A、B 的关系写成逻辑表达式则得到

$$\begin{cases} S = \bar{A}B + A\bar{B} \\ C_o = AB \end{cases} \quad (2.4.2)$$

表 1.4.2 半加器真值表

输 入		输 出	
A	B	S	C _o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

--半加器的 VHDL 语言程序

--输入端口为 A、B

--输出端口为 S、C_o(进位)

LIBRARY IEEE;

--标准库资源

USE IEEE.std_logic_1164.ALL;

--标准逻辑程序包

ENTITY half_adder IS

PORT (A, B : IN std_logic;

Co : OUT std_logic;

S : OUT std_logic);

END half_adder;

ARCHITECTURE rtl OF half_adder IS

BEGIN

S <= A XOR B;

Co <= A AND B;

END rtl;

根据二进制加法运算规则列出一位全加器真值表见表 1.4.3。C₁ 是低位的进位。将 S、C_o、和 A、B、C₁ 的关系写成逻辑表达式则得到

$$\begin{cases} S = \bar{A}\bar{B}C_1 + \bar{A}B\bar{C}_1 + A\bar{B}\bar{C}_1 + ABC_1 \\ C_o = \bar{A}BC_1 + A\bar{B}C_1 + AB\bar{C}_1 + ABC_1 \end{cases} \quad (2.4.3)$$

表 1.4.3 全加器真值表

输 入			输 出	
C ₁	A	B	S	C _o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

对逻辑表达式 (2.4.3) 化简，实现题目要求。化简过程和结果见下

$$\begin{cases} S = \bar{A}\bar{B}C_1 + \bar{A}B\bar{C}_1 + A\bar{B}\bar{C}_1 + ABC_1 = (\bar{A}B + A\bar{B})\bar{C}_1 + (\bar{A}\bar{B}_1 + AB)C_1 \\ \quad = A \oplus B \oplus C_1 \\ C_o = \bar{A}BC_1 + A\bar{B}C_1 + AB\bar{C}_1 + ABC_1 = (\bar{A}B + A\bar{B})C_1 + AB(\bar{C}_1 + C_1) \\ \quad = (\bar{A}B + A\bar{B})C_1 + AB = (A \oplus B)C_1 + AB \end{cases} \quad (2.4.4)$$

--一位全加器的 VHDL 语言程序（结构化描述）

```

LIBRARY IEEE;                                --标准库资源
USE IEEE.std_logic_1164.ALL;                  --标准逻辑程序包
ENTITY full_adder2 IS
PORT (  A, B : IN      std_logic;             --输入端口为 A、B、Cin(预进位)
      Cin : IN      std_logic;
      S_F : OUT     std_logic;                --输出端口为 S_F、Co_F (进位)
      Co_F : OUT     std_logic);
END full_adder2;
ARCHITECTURE structure OF full_adder2 IS
SIGNAL tmp1,tmp2,tmp3 : std_logic;            --定义信号线数据类型
  COMPONENT half_adder                        --将半加器定义为一个元件
    PORT( A,B : IN std_logic;
          S : OUT std_logic;
          Co : OUT std_logic );
  END COMPONENT;
  COMPONENT or_gate                            --将或门定义为一个元件
    PORT( a,b : IN std_logic;
          c : OUT std_logic);
  END COMPONENT;
BEGIN
  U0: half_adder    PORT MAP(A,B,tmp1,tmp2);    --调用半加器模块
  U1: half_adder    PORT MAP(tmp1,Cin,S_F,tmp3);
  U2: or_gate       PORT MAP(tmp3,tmp2,Co_F);    --调用或门模块
END structure;

```

Quartus II 综合后的 RTL 电路图见图 1.4.5。

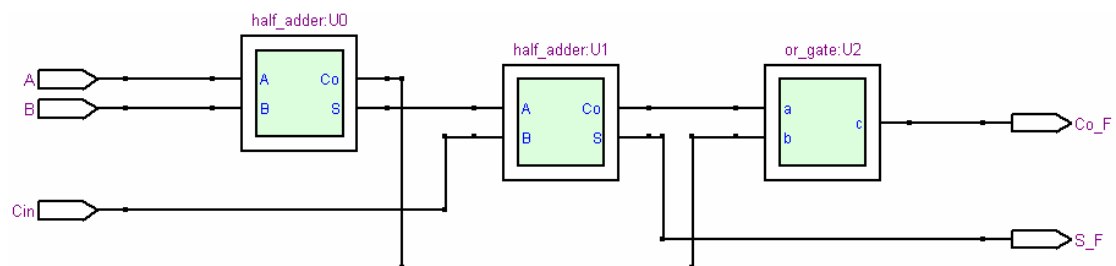


图 1.4.5a 一位全加器 RTL 电路图

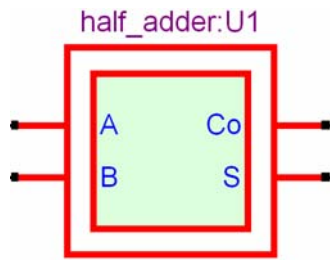


图 1.4.5b 全加器

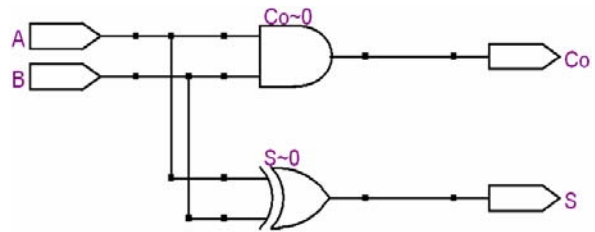


图 1.4.5c 全加器下的半加器 RTL 电路图

1.5 VHDL 的电路设计举例

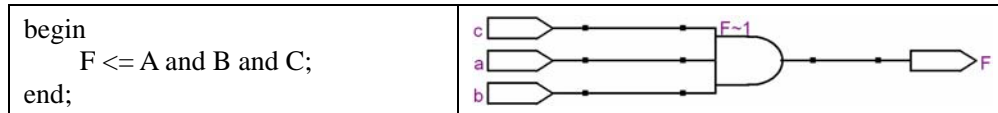
学习结构描述法后知道任何一个电路都可以分解为很多模块的互连，这些模块可以用 VHDL 语言在同一个结构体中分别用并行赋值语句或进程来描述。下面将介绍几种电路单元的 VHDL 描述。

1.5.1 组合逻辑电路

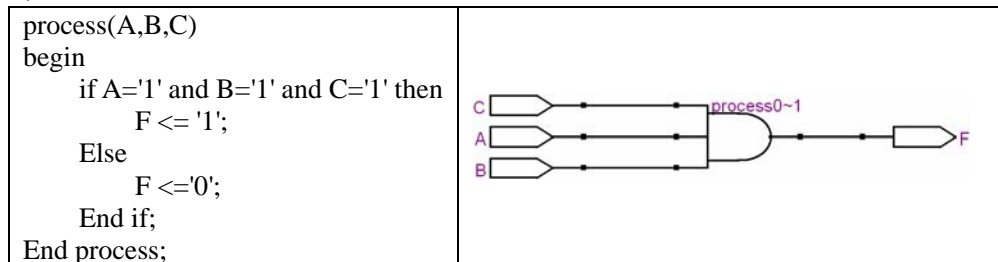
简单组合逻辑电路既可以使用并行赋值语句，也可以用进程描述。简单组合逻辑电路一般可以用一个布尔方程表示，例如三输入与门： $F = A \cdot B \cdot C$

并行赋值语句：

写法 1：



写法 2：



并行赋值语句适用于可以用简洁的布尔方程表示的组合逻辑电路。如果逻辑关系复杂，难以用布尔方程表示，则可以用进程实现，在进程中用条件分支语句和 CASE 语句直接描述电路的逻辑功能。

注意：要实现组合逻辑电路，所有的条件分支必须写全。

1.5.2 触发器（D 触发器）

D 触发器的功能描述如下：

1. 具有异步复位（置位）端，当复位信号有效时，输出端被复位或置位。复位（置位）

端的优先级比时钟端高。

- 2. 在时钟信号的上升沿（下降沿）把输入数据打到输出端。
- 3. 其它时候输出保持不变。

D 触发器只能用进程实现, 在进程的敏感表中只需有异步复位和时钟信号。下面的例子是一个典型的 D 触发器，它有两个分支分别描述触发器的两个功能，在其它情况下 Q 保持不变。

```
PROCESS (RST,CLK)
BEGIN
  if RST = '1' then
    Q <= '0';
  elsif CLK'event and CLK = '1' then
    Q <= D;
  end if;
END PROCESS;
```

1.5.3 分频器电路

多个触发器串接，除了具有计数功能外，还有分频功能，也就是将高频率的时钟降为低频率的时钟。本例介绍一个使用十六进制计数器实现的分频器电路。电路转换表见表 1.5.1。

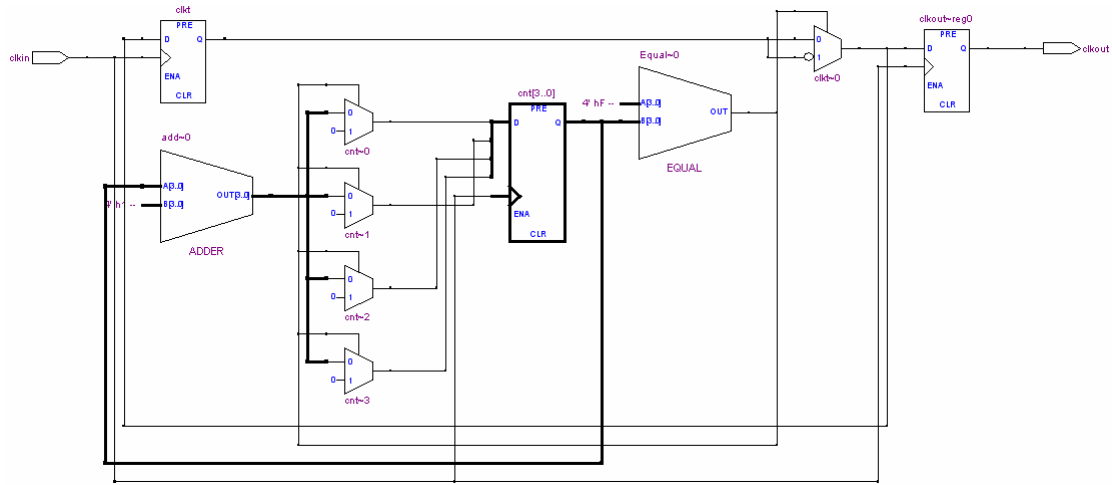
表 1.5.1 电路动态转换表

计数顺序	电路状态				等效十进制数	C _{out}
	Q ₃	Q ₂	Q ₁	Q ₀		
0	0	0	0	0	0	0
1	0	0	0	1	1	0
2	0	0	1	0	2	0
3	0	0	1	1	3	0
4	0	1	0	0	4	0
5	0	1	0	1	5	0
6	0	1	1	0	6	0
7	0	1	1	1	7	0
8	1	0	0	0	8	0
9	1	0	0	1	9	0
10	1	0	1	0	10	0
11	1	0	1	1	11	0
12	1	1	0	0	12	0
13	1	1	0	1	13	0
14	1	1	1	0	14	0
15	1	1	1	1	15	1
16	0	0	0	0	0	0

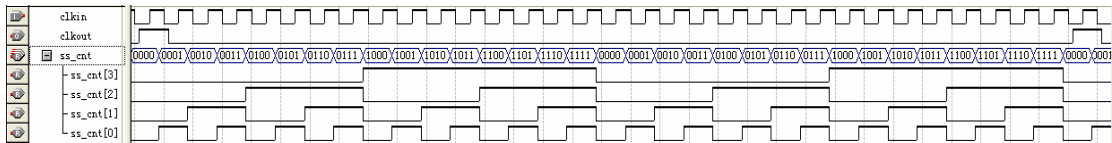
下段程序是通过改变整型变量 LEN 的值,将输入时钟 clkin 的频率分成相应的时钟频率。

```
LIBRARY IEEE;                                --标准库资源
USE IEEE.std_logic_1164.all;                  --标准逻辑程序包
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity divider16 is
    generic(LEN : integer := 16);              --整型 16 位
    port(
        clkin : in    std_logic;
        clkout : out   std_logic );
end divider16;
architecture beh of divider16 is
    signal s_cnt : integer ;                    --定义信号线数据类型
    signal ss_cnt : std_logic_vector(3 downto 0) ;
begin
    process(clkin)
        variable cnt : integer range 0 to LEN - 1;    --cnt 定义为整型变量
        variable clkt : std_logic ;
    begin
        if rising_edge(clkin) then
            if cnt = LEN - 1 then
                if clkt = '1' then
                    clkt := '0' ;
                else
                    clkt := '1' ;
                end if;
                cnt := 0 ;
            else
                cnt := cnt + 1;
            end if;
            s_cnt <= cnt ;
            ss_cnt <= conv_std_logic_vector(s_cnt,4) ;    --函数转换。将 integer 转换
为 std_logic_vector 型。
            clkout <= clkt and ss_cnt(3) and ss_cnt(2) and ss_cnt(1) and ss_cnt(0) ;
        end if;
    end process;
end beh;
```

分频器的 RTL 电路图见图 1.5.1, 电路仿真结果见图 1.5.2。



1.5.1



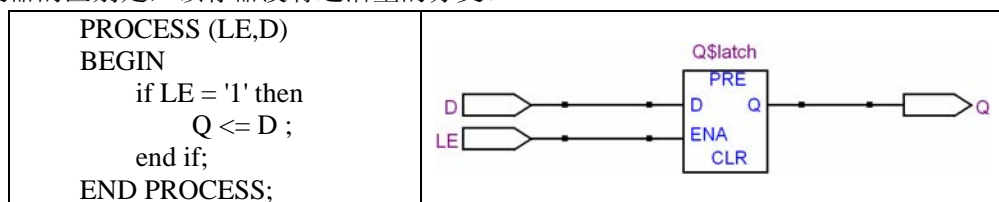
1.5.2

1.5.4 锁存器 (Latch)

锁存器的功能描述如下：

1. 使能端有效时，输入信号被输出到输出端。
2. 使能端无效时输出保持不变。

锁存器只能用进程实现, 在进程的敏感表中应包括所有用到的输入信号。下面的例子是一个典型的锁存器，它用一个分支描述锁存器的锁存功能，在其它情况下 Q 保持不变。和 D 触发器的区别是，锁存器没有边沿型的分支。



1.5.5 RAM/ROM 的设计

RAM/ROM 的设计比较复杂，在 VHDL 程序设计时一般通过调用宏功能模块来实现其功能。这里不做介绍，读者请参考相关资料。

1.6 状态机设计

1.6.1 概述

状态机是指用输入信号和电路状态（状态变量）的逻辑函数去描述时序逻辑电路功能的方法也叫时序机。有限状态机是指在设计电路中加入一定的限制条件。

时序逻辑电路（简称时序电路）的特点是任一时刻的输出信号不仅取决于当时的输入信号，而且还取决于电路原来的状态，或者说，还与以前的输入有关。

时序电路在电路结构上有两个显著的特点：

1. 时序电路通常包含组合电路和存储电路两个组成部分，而存储电路是必不可少的。
2. 存储电路的输出状态必须反馈到组合电路的输入端，与输入信号一起，共同决定组合电路的输出。

根据输出信号的特点可将时序电路划为 Mealy 型和 Moore 型两种。

- Moore 型电路中，输出信号仅仅取决于存储电路的状态。其状态转换图见图 1.5.3a。
- Mealy 型电路中，输出信号不仅取决于存储电路的状态，而且还取决于输入变量。

其状态转换图见图 1.5.3b。

Moore 型状态机可能要比相应的 Mealy 型状态机需要更多的状态。Moore 型有限状态机的输出与当前的输入部分无关，因此当前输入产生的任何效果将会延迟到下一个时钟周期。可见，Moore 型状态机的最大优点就是可以将输入部分和输出部分隔离开。对于 Mealy 型有限状态机来说，由于它的输出是输入信号的函数，因此如果输入信号发生改变，那么输出可以在一个时钟周期的中间发生改变。

有限状态机一般用来实现数字系统设计中的控制部分。

状态机与控制单元的对应关系为有限状态机中的每一个状态对应于控制单元的一个控制步；它的次态和输出对应于控制单元中与每一个控制步有关的转移条件。

有限状态机的描述方式分为三进程、双进程和单进程三种描述方式。

三进程描述方式：是指在 VHDL 语言程序的结构体中，使用三个进程语句来描述有限状态机的功能。一个进程用来描述有限状态机中的次态逻辑；一个进程用来描述有限状态机中的状态寄存器；另外一个进程用来描述有限状态机中的输出逻辑。

双进程描述方式：是指在 VHDL 语言程序的结构体中，使用两个进程语句来描述有限状态机的功能。一个进程语句用来描述有限状态机中次态逻辑、状态寄存器和输出逻辑中的任何两个；另外一个进程则用来描述有限状态机剩余的功能。

单进程描述方式：是指在 VHDL 语言程序的结构体中，使用一个进程语句来描述有限状态机中的次态逻辑、状态寄存器和输出逻辑。

在下面的程序中使用的是双进程描述方式。第一个进程负责状态转化，在 CP 上升沿到达时，当前状态（PresetState）向下一个状态（NextState）的转换；第二个进程负责检测输入信号（DIN）和当前状态（PresetState）的值，并由 CASE-WHEN 语句决定输出信号（OP）和下一个状态值（NextState）的值。

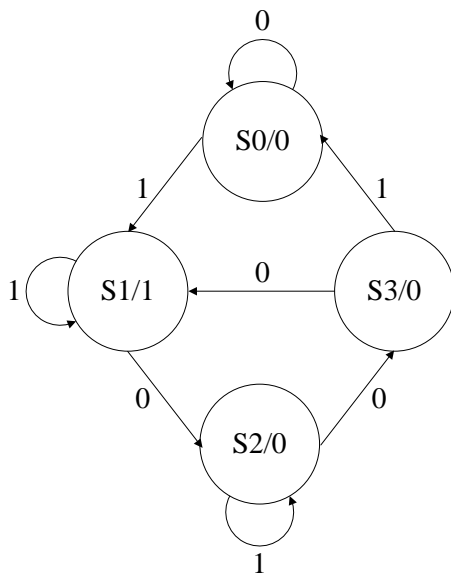


图 1.5.3a Moore 状态机

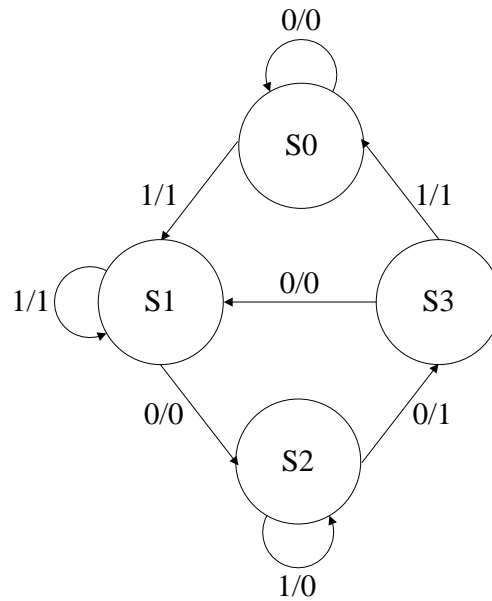


图 1.5.3b Mealy 状态机

2.6.2 程序举例

--Moore 状态机 VHDL 程序

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;      --标准库资源
USE IEEE.STD_LOGIC_ARITH.ALL;      --标准逻辑程序包
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Moore IS
    PORT(
        CP      : IN    STD_LOGIC;
        DIN      : IN    STD_LOGIC;
        OP       : OUT   STD_LOGIC );
END Moore;
ARCHITECTURE behave OF Moore IS
    TYPE STATE IS (S0,S1,S2,S3);    -- State Type Declare
    SIGNAL PresentState : STATE;      -- Present State
    SIGNAL NextState : STATE;         -- Next State
BEGIN
    SwitchToNextState : Process (CP)  -- PresentState -> NextState
    BEGIN
        IF CP'EVENT AND CP = '1' THEN
            PresentState <= NextState;
        END IF;
    END PROCESS SwitchToNextState;
    ChangeStateMode : PROCESS (DIN,PresentState)
    BEGIN
        CASE PresentState IS
    
```

```

        WHEN S0 =>                                --STATE S0
            IF DIN = '0' THEN                      --INPUT=0
                NextState <= S0;
            ELSE
                NextState <= S1;
            END IF;
            OP <= '0';                               --OUTPUT
        WHEN S1 =>                                --STATE S1
            IF DIN = '1' THEN                      --INPUT=1
                NextState <= S1;
            ELSE
                NextState <= S2;
            END IF;
            OP <= '1';                               --OUTPUT
        WHEN S2 =>                                --STATE S2
            IF DIN = '1' THEN                      --INPUT=1
                NextState <= S2;
            ELSE
                NextState <= S3;
            END IF;
            OP <= '0';                               --OUTPUT
        WHEN S3 =>                                --STATE S3
            IF DIN = '1' THEN                      --INPUT=1
                NextState <= S0;
            ELSE
                NextState <= S1;
            END IF;
            OP <= '0';                               --OUTPUT
        WHEN OTHERS =>                            --Initial State
            NextState <= S0;
            OP <= '0';                               --OUTPUT
        END CASE;
    END PROCESS ChangeStateMode;
END behave;

```

Moore 状态机的 RTL 图:

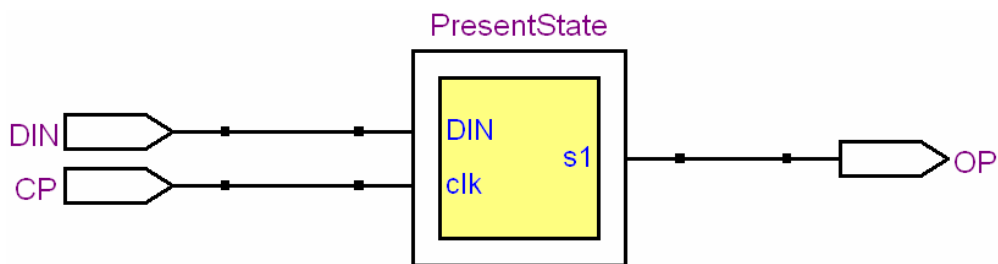


图 1.5.4 Moore 状态机的 RTL 顶层图

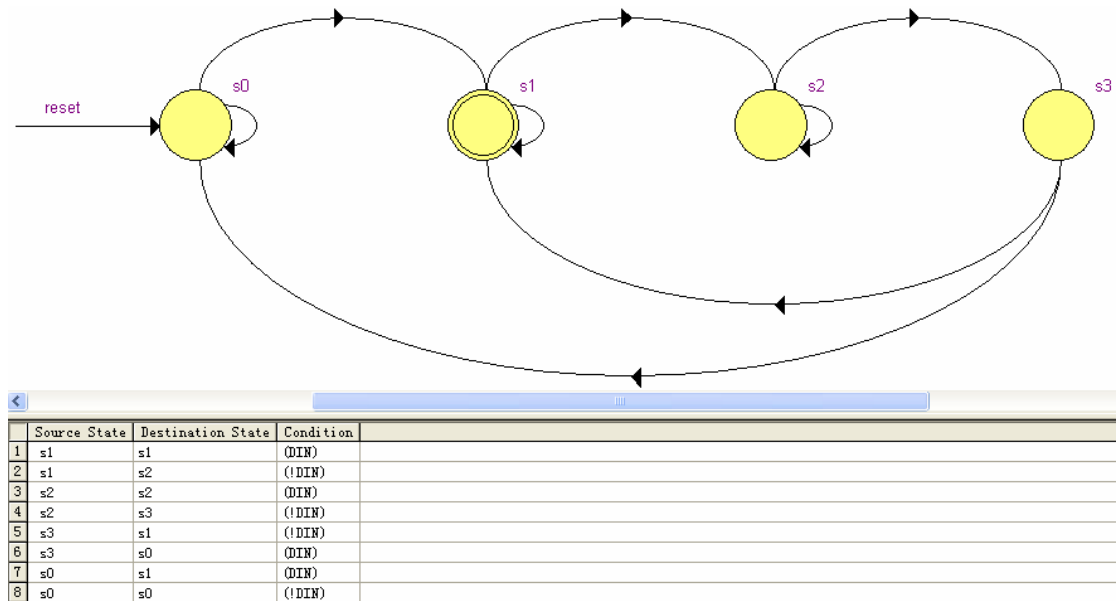


图 1.5.5 Moore 状态机的状态图

--Mealy 状态机 VHDL 程序

```

LIBRARY IEEE;                                --标准库资源
USE IEEE.STD_LOGIC_1164.ALL;                  --标准逻辑程序包
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Mealy is
  PORT ( CP      : IN    STD_LOGIC;           -- CLOCK
         DIN     : IN    STD_LOGIC;           -- I/P Signal
         OP      : OUT   STD_LOGIC            -- O/P Signal
        );
END Mealy;
ARCHITECTURE behave OF Mealy IS
  TYPE STATE IS (S0,S1,S2,S3);                --State Type Declare
  SIGNAL PresentState : STATE;                 -- Present State
  SIGNAL NextState    : STATE;                 -- Next State
BEGIN
  SwitchToNextState : Process (CP)              -- PresentState -> NextState
  BEGIN
    IF CP'EVENT AND CP = '1' THEN
      PresentState <= NextState;
    END IF;
  END PROCESS SwitchToNextState;
  ChangeStateMode : PROCESS (DIN,PresentState)
  BEGIN
    CASE PresentState IS
      WHEN S0 =>                                --STATE S0
        IF DIN = '0' THEN                       --INPUT=0
          NextState <= S0;
        
```

```

        OP <= '0';           --OUTPUT
    ELSE
        NextState <= S1;
        OP <= '1';           --OUTPUT
    END IF;
    WHEN S1 =>                --STATE S1
        IF DIN = '1' THEN    --INPUT=1
            NextState <= S1;
            OP <= '1';        --OUTPUT
        ELSE
            NextState <= S2;
            OP <= '0';        --OUTPUT
        END IF;
    WHEN S2 =>                --STATE S2
        IF DIN = '1' THEN    --INPUT=1
            NextState <= S2;
            OP <= '0';        --OUTPUT
        ELSE
            NextState <= S3;
            OP <= '1';        --OUTPUT
        END IF;
    WHEN S3 =>                --STATE S3
        IF DIN = '1' THEN    --INPUT=1
            NextState <= S0;
            OP <= '1';        --OUTPUT
        ELSE
            NextState <= S1;
            OP <= '0';        --OUTPUT
        END IF;
    WHEN OTHERS =>           --Initial State
        NextState <= S0;
        OP <= '0';           --OUTPUT
    END CASE;
END PROCESS ChangeStateMode;
END behave;

```

Mealy 状态机的 RTL 图:

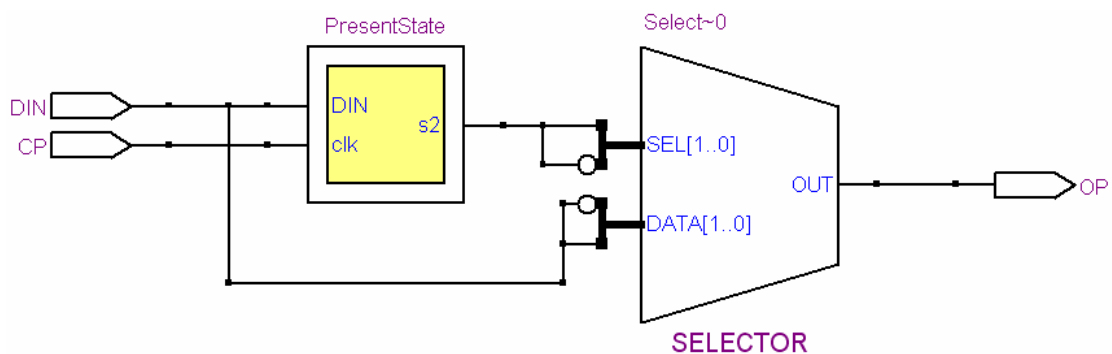


图 1.5.6 Mealy 状态机的 RTL 顶层图

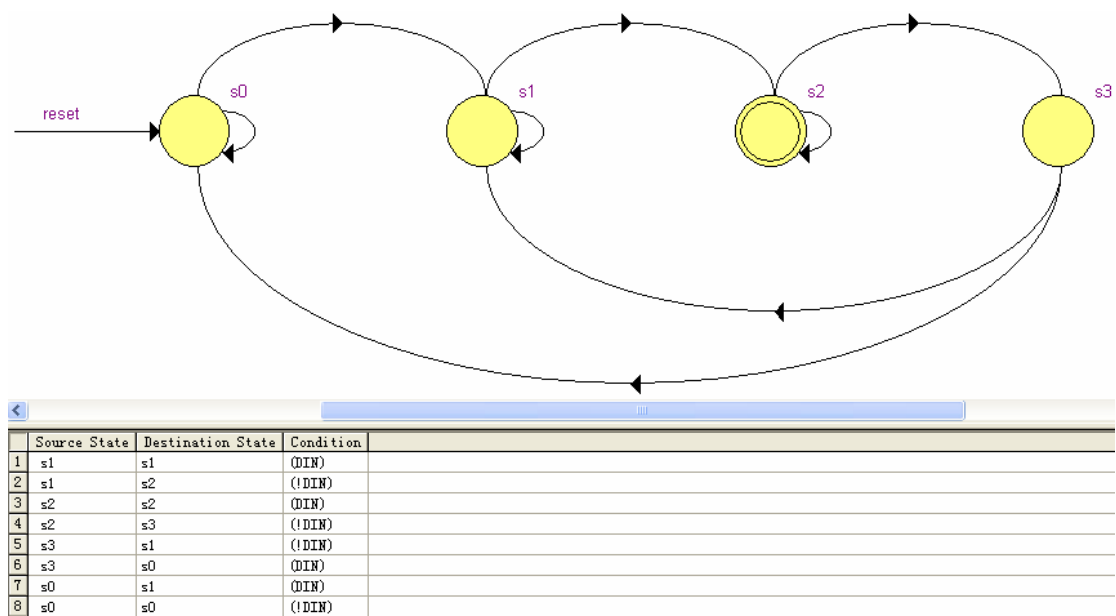


图 1.5.7 Mealy 状态机的状态图

第二章 Quartus II 的使用

2.1 Quartus II 概述

Quartus II 是 Altera 公司继 MAX+PLUS II 后，所提供的 FPGA/CPLD 开发集成环境，主要针对本公司新器件和大规模 FPGA 的开发。Quartus II 提供一个容易适应特定设计所需要的完整的多平台设计环境。它不仅包括 FPGA/CPLD 设计所有阶段的解决方案，而且也提供可编程片上系统（SOPC）设计的综合性环境。Quartus II 除了保留有 MAX+PLUS II 的特色外，也可以利用第三方的综合工具，如 Synopsys、NativeLink、仿真工具 ModelSim 等。

Quartus II 可以使设计者完成设计输入、分析与综合、仿真、布局布线、时序分析及编程下载等工作。下图显示了使用 Quartus II 进行设计的各主要环节。

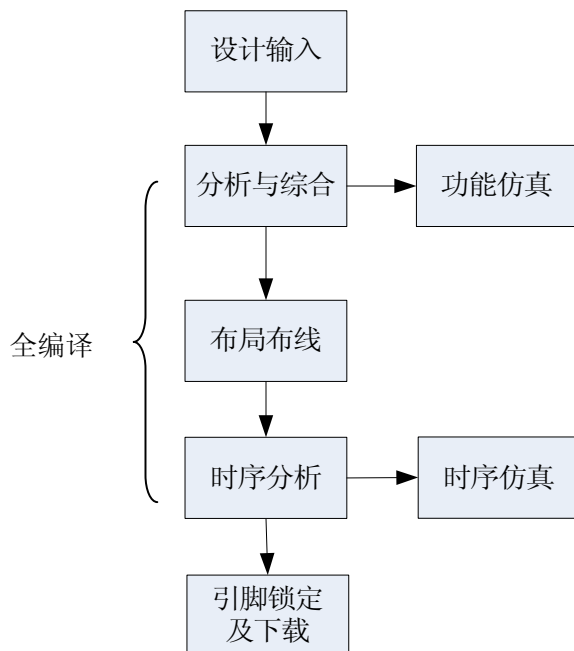


图 2.1.1

这几个环节分别介绍如下：

1. 设计输入

设计输入包括图形输入和硬件描述语言（HDL）文本输入两大类型。本次实验中主要用到其中的原理图输入和 VHDL 输入两种方式。HDL 设计方式是现今设计大规模数字集成电路的常用形式，除 IEEE 标准中 VHDL 与 Verilog HDL 两种形式外，还有各自 FPGA 厂家推出的专用语言，如 Quartus II 下的 AHDL。HDL 语言描述在状态机、控制逻辑、总线功能方面较强；而原理图输入在顶层设计、数据通路逻辑等方面具有图形化强、功能明确等特点。

Quartus II 支持层次化设计，可以在一个新的输入编辑环境中调用不同输入设计方式完成的模块，从而完成混合输入设计以发挥二者各自特色。

2. 分析与综合

在完成设计输入之后，即可对其进行分析与综合。其中先进行语法的分析与校正，然后依据逻辑设计的描述和各种约束条件进行编译、优化、转换和综合。最终获得门级电路甚至更底层的电路描述网表文件。因此，综合就是将电路的高级语言（如行为描述）转换成低级的，可与 FPGA/CPLD 的基本结构相映射的网表文件或程序。既可以使用 Quartus II 中的综合器来分析设计文件和建立工程数据库，也可使用其他 EDA 综合工具综合设计文件，然后产生与 Quartus II 软件配合使用的网表文件。

3. 功能仿真

进行功能仿真，即直接对 VHDL、原理图描述或其他描述形式的逻辑功能进行测试模拟，以了解其实现的功能否满足原设计的要求，仿真过程没有加入时序信息，不涉及具体器件的硬件特性。Quartus II 可以通过建立和编辑波形文件，来执行功能的波形模拟分析。

4. 布局布线

若功能仿真结果满足逻辑设计，则可执行布局布线。它的目的是将综合后产生的网表文件配置于指定的目标器件中，使之产生最终的下载文件。在 Quartus II 中，是使用由综合中建立的数据库，将工程的逻辑和时序要求与器件的可用资源相匹配。它将每个逻辑功能分配给最好的逻辑单元位置，进行布线和时序，并选择相应的互连路径和引脚分配。

5. 时序分析

时序分析就是接近真实器件运行特性的仿真，仿真文件中已包含了器件硬件特性参数，因而，仿真精度高。Quartus II 中的时序分析功能可以分析设计中所有逻辑的性能，并协助引导适配器满足设计中的时序分析要求。还可以进行最少的时序分析，报告最佳情况时序结果，验证驱动芯片外信号的时钟至管脚延时。设计者可以根据所产生的仿真波形来分析、调试和验证设计的时序性能。

6. 引脚锁定及下载

为了对设计工程进行硬件测试，应将其输入输出信号锁定在芯片确定的引脚上。最后是将下载或配置文件通过编程电缆向 FPGA 或 CPLD 进行下载，以便进行硬件调试和验证。

2.2 Quartus II 的 VHDL 输入设计流程

本节将以一个 4 分频的分频器为例，介绍运用 Quartus II 实现其功能的详细步骤及方法。其主要设计流程如下：

新建工程→新建 VHDL 设计文件→功能仿真→全编译→时序仿真→引脚锁定和下载。

2.2.1 新建工程


首先建立自己的工作文件夹，用来存放所有的设计工程及文件，建议用 DA+学号。在工作文件夹中还可以再建立设计工程的文件夹，不同的设计项目放在不同的文件夹中。在机房里所有用户文件夹都建在 D 盘中，文件夹名称中不要含有中文。如

D:\DA2004010111\divider4。具体操作步骤如下：

1. 双击桌面上的 Quartus II 图标，打开主界面。选取菜单中 File—New Project Wizard，出现新建工程向导窗口。直接点击 Next 进入设置窗口如图 2.2.1 所示。



图 2.2.1

2. 在工程目录设定处按 ，在 Select Directory 对话框中选择此工程的存放路径，如图中所示为 D:\DA2004010111\divider4。选中后，点击打开按钮。与此同时，Quartus II 自动将工程名称、顶层设计实体名称与存放工程的文件夹名称一样，同为 divider4。

3. 点击 Next 进入添加文件窗口（图略）。如果文件夹中存有已录入的与工程相关的输入文件，那么可以直接添加到工程中来。因为目前没有任何输入文件，所以点击 Next 进入下一窗口，即选择目标芯片窗口如图 2.2.2 所示。

4. 在 Family 栏中选择目标芯片系列——Cyclone，然后选择此系列中的具体芯片型号为 EP1C6Q240C8。

5. 点击 Next 进入 EDA 工具设置窗口（图略），勾选要用的第三方 EDA 工具。本次操作不采用第三方工具，因此点击 Next 进入最后的总结窗口（图略）。在这个窗口中列出了所有前面设置的结果。如果有错误可以点击 Back 回去一一修改，否则按 Finish 结束。

经过第一次的设置后，以后再新建工程时目标芯片等设置可以略掉，只需在图 2.2.1 中设定好工程的存放路径后就直接点击 Finish 结束。

这时在 Quartus II 主界面左侧的工程导航栏 Project Navigator 中显示本工程的顶层设计文件名称为 divider4，如图 2.2.3 所示。若没有出现导航栏，可以从菜单栏 View—Utility—Project Navigator 中调出。

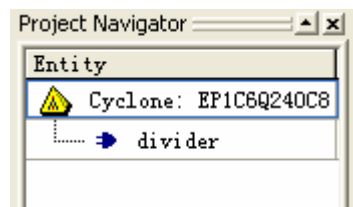


图 2.2.3

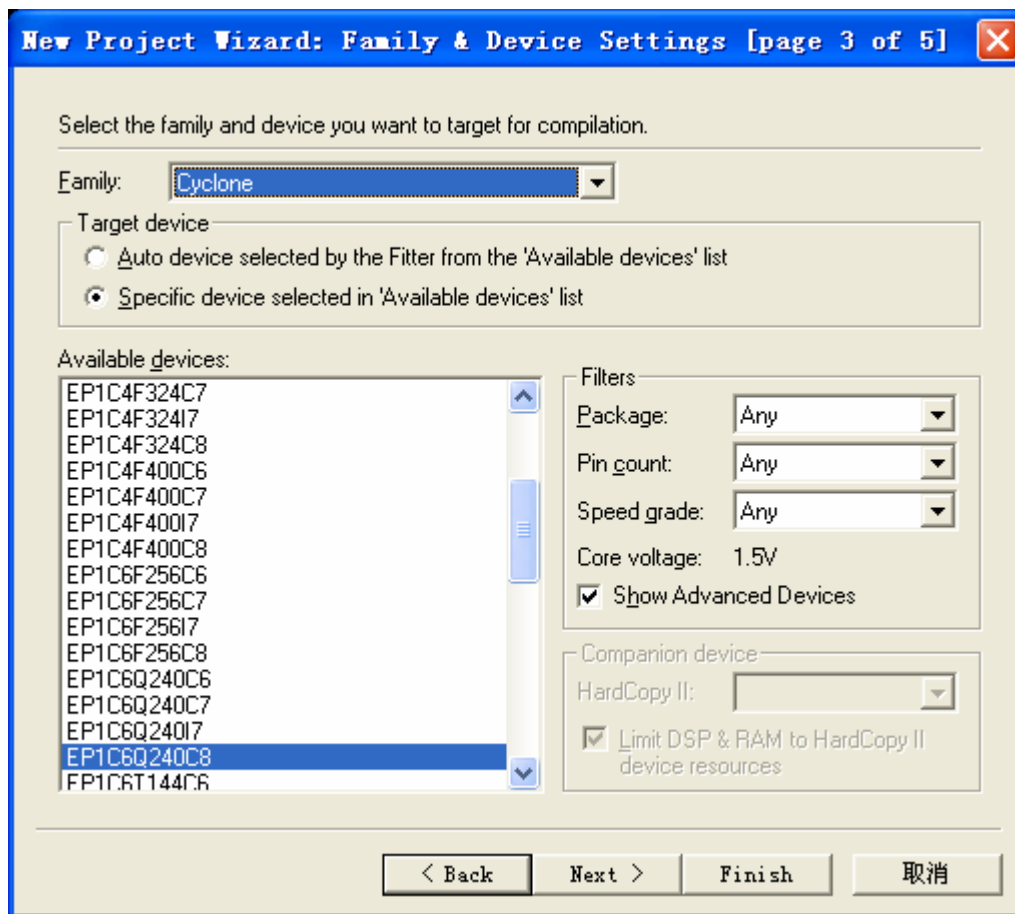


图 2.2.2

2.2.2 新建 VHDL 设计文件

在建好工程后，就可以添加 VHDL 输入文件。有以下几个步骤：

1. 在 Quartus II 主界面菜单栏中选择 File—New，弹出新建设计文件窗口如图 2.2.4 所示。在 Device Design Files 页选中 VHDL File 项，点击 OK 按钮打开 VHDL 文本编辑窗口，其默认文件名为

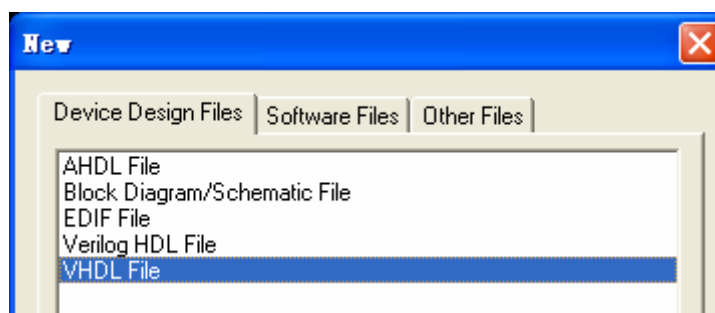


图 2.2.4

“Vhdl.vhd”。

2. 输入 VHDL 文件有两种方式：一个是直接在空白处输入设计文件，4 分频的 VHDL 文件如下所示

```
1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.all;
3
4 entity divider4 is
5     generic(LEN : integer := 2);
6     port(
7         clkkin : in    std_logic;
8         clkout  : out   std_logic
9     );
10 end divider4;
11
12 architecture beh of divider4 is
13 begin
14     process(clkkin)
15         variable cnt : integer range 0 to LEN - 1;
16         variable clkt : std_logic ;
17     begin
18         if rising_edge(clkkin) then
19             if cnt = LEN - 1 then
20                 if clkt = '1' then
21                     clkt := '0' ;
22                 else
23                     clkt := '1' ;
24                 end if;
25                 cnt := 0 ;
26             else
27                 cnt := cnt + 1;
28             end if;
29             clkout <= clkt;
30         end if;
31     end process;
32 end beh;
```

最后保存文件名为“divider4.vhd”。注意： 确认文件保存在本工程文件夹下、实体名和 VHDL 设计文件名一致，而且在保存时要勾选“保存为”对话框中下方的“Add file to current project”选项。

另一个方式是通过使用模板来输入文件。方法是选择菜单 Edit—Insert Template 或在空白窗口中单击右键选择 Insert Template，弹出插入模板窗口如图 2.2.5 所示。在左侧 Show syntax of 列表中选中 VHDL，然后从右侧 Template section 列表选取 Architecture Body 并点击 OK 即可。那么就会将结构体模板插入到文本编辑窗口中，随后修改模版内容成为所需要的 VHDL 输入文件。在 Template section 列表中还有其他多种模版形式，可以提供给设计者使用。

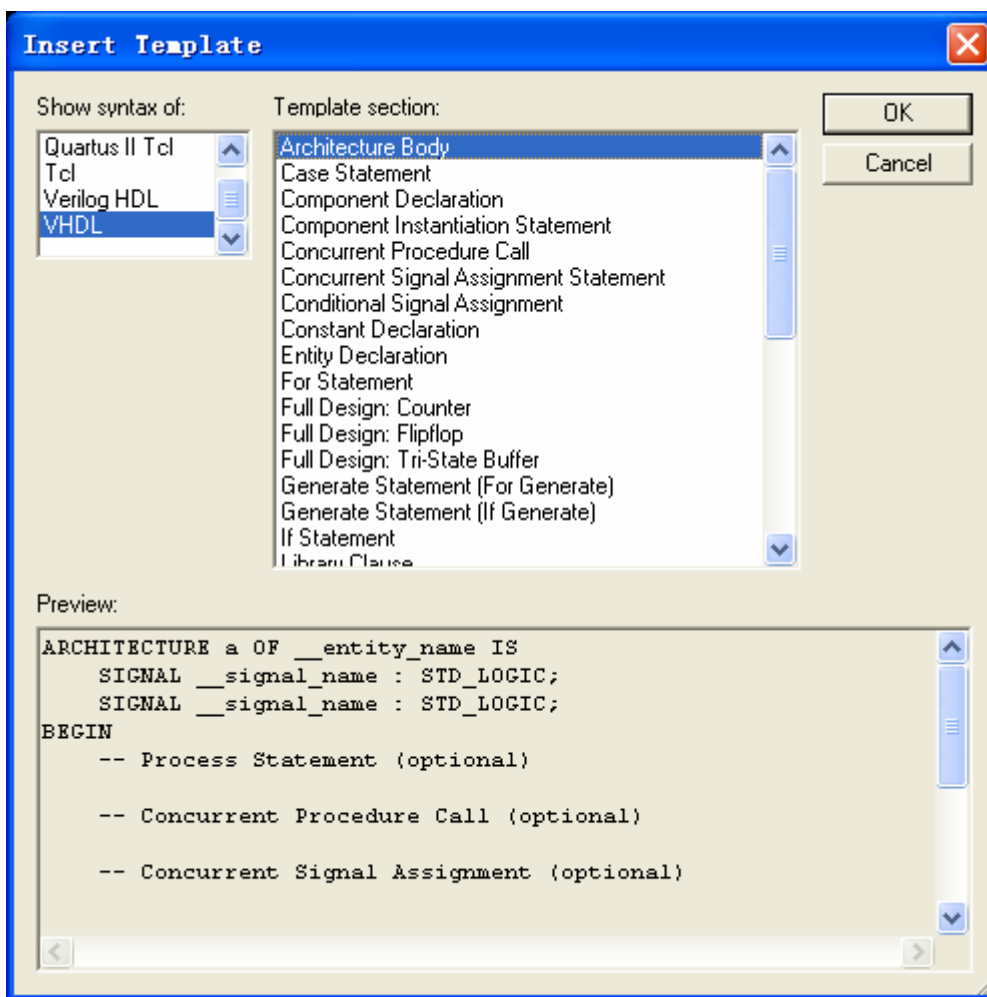



图 2.2.5


2.2.3 功能仿真

因为功能仿真只是要对设计文件进行逻辑功能的测试，不经过适配也不涉及具体器件的硬件特性。所以直接进行功能仿真的好处是编译耗时短，提高开发效率。在这节中将介绍如何对 VHDL 设计文件进行分析与综合，然后通过观察输入输出波形的关系来检查它是否满足设计要求。

一、对 VHDL 设计文件执行分析与综合。从菜单栏中选择 Processing — Start — Start Analysis&Synthesis 或单击快捷按钮.

编译进行时，将检查文件的逻辑完整性及语法错误等，并在左侧 Status 栏中显示编译的进度，同时主界面下方的信息窗口中实时显示进程中的各条信息。如果出现错误信息，可双

击此条文，则立即在 VHDL 设计文件中标记至相应位置。一般在多条错误信息中只要修改最上面显示的错误即可，因为一种错误会导致多个错误信息的出现。修改后保存文件重新执行编译，直至排除所有的错误。

二、指定功能仿真模式。选择菜单中 Assignments—settings 或快捷按钮 ，在左侧 Category 栏中选中 Simulator，然后在右侧 Simulation mode 的下拉栏中选中 Functional 如图 2.2.6 所示。

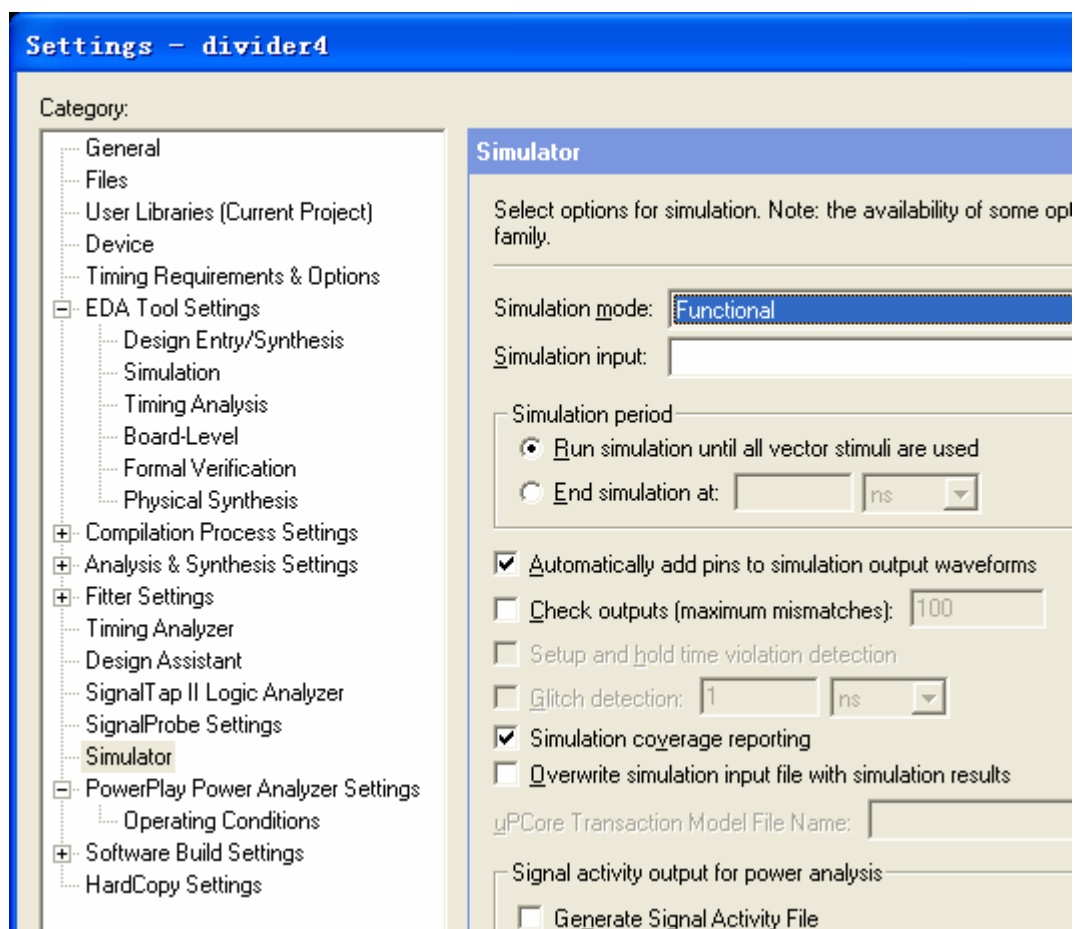


图 2.2.6

三、通过建立波形文件进行仿真。具体步骤如下：

1. 在 Quartus II 主界面菜单栏中选择 File—New，在 Other Files 页选中 Vector Waveform File 项，如图 2.2.7 所示。点击 OK 按钮打开空白波形编辑窗口，其默认文件名为 “Waveform1.vwf”。

2. 选择菜单栏中 Edit—Insert Node or Bus，弹出插入节点窗口如图 2.2.8 所示。

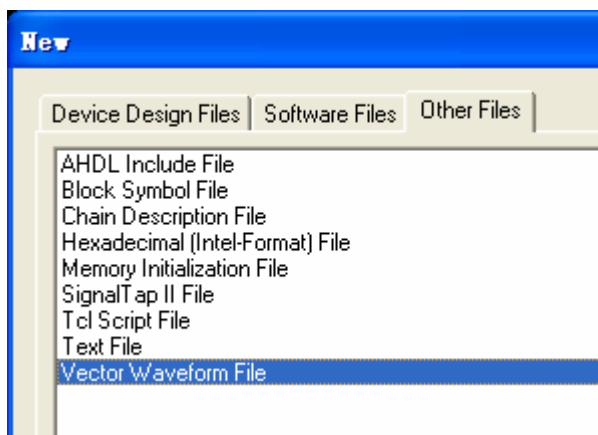


图 2.2.7

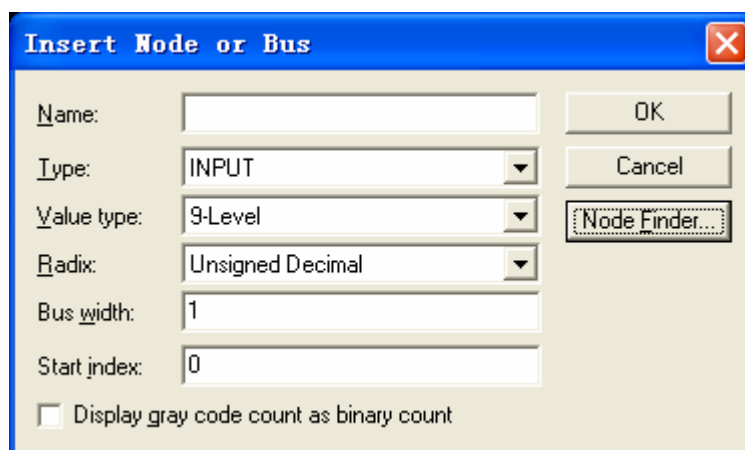



图 2.2.8

3. 点击上图中的 Node Finder 按钮，再点击弹出窗口中的 List 按钮。在左侧 Nodes Found 窗口中选取 clkin 及 clkout，然后点击  按钮将选中信号选取至右侧 Selected Nodes 窗口中，如图 2.2.9 所示。

最后点击 OK 回到插入节点窗口，再次点击 OK 回到波形编辑窗口。

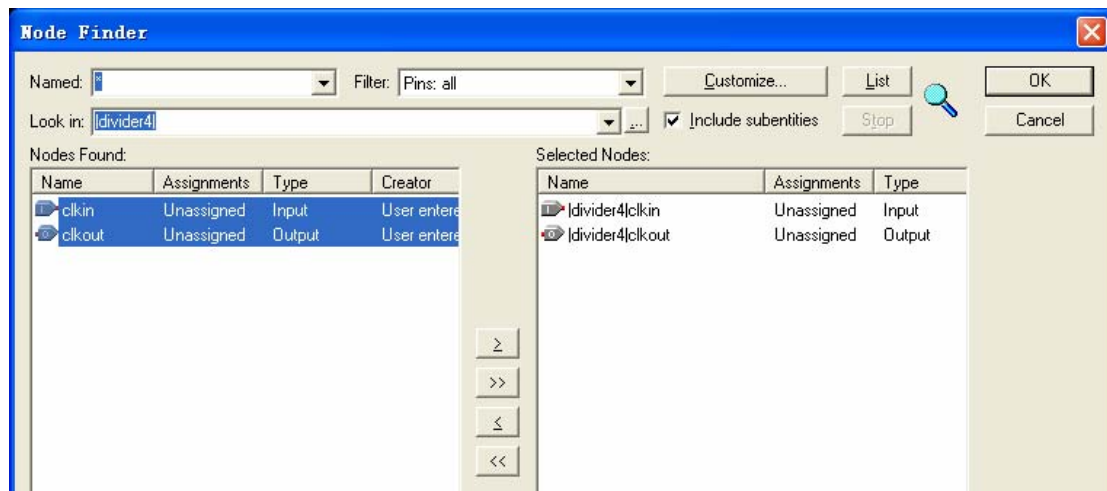



图 2.2.9

4. 选中输入信号 **clkin** 使之成为蓝条显示, 选取波形编辑窗口左侧栏中的  按钮, 接受默认设置, 结果如图 2.2.10 所示。保存此波形文件名为 “divider4.vwf”。

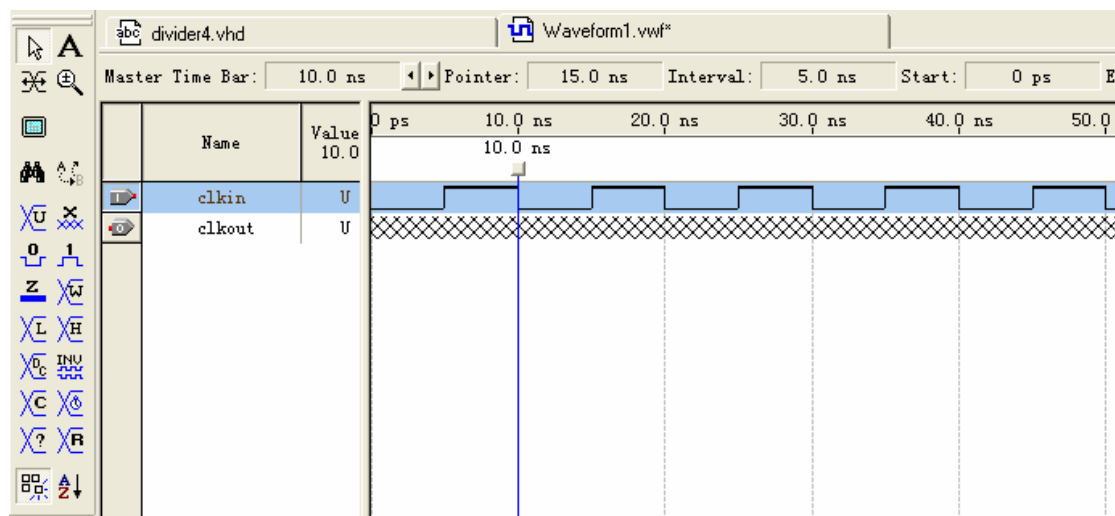




图 2.2.10

5. 运行菜单 **Processing—Generate Functional Simulation Netlist** 命令产生用于功能仿真的网表文件。

6. 选取 **Processing—Start Simulation** 或快捷按钮  执行模拟仿真。仿真无误后, 通过点击  按钮或右键菜单中的 **Zoom** 命令将波形放至合适大小, 仿真结果波形图如图 2.2.11

所示。从图中可以看到输出 `clkout` 的周期是输入 `clkin` 的 4 倍，符合设计要求。

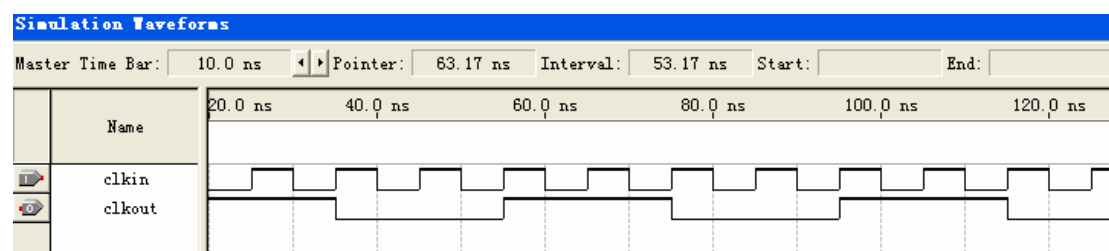


图 2.2.11


2.2.4 编译前的一些设置及全编译

Quartus II 编译器是由一系列处理模块构成的，它们负责对设计项目进行差错、逻辑综合、结构综合、输出结果的编辑配置及时序分析。在编译前，设计者可以通过不同的设置，使编译器利用不同的综合和适配技术，以提高设计项目的工作速度，优化器件的资源利用率。设计者在执行编译时，既可以 **Start Compilation** 全编译；也可以选择 **Start** 菜单中的不同选项，来分别进行分析与综合、布局布线（适配）、时序分析等等。

我们在前一小节做功能仿真时实际上就已经执行了全编译中分析与综合，当然也可以执行完全编译后再做功能仿真。

在这节中将介绍编译前的准备工作，进行全编译的步骤及编译成功后获得的结果。

一、在编译处理前，必须做好一些必要的设置。一般常做的有以下几个步骤：

1. 如果前面新建工程时已经选定了目标芯片，那么这步可以跳过不做。否则可以选择菜单中 **Assignments—settings** 或快捷按钮 ，在左侧 **Category** 栏中选中 **Device**，然后在右侧界面中按图 2.2.2 中所示选取目标芯片 EP1C6Q240C8。

2. 选择菜单中 **Assignments—settings** 或快捷按钮 ，在左侧 **Category** 栏中选中 **Device**，然后在右侧界面中单击 **Device&Pin Options** 按钮，弹出窗口如图 2.2.12 所示。选中 **General** 页，在 **Options** 栏中 **Auto-restart Configuration after error**，使对 FPGA 的配置失败后能自动重新配置。

3. 如果需要将配置文件 (*.pof) 下载到配置器件中，可在编译前做好设置。在此次实验中，就是通过下载到配置器件中的 pof 文件来实现对小车的控制的。选中图 2.2.12 中的 **Configuration** 页，在如图 2.2.13 所示窗口里选中 **Generate Compressed bitstreams** 选项。

4. 由于我们所用的实验系统上配置器件是 EPCS1，而对 EPCS1 的编程必须用 **AS Mode**。因此在 **Configuration** 页还要选择 **Configuration scheme** 为 **Active Serial**，**Configuration device** 为 **EPCS1**。

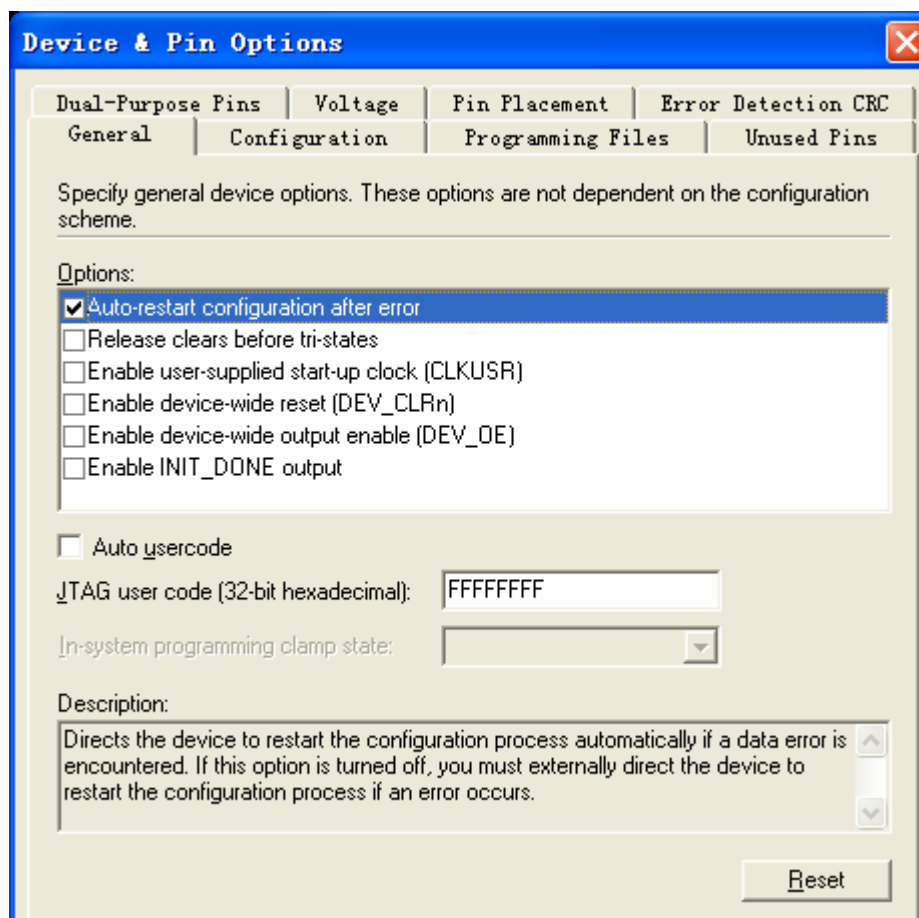


图 2.2.12

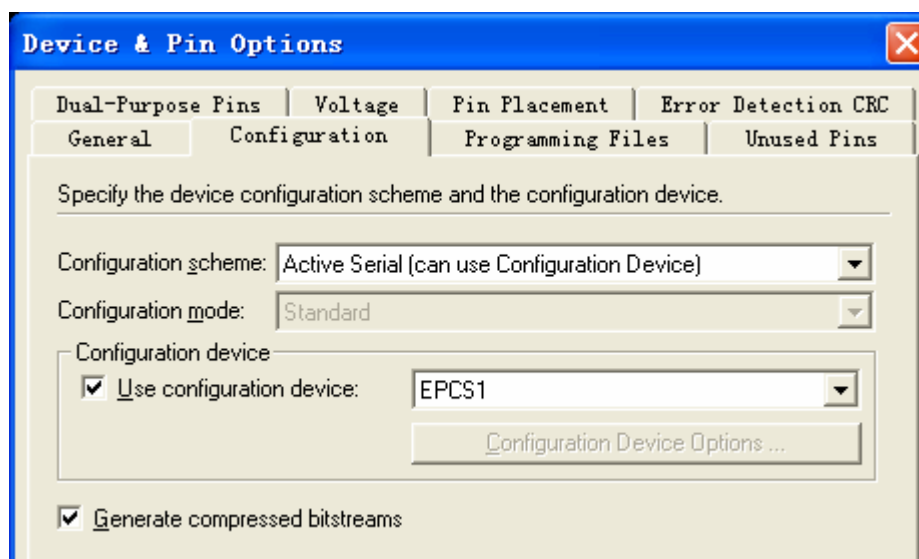



图 2.2.13

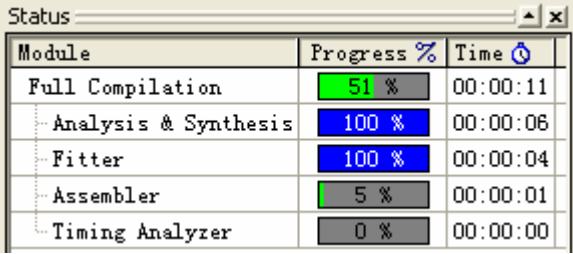
二、执行全编译。

选取菜单中 Processing — Start

Compilation 或快捷按钮  执行全编译。

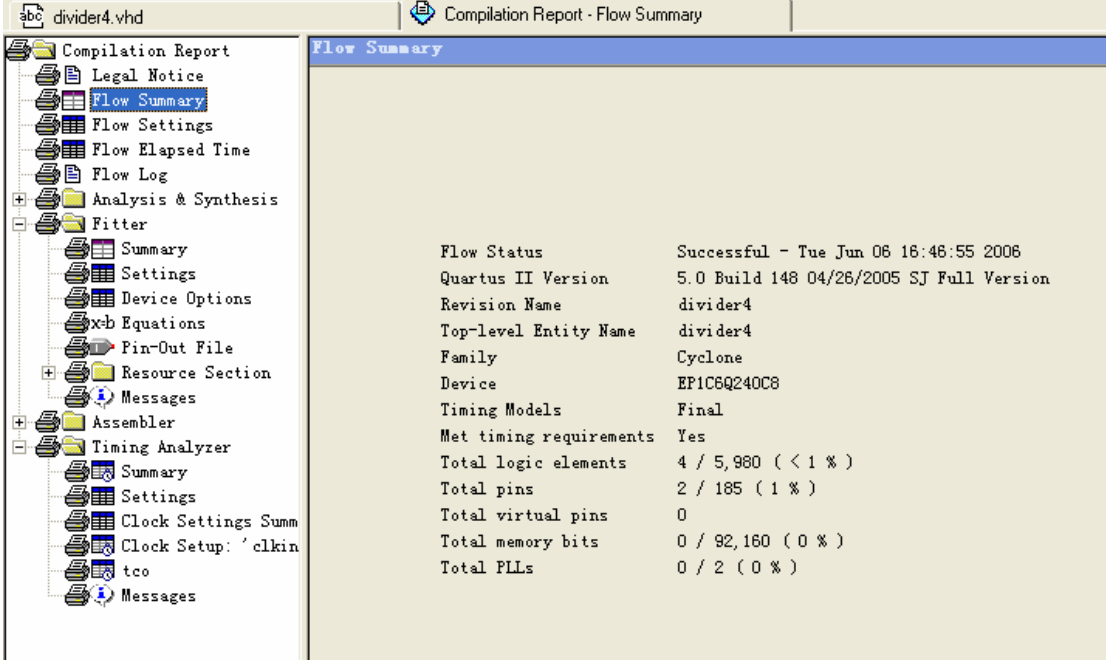
编译过程中 Status 栏里的进度状况如图 2.2.14 所示，其中显示了每一个编译的执行步骤及所耗费时间等信息，主界面最下方的信息窗口中同时显示各条信息。

编译成功后，可以从 Compilation Report 页中读到硬件耗用统计报告、布局布线报告及时序特性报告等信息，如图 2.2.15 所示。



Module	Progress %	Time
Full Compilation	51 %	00:00:11
Analysis & Synthesis	100 %	00:00:06
Fitter	100 %	00:00:04
Assembler	5 %	00:00:01
Timing Analyzer	0 %	00:00:00

图 2.2.14




Flow Summary	
Flow Status	Successful - Tue Jun 06 16:46:55 2006
Quartus II Version	5.0 Build 148 04/26/2005 SJ Full Version
Revision Name	divider4
Top-level Entity Name	divider4
Family	Cyclone
Device	EP1C6Q240C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	4 / 5,980 (< 1 %)
Total pins	2 / 185 (1 %)
Total virtual pins	0
Total memory bits	0 / 92,160 (0 %)
Total PLLs	0 / 2 (0 %)

图 2.2.15

全编译完成后将产生下载所需的 sof 或 pof 文件。

2.2.5 时序仿真

在全编译期间已经对设计文件自动进行了时序分析，并从编译报告中读取了相关时序结果。在这里我们希望通过波形模拟方式分析输入输出信号。以下是具体操作过程，其中一些步骤与做功能仿真时相同。

一、指定时序仿真模式。选择菜单中 Assignments—settings 或快捷按钮 ，在左侧 Category 栏中选中 Simulator，然后在右侧 Simulation mode 的下拉栏中选中 Timing，如图 2.2.16 所示。

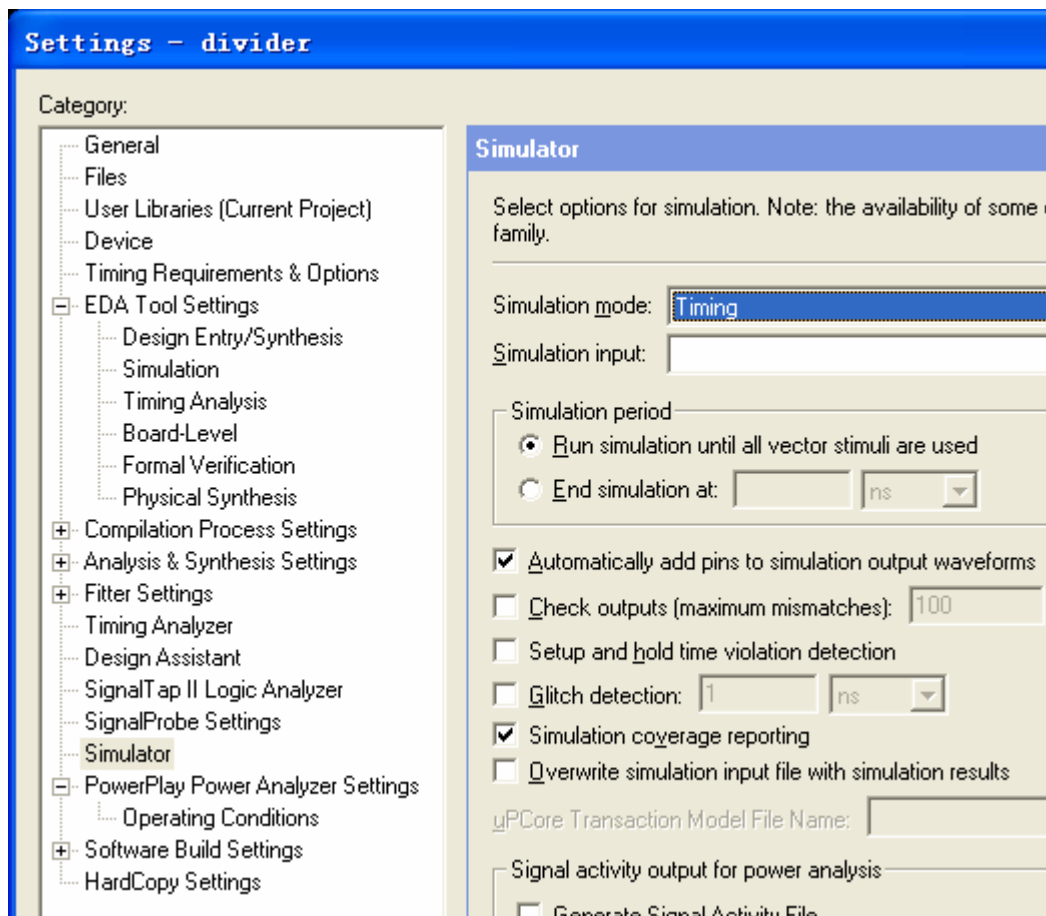


图 2.2.16

二、通过建立波形文件进行仿真，可以按以下步骤进行：

1. 在 Quartus II 主界面菜单栏中选择 File—New, 在 Other Files 页选中 Vector Waveform File 项。点击 OK 按钮打开空白波形编辑窗口，其默认文件名为 “Waveform1.vwf”。或者打开已有的波形文件*.vwf。

2. 对于时序仿真来说，将仿真时间设置在一个合理区域十分重要。通常设置的时间范围在数十微秒之间。选择菜单中 Edit—End Time, 在弹出的结束时间窗口中设置为 1us, 如图 2.2.17 所示。

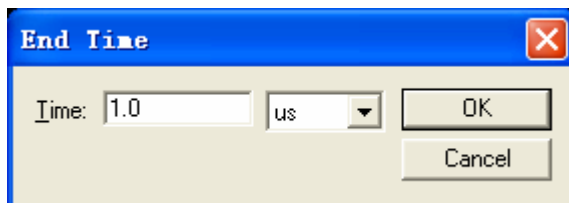




图 2.2.17

3. 加入输入输出信号，设置输入信号周期，保存文件等步骤均都与前面介绍功能仿真时相同。

4. 选取 Processing—Start Simulation 或快捷按钮  执行模拟仿真。运用  按钮或右键菜单中的 Zoom 命令将波形放至合适大小，仿真结果波形图如图 2.2.18 所示。

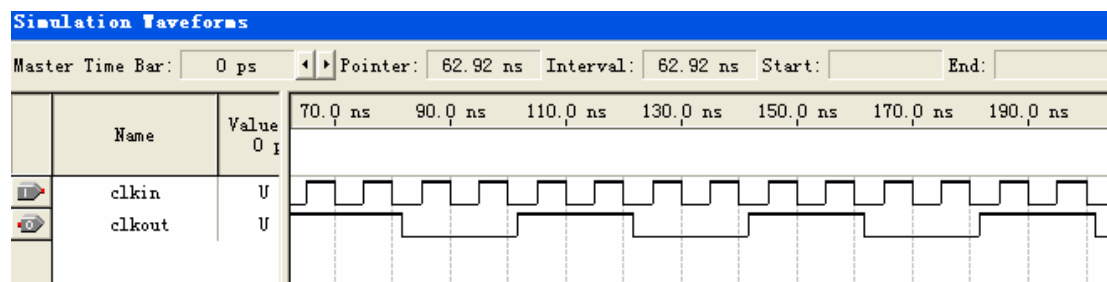


图 2.2.18

从图中可以看到，clkout 波形与前面功能仿真后的波形相比，出现了时间差异，请大家思考一下是什么原因造成的？

2.2.6 引脚锁定和下载

在时序仿真也完成之后，为了能完成对分频器的硬件测试，应将其输入输出信号锁定在具体的芯片引脚上，并编译下载。

一、引脚锁定。这一步就是将设计文件中的输入输出管脚与 FPGA 芯片的实际管脚对应起来。

1. 选择菜单栏中 Assignments—Pins, 弹出引脚分配窗口如图 2.2.19 所示。双击 To 栏下的<<new>>, 在出现的下拉列表中分别选定要锁定的输入输出信号名，然后双击 Location 栏下的<<new>>, 在出现的下拉列表中分别指定与输入输出信号所对应的引脚号。

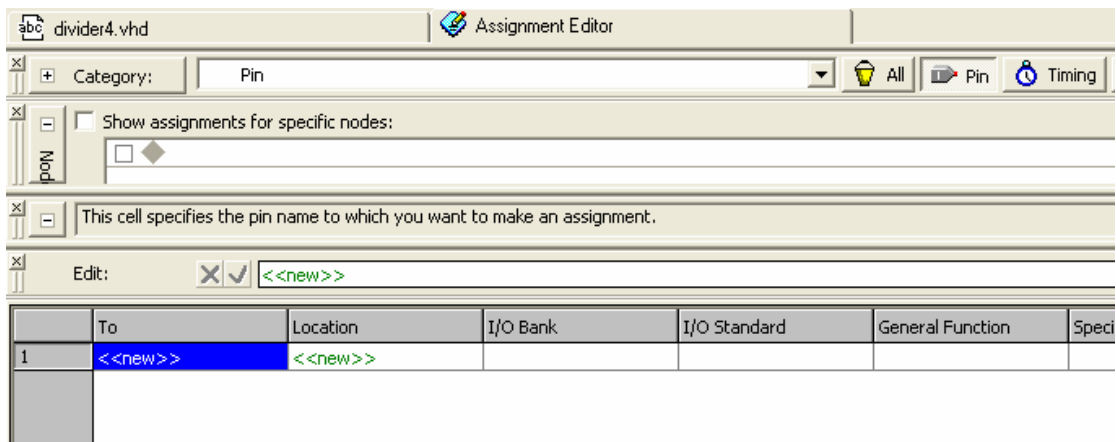


图 2.2.19

2. 例如：clk_{in} 锁定 28 脚，clk_{out} 锁定 214 脚，结果如图 2.2.20 所示（芯片引脚对照表参阅附录）。

	To	Location	I/O Bank	I/O Standard	General Function	Special Function
1	clk _{in}	PIN_28	1	LVTTTL	Dedicated Clock	CLK0/LVDSCLK1p
2	clk _{out}	PIN_214	2	LVTTTL	Column I/O	LVDS24n
3	<<new>>	<<new>>				

图 2.2.20

3. 保存引脚锁定信息，再做一次全编译（Start Compilation）以便将引脚锁定信息编译进下载文件中。

二、将编译产生的 sof 文件下载到 FPGA 中。

1. 将使用的 GW48 系列 SOPC/EDA 实验开发系统（具体介绍参阅附录）和并口通信线连接好。

2. 打开系统电源，设实验系统的工作模式为模式 5。

3. 选择菜单栏中 Tools—Programmer 或快捷按钮 ，弹出窗口如图 2.2.21 所示。

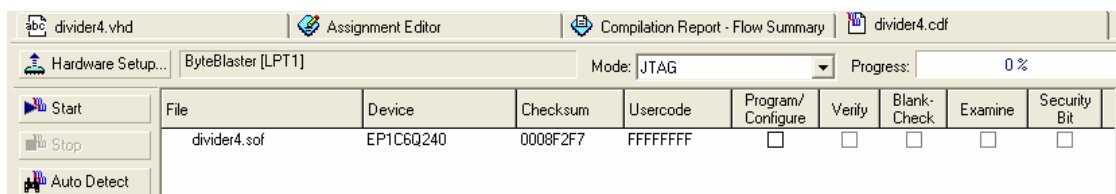


图 2.2.21

4. 初次使用 Quartus II 要将 Hardware Setup 选择为 ByteBlaster II[LPT1]。实验室里软件的这项设置已经做好，不必一一设置了。

5. 在 Mode 的下拉列表中选择 JTAG。
6. 添加下载文件 divider4.sof。
7. 勾选中 Program/Configure 项。
8. 单击 Start 按钮执行下载操作。下载成功后主界面下方信息提示窗中显示成功，Progress 显示 100%，如图 2.2.22 所示。

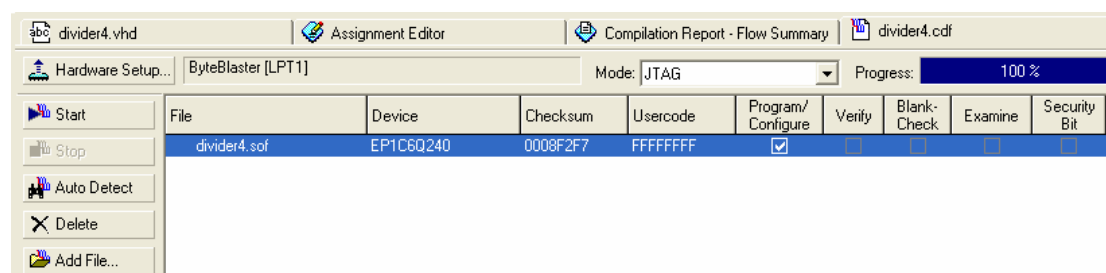


图 2.2.22

如果希望 FPGA 上电后能够保持原有的配置文件而不用重新下载，必须将配置文件*.pof 下载到配置芯片 EPCS1 中。EPCS1 是 Cyclone 系列芯片的专用配置芯片，编程模式为 Active Serial 模式。所以将图 2.2.21 中所示的设置改为：在 Mode 的下拉列表中选 Active Serial Programming，在弹出窗口中选择“是”。点击 Add files 按钮添加配置文件 divider4.pof。勾选中 Program/Configure、Verify、Blank-Check 项，如图 2.2.23 所示。单击 Start 按钮执行下载操作。

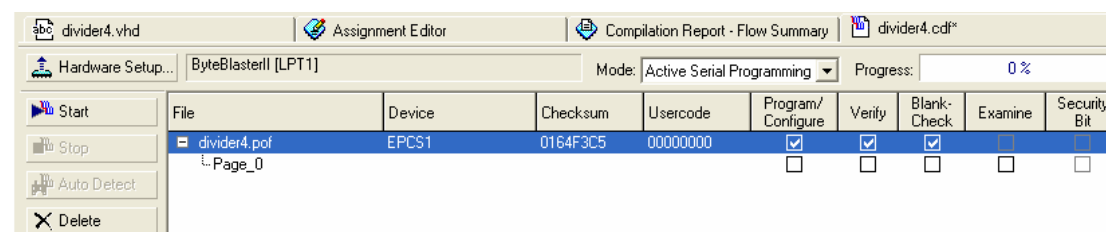


图 2.2.23

最后是在实验装置上验证 4 分频的分频器的功能。例如选择 clock0（输入端 clkin）的跳线位置为 750kHz，通过引出的芯片管脚用示波器观察输入输出信号的频率。

2.3 Quartus II 的原理图输入设计流程

在这一节中我们将在 2.2 节中 4 分频的基础上设计一个 16 分频的分频器，并且介绍用原理图输入方式进行层次化设计的流程，其中大部分步骤和上一节是相同的。

2.3.1 新建工程和生成元件符号

为了在原理图中设计输入 16 分频的分频器，我们采用把 2 个 4 分频的分频器相串联的方式。因此可以将 4 分频 VHDL 设计文件生成为一个元件符号，提供给顶层原理图调用，实现 VHDL 设计和原理图设计的混合输入设计方法。

1. 新建一个工程文件夹，如 D:\DA200401011\divider。由于上节中已经录入过 4 分频 VHDL 文件，所以可以直接拷贝来做为底层文件调用：将 divider4 工程文件夹中的 divider4.vhd 文件拷贝至当前文件夹中。

2. 新建工程名为 divider，同样选取目标芯片为 Cyclone 的 EP1C6Q240C8，然后点击 Finish 结束新建工程向导。

3. 选择 File—Open 打开 divider4.vhd 文件。再选择 File—Create/Update—Create Symbol Files for Current File 命令，将当前 VHDL 文件生成同名的元件符号存放在当前工程文件夹中。

如果当前文件是原理图文件，可以用同样的方式生成元件符号以备调用。

2.3.2 新建原理图设计文件

本小节介绍在建好的工程中加入原理图输入文件。由于要调用前面生成的 divider4 元件，因此它将是整个设计工程的顶层文件。包括以下几个步骤：

1. 在 Quartus II 主界面菜单栏中选择 File—New，弹出新建设计文件窗口如图 2.3.1 所示。在 Device Design Files 页选中 Block Diagram/Schematic File 项，点击 OK 按钮打开原理图编辑窗口，其默认文件名为“Block1.bdf”。

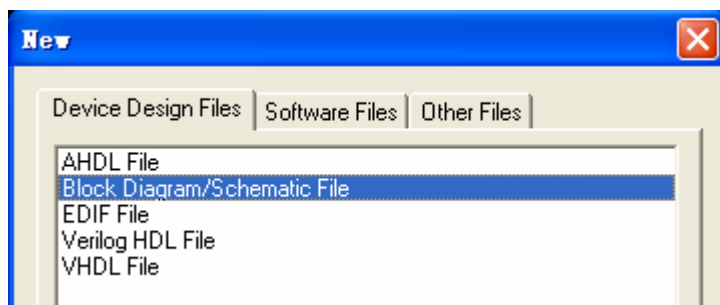


图 2.3.1

2. 在空白窗口中任意位置双击左键，弹出输入元件窗口。当选中左侧库列表中 Project 文件夹下的 divider4 元件时，则在右侧窗口中显示此元件的预览图，如图 2.3.2 所示。

3. 点击 OK 按钮之后，光标变为十字型且有元件模型随之移动。在空白图中适合位置单击左键放下元件。然后重复操作再放置一个。

4. 在输入元件窗口左侧库列表中，选中元件库文件夹中 primitives/pin 中的 input 元件，如图 2.3.3 所示。

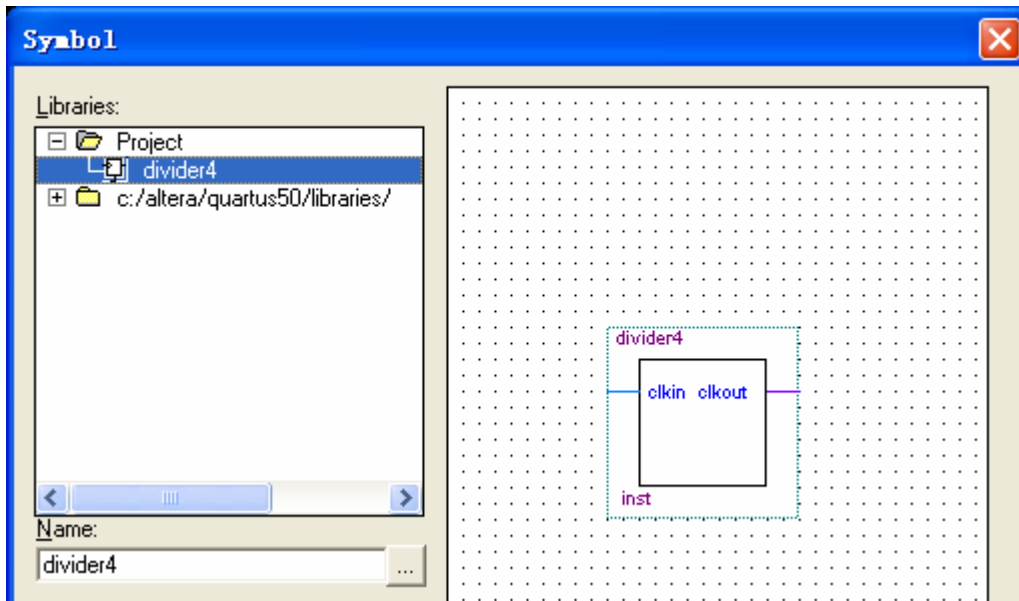


图 2.3.2

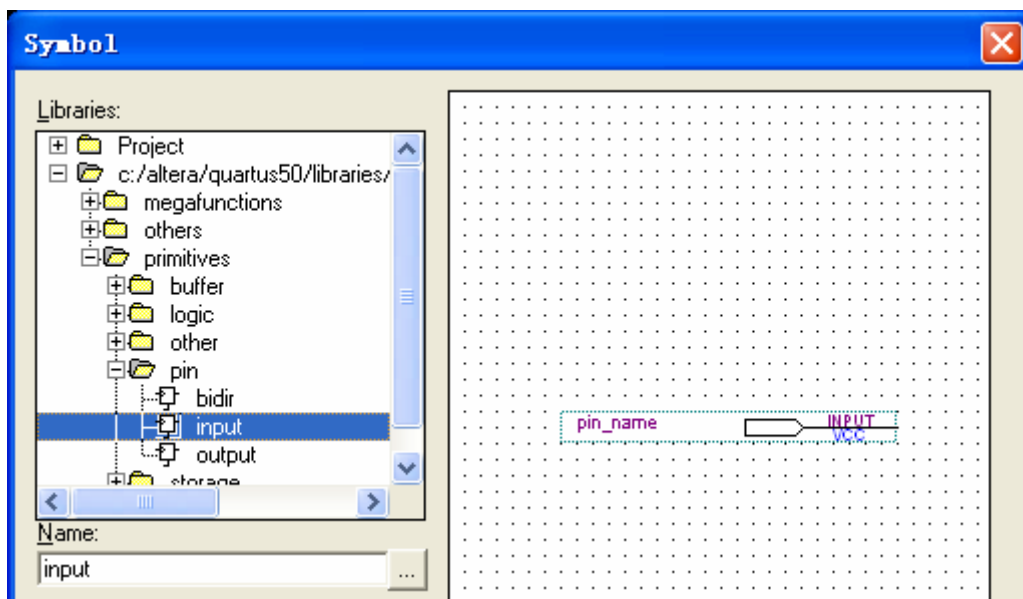


图 2.3.3

5. 点击 OK 按钮后放置在图纸最左侧。用同样的方法调出两个 output 元件，分别放在图中。
6. 将光标移至 input 元件的管脚边，使光标变为如图 2.3.4 所示的十字型。
7. 单击十字型光标，并移动至左边 divider4 元件的 clkin 管脚上，再次单击形成一根

连线。注意不要超过管脚端，不到或超过管脚端都会在线端显示×。完成其他连线，如图 2.3.6 所示。

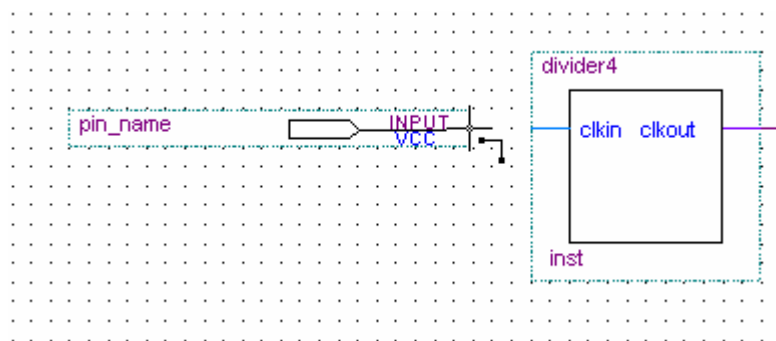


图 2.3.4

8. 双击 input 元件打开 Pin Properties 窗口，在 Pin names(s): 栏中输入输入端名称 clkinput，如图 2.3.5 所示，点击确定按钮退出。

9. 同样方法分别将两个 output 元件分别命名为 clkoutput1 和 clkoutput。完成顶层原理图的设计如图 2.3.6 所示。

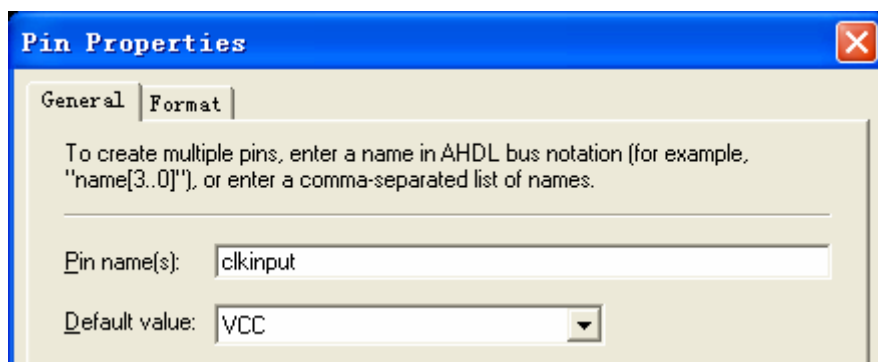


图 2.3.5

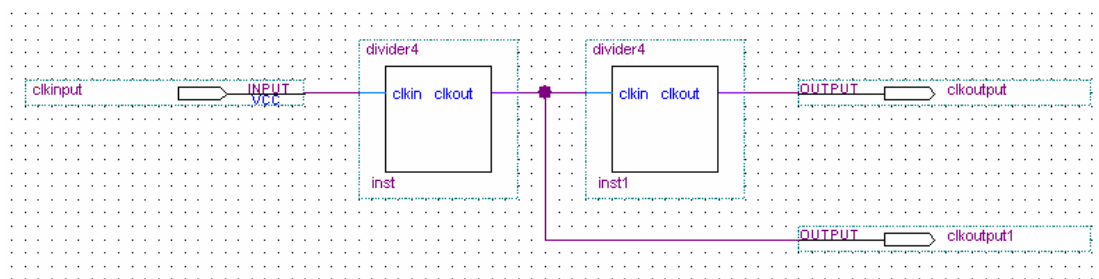


图 2.3.6

最后保存原理图的文件名为“divider.bdf”，注意保存在当前工程文件夹中，且在保存时

要勾选“保存为”对话框中下方的“Add file to current project”选项。

2.3.3 全编译和时序仿真

在这里我们略过功能仿真部分，直接观察时序仿真结果。这节中许多操作步骤与 2.2 节中介绍的完全相同，可参考相关小节。

在做好编译前的必要设置并执行完全编译后，在主界面左侧导航栏中可以看到工程的顶层和底层文件名称及层次结构，如图 2.3.7 所示。

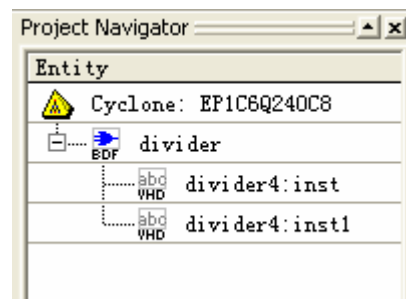



图 2.3.7

一、指定时序仿真模式。选择菜单中

Assignments—settings 或快捷按钮 ，在左侧

Category 栏中选中 Simulator，然后在右侧



Simulation mode 的下拉栏中选中 Timing，如图 2.2.16 所示。

二、通过建立波形文件进行仿真，步骤如下：

1. 选择菜单栏中 File—New，在 Other Files 页选中 Vector Waveform File 项。点击 OK 按钮打开空白波形编辑窗口。

2. 选择菜单中 Edit—End Time，在弹出的结束时间窗口中设置为 1us，如图 2.2.17 所示。

3. 加入输入输出信号，设置输入信号周期为 10ns，然后保存波形文件。可参见 2.2.5 小节。

4. 选取 Processing—Start Simulation 或快捷按钮  执行模拟仿真。运用  按钮或右键菜单中的 Zoom 命令将波形放至合适大小，仿真结果波形图如图 2.3.8 所示。

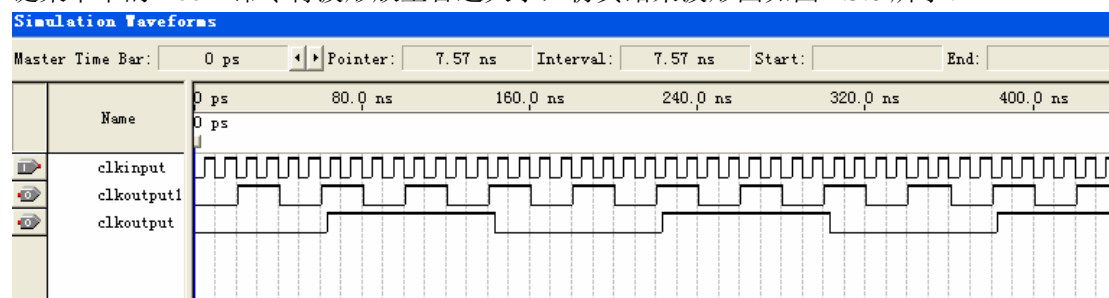


图 2.3.8

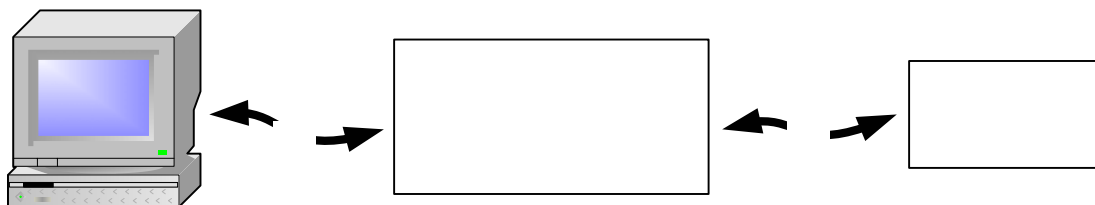
从图中可以看出输出端 clkoutput 满足 16 分频的要求。

时序仿真结果满足设计要求后，可对整个工程进行引脚分配和下载。这些也都介绍过了，就不再一一赘述了。

附录 实验装置介绍

1.1 简介

实验中使用的 SOPC/DSP/EDA 实验开发系统从外观看由 GW48-PK2 主板和 GWAC6L 适配板两部分组成。做实验时将适配板插在实验开发系统主板使用；并通过电缆与计算机通讯。附图 1-1 是 FPGA 实验系统的结构图。以下部分我们介绍实验开发系统主板和适配板的主要功能。



附图 1-1 FPGA 实验系统结构图

1.2 GWAC6L 适配板

GWAC6L 适配板上配有型号为 Cyclone EP1C6Q240C8 的 FPGA 芯片和两片 32 位 SRAM、一片 16 位 FLASH、50MHz 晶振、RS-232 串行接口、VGA 接口、PS2 接口；还可外加 A/D、D/A 器件板。图 1-2 是 GWAC6L 适配板的面板图。

GWAC6L 适配板主要模块介绍：

1. JTAG PORT：用于编程开发、测试和 SOPC 软件调试。注意使用时应该将所配的 10 芯线与 GW48-PK2 主板上的 ByteBlasterII 接口相连。
2. AS PORT：用于对 FPGA 的配置芯片 EPCS1/4 的编程。
3. VGA 输出口：可输出 256 色。与 Cyclone 的管脚连线：R 色（R2 接 PIO34、R1 接 PIO33、R0 接 PIO32）；G 色（G2 接 PIO31、G1 接 PIO30、G0 接 PIO29）；B 色（B1 接 PIO28、B0 接 PIO27）；VS 接 PIO35、HS 接 PIO36。
4. RS232 输出口：用于 Nios 系统 C 程序调试的。与 Cyclone 的管脚连线：RXD 接 P170 脚；TXD 接 176 脚。
5. 适配板上的 SRAM 和 Flash ROM，主要用于 SOPC 设计，作为 Nios CPU 的外围接口存储器，是进行 SOPC Nios 嵌入式系统实验开发用的，本次实验不涉及。

1.3 GW48-PK2 主板

1.3.1 GW48-PK2 主板结构

系统板面主要部件及其使用方法说明如下：（参见附图 1-2 GW48-PK2 主板面板图）

1. 模式选择键：按动该键能使实验板产生 12 种不同的实验电路结构。例如附图 1-3 中模式指示数码管显示 5，表明系统工作在模式 5 的实验电路结构下。

2. 适配板插座：用于安插适配板。可配备其它型号的适配板。

3. ByteBlasterMV 编程配置口：ByteBlasterMV 模式下的 CPLD/FPGA 编程下载口。使用时注意 JP5 上的跳线安插位置。

4. ByteBlasterII 编程配置口：ByteBlasterII 模式下的编程下载口。使用时注意 JP5 上的跳线安插位置。

5. JP5 编程模式选择跳线：当要对 Cyclone 的配置芯片进行编程，应该将跳线接于“ByBtII”端，在将标有“ByteBlasterII”编程配置口同 GWAC6L 适配板上 EPCS4/1 的 AS 模式下载口用 10 芯线连接起来，通过 QuartusII 进行编程。当短路“Others”端时，可以对其它所有器件编程。

6. JP6 编程电压选择跳线：编程下载口的选择跳线。对 5V 器件，如 10K10、10K20、7128S、1032、95108 等，必须选 5.0V。对低于或等于 3.3V 的低压器件，如 1K30、1K100、10K30E、20K300、Cyclone、7128B 等一律选择 3.3V 一端。本次实验应选择 3.3V 一端。

7. 下载开关 SW6：需要下载时必须将开关打到 DLOAD 位置；在 LOCK 位置时，下载口被关闭，这时可以将下载并行线拔下而作它用（已经下载进 FPGA 的文件不会因为并行线拔下而丢失）。

注意：3~7 项中的内容可参见附图 1-3。

8. 并行下载口：通过下载线将 GW48 系统主板与微机的打印机口相连。来自 PC 机的下载控制信号和 CPLD/FPGA 的目标码将通过此口，完成对目标芯片的编程下载。

9. 系统复位键：此键是系统板上负责监控的微处理器的复位控制键，同时也与接口单片机和 LCD 控制单片机的复位端相连。因此兼作单片机的复位键。

10. 时钟频率选择：位于主系统的右小侧，通过短路帽的不同接插方式，使目标芯片获得不同的时钟频率信号。参见附图 1-4。

对于“CLOCK0”，同时只能插一个短路帽，以便选择输向“CLOCK0”的一种频率：信号频率范围：0.5Hz~50MHz。由于 CLOCK0 可选的频率比较多，所以比较适合于目标芯片对信号频率或周期测量等设计项目的信号输入端。右侧座分三个频率源组，它们分别对应三组时钟输入端：CLOCK2、CLOCK5、CLOCK9。例如，将三个短路帽分别插于对应座的 2Hz、1024Hz 和 12MHz，则 CLOCK2、CLOCK5、CLOCK9 分别获得上述三个信号频率。需要特别注意的是，每一组频率源及其对应时钟输入端，分别只能插一个短路帽。也就是说最多只能提供 4 个时钟频率输入 FPGA：CLOCK0、CLOCK2、CLOCK5、CLOCK9。

11. 拨码开关 SW1：平常工作时，除第 4 档“DS8 使能”向下拨（8 数码管显示）外，其余都默认向上。

12. 拨码开关 SW3：用来控制数码管作扫描显示用的。当要将 8 个数码管从原来的重配置可控状态下向扫描显示方式转换时，可以将此拨码开关全部向下拨，然后将左下侧的拨码开关的“DS8 使能”向上拨。

13. 键 1~键 8：信号控制键，此 8 个键受多模式电路控制，不需要加防抖电路。

14. 键 9~键 14：此 6 个键不受多模式电路控制，需要加防抖电路，键输出默认高电平。

15. 数码管 1~8/发光管 D1~D16 : 受多模式电路控制。
16. 扬声器: 与目标芯片的“SPEAKER”端相接, 通过此口可以进行奏乐或了解信号的频率, 它与目标器件的具体引脚号, 应该查阅附录第 3 节的表格。
17. PS/2 接口: 通过此接口, 可以将 PC 机的键盘和/或鼠标与 GW48 系统的目标芯片相连, 从而完成 PS/2 通信与控制方面的接口实验。
18. VGA 视频接口: 通过它可完成目标芯片对 VGA 显示器的控制。
19. TO_ FPGA/ TO_MCU 选择开关: 通过此开关可以进行不同的通信实验。开关拨向 TO_ FPGA 处, RS232 通信口直接与 FPGA 相接; 拨若向 TO_MCU, 则与 89C51 单片机的 P30 和 P31 端口相接。
20. RS-232 串行通讯接口: 用于 FPGA 与 PC 通讯和 SOPC 调试。可以实现 PC 机、单片机、FPGA/CPLD 三者双向通信。
21. +/-12V 电源开关: 在实验板左上角。有指示灯。电源提供对象: 1) 与 082、311 及 DAC0832 等相关的实验; 2) 模拟信号发生源; 3) GW48-DSP/DSP+适配板上的 D/A 及参考电源; 此电源出口可参见附图 1。平时, 此电源必须关闭!

1.3.2 GW48-PK2 主板上部分多模式电路介绍

1. 结构图 NO.0: 目标芯片的 PIO19 至 PIO44 共 8 组 4 位 2 进制码输出, 经外部的 7 段译码器可显示于实验系统上的 8 个数码管。键 1 和键 2 可分别输出 2 个四位 2 进制码。一方面这四位码输入目标芯片的 PIO11~PIO8 和 PIO15~PIO12, 另一方面, 可以观察发光管 D1 至 D8 来了解输入的数值。例如, 当键 1 控制输入 PIO11~PIO8 的数为[^]HA 时, 则发光管 D4 和 D2 亮, D3 和 D1 灭。电路的键 8 至键 3 分别控制一个高低电平信号发生器向目标芯片的 PIO7 至 PIO2 输入高电平或低电平, 扬声器接在“SPEAKER”上, 具体接在哪一引脚要看目标芯片的类型, 这需要查第 3 节的引脚对照表。如目标芯片为 FLEX10K10, 则扬声器接在“3”引脚上。目标芯片的时钟输入未在图上标出, 也需查阅第 3 节的引脚对照表。例如, 目标芯片为 XC95108, 则输入此芯片的时钟信号有 CLOCK0 至 CLOCK9, 共 4 个可选的输入端, 对应的引脚为 65 至 80。具体的输入频率, 可参考主板频率选择模块。此电路可用于设计频率计, 周期计, 计数器等。

2. 结构图 NO.1: 适用于作加法器、减法器、比较器或乘法器等。例如, 加法器设计, 可利用键 4 和键 3 输入 8 位加数; 键 2 和键 1 输入 8 位被加数, 输入的和数和被加数将显示于键对应的数码管 4-1, 相加的和显示于数码管 6 和 5; 可令键 8 控制此加法器的最低位进位。

3. 结构图 NO.2: 可用于作 VGA 视频接口逻辑设计, 或使用数码管 8 至数码管 5 共 4 个数码管作 7 段显示译码方面的实验; 而数码管 4 至数码管 1, 4 个数码管可作译码后显示, 键 1 和键 2 可输入高低电平。

4. 结构图 NO.3: 特点是有 8 个琴键式键控发生器, 可用于设计八音琴等电路系统。也可以产生时间长度可控的单次脉冲。该电路结构同结构图 NO.0 一样, 有 8 个译码输出显示的数码管, 以显示目标芯片的 32 位输出信号, 且 8 个发光管也能显示目标器件的 8 位输出信号。

5. 结构图 NO.4: 适合于设计移位寄存器、环形计数器等。电路特点是, 当在所设计

的逻辑中有串行 2 进制数从 PIO10 输出时，若利用键 7 作为串行输出时钟信号，则 PIO10 的串行输出数码可以在发光管 D8 至 D1 上逐位显示出来，这能很直观地看到串出的数值。

6. 结构图 NO.5: 此电路结构有较强的功能，主要用于目标器件与外界电路的接口设计实验。主要含以 9 大模块：

(1) 普通内部逻辑设计模块。在图的左下角。此模块与以上几个电路使用方法相同，例如同结构图 NO.3 的唯一区别是 8 个键控信号不再是琴键式电平输出，而是高低电平方式向目标芯片输入。此电路结构可完成许多常规的实验项目。

(2) RAM/ROM 接口。在图左上角，此接口对应于主板上，有 1 个 32 脚的 DIP 座，在上面可以插 RAM，也可插 ROM（仅 GW48-GK/PK 系统包含此接口）例如：RAM: 628128；ROM: 27C020、27C040、29C040 等。此 32 脚座的各引脚与目标器件的连接方式示于图上，是用标准引脚名标注的，如 PIO48（第 1 脚）、PIO10（第 2 脚）、OE 控制为 PIO62 等等。注意，RAM/ROM 的使能 CS1 由主系统左边的拨码开关“1”控制。对于不同的 RAM 或 ROM，其各引脚的功能定义不尽一致，即，不一定兼容，因此在使用前应该查阅相关的资料，但在结构图的上方也列出了部分引脚情况，以资参考。

(3) VGA 视频接口。

(4) 两个 PS/2 键盘接口。注意，对于 GW48-CK 系统，只有 1 个，连接方式是下方的 PS/2 口。

(5) A/D 转换接口。

(6) D/A 转换接口。

(7) LM311 接口。

(8) 单片机接口。

(9) RS232 通信接口。

实验中主要使用模式 5 电路，所以将结构图 NO.5 放在附图 1-10 中，请读者参考。

1.4 GW48 系统使用注意事项

1. 闲置不用 GW48 系统时，必须关闭电源，拔下电源插头。
2. 在实验中，当选中某种模式后，按一下右侧的系统复位键，使系统进入相应结构模式工作。
3. 换芯片时不要插反或插错，不要带电插拔，确信插接正确后再打开电源。尽可能不插拔适配板，及实验系统上的芯片。
4. GW48-PK2 主板上，左下角拨码开关 SW1 除第 4 档“DS8 使能”向下拨（8 数码管显示）外，其余都默认向上。

表 1-1 FPGA 实验系统芯片引脚对照表

信号名	GW48-SOPC/DSP EP1C6/1C12 Q240		结 构 图 上 的信号名	GW48-SOPC/DSP EP1C6/1C12 Q240		信号名	GW48-SOPC/DSP EP1C6/1C12 Q240	
	引脚号	引脚 名称		引脚号	引脚 名称		引脚号	引脚 名称
PIO0	233	I/O0	PIO31	136	I/O31	PIO72	184	PIO72
PIO1	234	I/O1	PIO32	137	I/O32	PIO73	185	PIO73
PIO2	235	I/O2	PIO33	138	I/O33	PIO74	186	PIO74
PIO3	236	I/O3	PIO34	139	I/O34	PIO75	187	PIO75
PIO4	237	I/O4	PIO35	140	I/O35	PIO76	216	PIO76
PIO5	238	I/O5	PIO36	141	I/O36	PIO77	215	PIO77
PIO6	239	I/O6	PIO37	158	I/O37	PIO78	188	PIO78
PIO7	240	I/O7	PIO38	159	I/O38	PIO79	195	PIO79
PIO8	1	I/O8	PIO39	160	I/O39	SPEAKER	174	I/O
PIO9	2	I/O9	PIO40	161	I/O40	CLOCK0	28	I/O
PIO10	3	I/O10	PIO41	162	I/O41	CLOCK2	178	I/O
PIO11	4	I/O11	PIO42	163	I/O42	CLOCK5	177	I/O
PIO12	6	I/O12	PIO43	164	I/O43	CLOCK9	175	I/O
PIO13	7	I/O13	PIO44	165	I/O44			
PIO14	8	I/O14	PIO45	166	I/O45			
PIO15	12	I/O15	PIO46	167	I/O46			
PIO16	13	I/O16	PIO47	168	I/O47			
PIO17	14	I/O17	PIO48	169	I/O48			
PIO18	15	I/O18	PIO49	173	I/O49			
PIO19	16	I/O19	PIO60	226	PIO60			
PIO20	17	I/O20	PIO61	225	PIO61			
PIO21	18	I/O21	PIO62	224	PIO62			
PIO22	19	I/O22	PIO63	223	PIO63			
PIO23	20	I/O23	PIO64	222	PIO64			
PIO24	21	I/O24	PIO65	219	PIO65			
PIO25	41	I/O25	PIO66	218	PIO66			
PIO26	128	I/O26	PIO67	217	PIO67			
PIO27	132	I/O27	PIO68	180	PIO68			
PIO28	133	I/O28	PIO69	181	PIO69			
PIO29	134	I/O29	PIO70	182	PIO70			
PIO30	135	I/O30	PIO71	183	PIO71			