第6章 VHDL仿真

前面已经讲述了 VHDL 语法和建模，VHDL 程序作为硬件的描述语言，可以实现仿真测试，包括 RTL 门级仿真和布线后仿真。通过仿真，可以很容易验证 VHDL 程序以及其描述硬件的正确性。本章将讲述如何建立 VHDL 程序的仿真模型和平台，以及 VHDL 语言的具体仿真过程。

6.1 VHDL 仿真概述

当使用 VHDL 设计的数字逻辑电路更复杂时，则仿真验证就非常重要，是保证一个项目设计成功的重要方法。在实际仿真中，需要一个 VHDL 仿真器来实现，目前有很多功能强大的仿真器，而 Mentor Graphics 的 ModelSim 是应用最广泛的仿真器之一，本节将以 ModelSim 6.1 为仿真工具软件来讲述 VHDL 的仿真操作。在编写好了 VHDL 程序，并检查了语法后，就可以进行 VHDL 程序的仿真操作。通过仿真验证，可以为后续的综合和后续的布局布线节省更多的时间，从而保证项目的顺序完成。

VHDL 仿真器通常需要两个输入，即设计的描述（项目的 VHDL 程序）和驱动设计的激励。有时候，设计项目的 VHDL 程序本身可能是一个自激励的程序，不需要外部的激励。但是在大多数情况下，设计工程师需要开发出与设计项目的 VHDL 程序相对应的激励程序，这个激励程序就是测试平台文件（TestBench）。

为了对设计项目进行仿真，在完成了项目的 VHDL 程序开发后，必须编写其测试平台文件。仿真工具会加载 VHDL 测试平台文件和原项目文件，然后进行编译仿真，从而实现硬件的仿真验证。测试平台文件可以是一个简单 VHDL 程序，它和需要仿真的项目文件的实体对应，并且具有相应的激励信号即可。当然测试平台文件也可以是复杂的 VHDL 程序，它可以从磁盘文件中读取数据并将测试结果输出到屏幕或者保存到磁盘文件中。

仿真模型设计的基本结构如图 6-1 所示，顶层的描述包括两个元件，即所测试的设计项目（Design Under Test，DUT）元件和激励驱动器。这些元件之间通过设计项目的实体信号连接。仿真模型结构的顶层并不包括任何外部端口，仅仅是连接两个元件的内部信号。

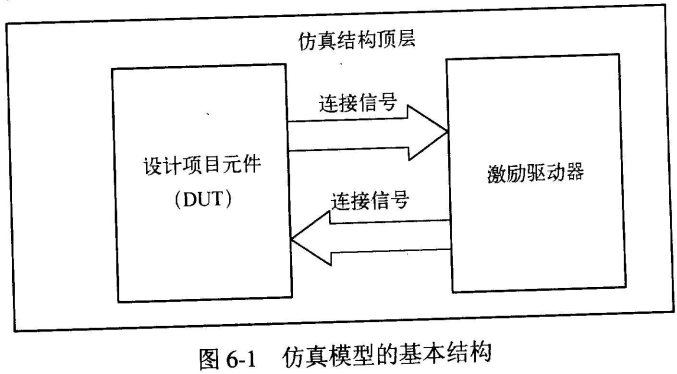

图 6-1 仿真模型的基本结构

图 6-2 描述了 VHDL 的一般仿真过程。首先，可以使用 VHDL 程序编辑器编写 VHDL 程序，或者使用图形编辑器设计出逻辑电路。同时，还需要根据项目设计编写出测试平台文件。然后仿真器会读入文本式的或图形化的 VHDL 项目文件，并读入测试平台文件，进行编译处理。因为 VHDL 项目文件还需要调用相应的库文件，因此仿真器还需要访问 VHDL 库资源。

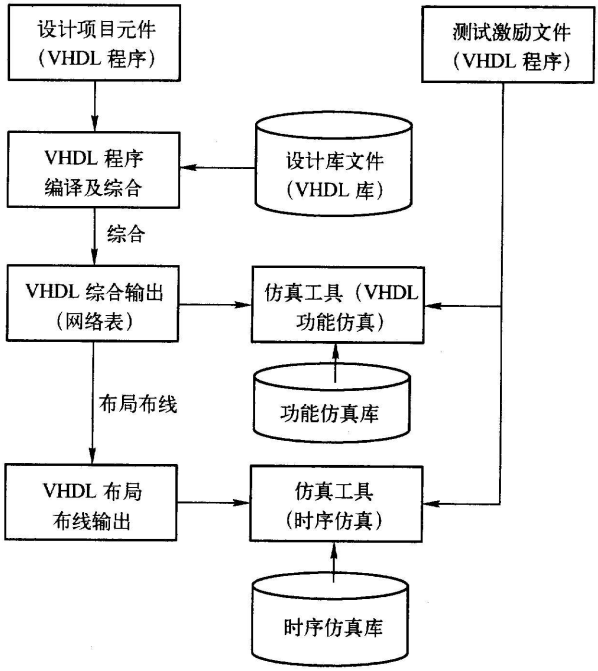

图 6-2 仿真流程图

然后仿真器就可以通过测试平台的激励信号产生驱动信号源，并根据项目设计综合或布局布线的输出，实现功能或时序仿真，并输出仿真波形或者数据。功能仿真是在布局布线前的仿真操作，主要验证 VHDL 设计的功能是否满足设计要求。时序仿真是在布局布线后的仿真操作，主要在布局布线后对信号的时序进行验证分析。

6.2 仿真测试平台文件

一个测试平台文件就是一个 VHDL 模型，可以用来验证所设计的硬件模型的正确性。测试平台文件为所测试的元件提供了激励信号，仿真结果可以以波形的方式显示或存储测试结果到文件中。激励信号可以直接集成在测试平台文件中，也可以从外部文件加载。可以直接使用 VHDL 语言来编写测试平台文件。

6.2.1 测试平台文件的结构

使用 VHDL 语言编写测试平台文件时，所有的基本 VHDL 语法都是适用的，但是测试平台文件与一般的项目设计存在一些区别。一个测试平台文件必须包括与所测试的元件

（DUT）向对应的元件声明，以及输入到 DUT 的激励描述。一个测试平台文件的基本结构如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY TEST_BENCH IS          --测试平台文件的空实体（不需要定义端口）
END TEST_BENCH;
ARCHITECTURE TB_BEHAVIOR OF TEST_BENCH IS
    COMPONENT ENTITY_UNDER_TEST      --被测试元件的声明
        PORT ( list-of-ports-their-types-and-modes);
    END COMPONENT;
    Local-signal-declarations;   --局部信号的声明
BEGIN
    Instantiation: ENTITY_UNDER_TEST port map ( port-associations );
                                          --被测试元件的例化或映射
    PROCESS()                    --产生时钟信号
    ......
    END PROCESS;
    PROCESS()                    --产生激励源
    ......
    END PROCESS;
END TB_BEHAVIOR;
```

从上面的基本结构中，可以看出其中包含几个最基本的语句，即实体的定义、所测试元件的例化、产生时钟信号和产生激励源等语句。测试平台中的实体定义不需要定义端口，也就是说测试平台没有输入输出端口，它只是和被测试元件（DUT）通过内部信号相连接。

下面的实例程序即为一个测试平台程序以及它所测试的元件。

（1）测试平台文件。时钟周期为 20ns，在一个时钟波形产生的进程中定义。激励信号波形在另一个进程中产生。实体为一个空实体，没有输入输出信号端口。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY Counter_TB_vhd IS        --测试平台实体
END Counter_TB_vhd;
ARCHITECTURE behavior OF Counter_TB_vhd IS
    -- 被测试元件（DUT）的声明
    COMPONENT sim_counter
    PORT(clk : IN std_logic;
         reset : IN std_logic;
         count : OUT std_logic_vector(3 downto 0));
    END COMPONENT;
    --输入信号
    SIGNAL clk :   std_logic := '0';
    SIGNAL reset :   std_logic := '0';
```

```
    --输出信号
    SIGNAL count :   std_logic_vector(3 downto 0);
    constant clk_period : time :=20 ns;          --时钟周期的定义
BEGIN
    --被测试元件（DUT）的例化
    DUT: sim_counter PORT MAP(clk => clk, reset => reset, count => count);
    clk_gen: process          --产生时钟信号
    BEGIN
        clk <= '1';
        wait for clk_period/2;
        clk <='0';
        wait for clk_period/2;
    END PROCESS;
    TB : PROCESS          --激励信号
    BEGIN
        wait for 20 ns;
        RESET<='1';
        wait for 20 ns;
        RESET<='0';
        wait for 200 ns;
        wait; -- will wait forever
    END PROCESS;
END;
```

（2）定义的所测试文件的 VHDL 程序，该程序是一个简单的计数程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Sim_counter IS
    PORT (clk: IN STD_LOGIC;
          reset: IN STD_LOGIC;
          count: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
  );
END Sim_counter;
ARCHITECTURE Behavioral OF Sim_counter IS
SIGNAL Temp: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
    PROCESS (clk, reset)
    BEGIN
        if (reset = '1') then
          Temp <= "0000";
        elsif(clk'event and clk = '1') then
          Temp <= Temp + 1;
        end if;
    END PROCESS;
```

```
    count<= Temp;
END Behavioral;
```

在上面所定义的测试平台文件的基础上，当使用 ModelSim 对上面所定义的元件进行仿真，所得到的仿真波形如图 6-3 所示。

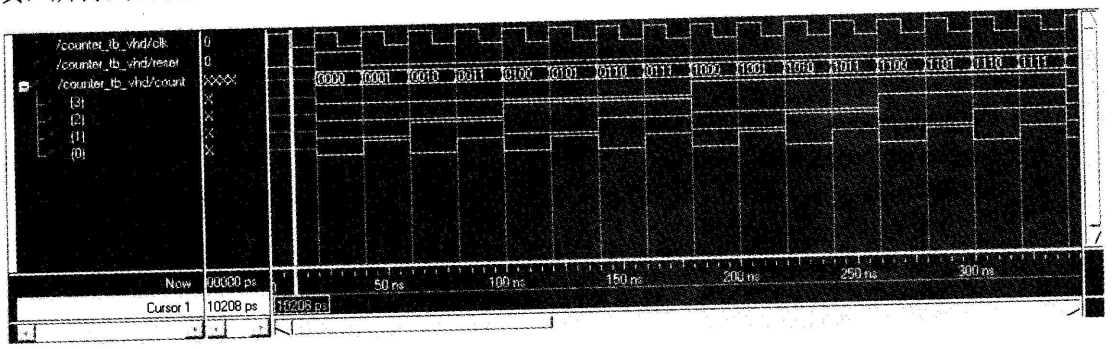

图 6-3　上面的计数器实例仿真结果

6.2.2　激励信号的产生

有两种方式产生激励信号，一种是以一定的离散事件间隔产生激励信号的波形，另一种是基于实体的状态产生激励信号，也就是说基于实体的输出响应产生激励信号。

在测试平台文件中，有两种常用的激励信号，一种是周期性的激励信号，其波形是周期性变化的；另一种是时序变化的，比如复位信号以及其他输入信号。下面各实例来讲述激励信号的产生。

1．时钟信号

一个周期性的激励信号可以使用一个并行的信号赋值语句来建立，例如下面的语句即建立周期为 40ns 的信号，其波形如图 6-4 所示。

```
A <= not A after 20 ns;          -- 产生一个周期为 40ns 的信号 A
```

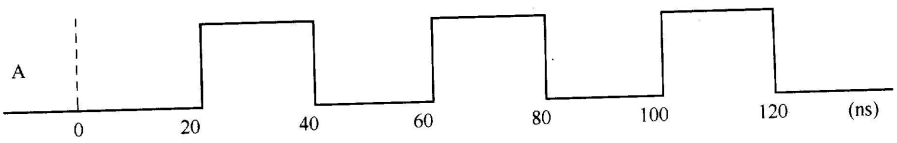

图 6-4　周期性的信号波形

时钟信号是同步设计中最重要的信号之一。它既可以使用并行的信号赋值语句产生（如上面的语句），也可以使用时钟产生的进程来实现定义。当使用并行的信号赋值语句时，产生的时钟信号可以是对称的或不对称的，但是信号的初始值不能为'u'，它的初始值必须是明确声明的（'1'或'0'）；如果使用进程来定义时钟信号，也可以产生各种时钟信号，包括对称和不对称的。

在大部分情况下，时钟信号是一直运行的，并且是对称的。当定义不对称的时钟信号，如果使用并行信号赋值语句，则需要使用条件信号赋值语句；如果使用进程，则比较简单，适用顺序逻辑就可以。例如下面的语句，使用了条件信号赋值语句，定义了一个 25%占空比

的时钟信号。

```
W_CLK <= '0' after PERIOD/4 when W_CLK='1' else
         '1' after 3*PERIOD/4 when W_CLK='0' else
         '0';
```

上述的两个时钟信号，即对称的和不对称的时钟信号，也可以使用进程来定义，下面的语句即可以分别实现上述的并行语句所定义的时钟信号。

```
clk_gen1: process        --产生对称的时钟信号，周期为 40ns
constant clk_period: TIME := 40 ns;
BEGIN
    clk <= '1';
    wait for clk_period/2;
    clk <='0';
    wait for clk_period/2;
END PROCESS;
clk_gen2: process        --产生非对称的时钟信号，周期为 40ns，占空比为 25%
constant clk_period: TIME := 40 ns;
BEGIN
    clk <= '1';
    wait for clk_period/4;
    clk <='0';
    wait for 3*clk_period/4;
END PROCESS;
```

2. 复位信号

在仿真开始时，通常需要使用复位信号对系统进行复位，以便初始化系统。通常复位信号可以以并行赋值语句来实现，也可以在进程中设定。例如下面的复位信号的设置：仿真开始时，复位信号为'0'；经过 20ns 后，复位信号变为'1'；再经过 20ns 后，复位信号变为'0'，其波形如图 6-5 所示。

```
RESET <= '0', '1' after 20 ns, '0' after 40 ns;
```

图 6-5　复位信号波形

再看另一个复位信号设置实例，例如下面的代码。RESET 信号初始为'0'，经过 100ns 后，变为'1'；再经过 80ns，该信号变为'0'；再经过 30ns，该信号返回到'1'。信号波形如图 6-6 所示。

```
RESET <= '0', '1' after 100 ns, '0' after 180 ns, '1' after 210 ns;
```

3. 周期性的信号

可以在进程中使用信号赋值语句实现信号的周期性信号设置。例如下面的实例代码，定义了两个周期性信号，为了实现信号的周期性变化，后面使用一个 WAIT 语句。其波形如图

6-7 所示。

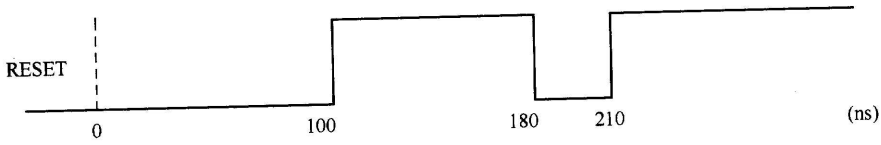

图 6-6　复位信号波形

```
signal CLK1, CLK2: STD_LOGIC := '0';
 ...
TWO_PHASE: PROCESS
BEGIN
     CLK1 <= '1' after 5 ns, '0' after 10 ns, '1' after 20 ns,'0' after 25 ns;
     CLK2 <= '1' after 10 ns, '0' after 20 ns, '1' after 25 ns,'0' after 30 ns;
     wait for 35 ns;
END PROCESS;
```

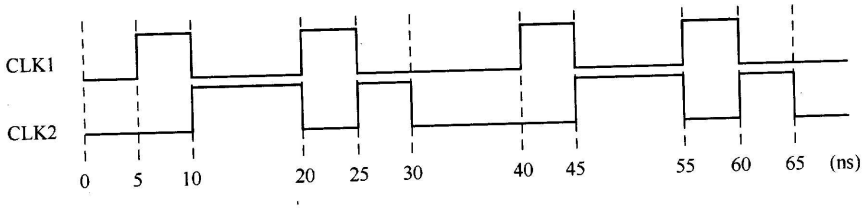

图 6-7　产生的两周期性的波形

4. 使用延迟 DELAYED

还可以使用预定义属性 DELAYED 关键词来产生信号。比如已经产生了一个时钟信号，在这个时钟信号的基础上，可以使用 DELAYED 来使已产生的时钟信号延迟一定的时间，从而获得另一个时钟信号。

例如我们已经使用如下的语句定义了一个时钟信号 W_CLK。

```
W_CLK <= '1' after 30 ns when W_CLK='0' else
         '0' after 20 ns;
```

然后可以使用如下的延迟语句获得一个新的时钟信号 DLY_W_CLK，它比 W_CLK 延迟了 10ns。这两个时钟信号波形如图 6-8 所示。

```
DLY_W_CLK <= W_CLK'DELAYED (10 ns);
```

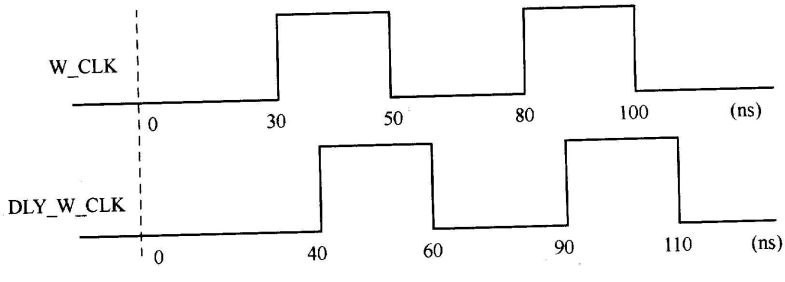

图 6-8　使用延迟定义的时钟信号波形

5. 一般的激励信号

可以定义普通的激励信号来用作模型的输入信号。定义一般的激励信号通常在进程中定义。一般都可以使用 WAIT 语句来定义一般的激励信号。例如下面的激励信号的定义，其波形如图 6-9 所示。

```
SIGNAL C: STD_LOGIC := '0';
    ……
    STIMULI : PROCESS
    BEGIN
        wait for 80 ns;
        C<='1';
        wait for 50 ns;
        C<='0';
        wait for 60 ns;
        C<='1';
        wait for 120 ns;
        C<='0';
        ……
        wait; -- 一直等待
    END PROCESS;
```

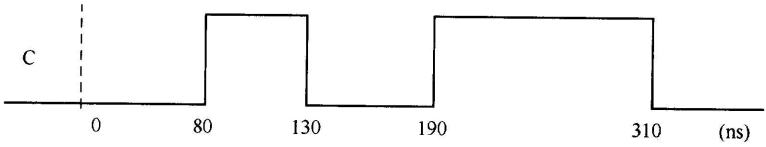

图 6-9　一般激励信号的波形

6. 动态激励信号

动态激励信号就是与被仿真的实体（DUT）的行为模型相关，即 DUT 的输入激励信号受模型的行为所影响。比如下面的信号定义，模型的输入信号 Sig_A 和模型输出信号 Count 相关。

```
PROCESS(Count)
BEGIN
    CASE Count IS
        when 2 =>   Sig_A <= '1' after 10 ns;
        when others => Sig_A <= '0' after 10 ns;
    END CASE;
END PROCESS;
```

【例 6-1】　移位寄存器的仿真。

下面以一个 4 位的双向移位寄存器为例来进一步说明测试平台文件和激励信号的定义。双向移位寄存器的 VHDL 建模如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

```
ENTITY Bi_shift IS
        PORT (clk, clear : IN STD_LOGIC;
              rsi, lsi : IN STD_LOGIC;
              direction : IN STD_LOGIC;
              q : out STD_LOGIC_VECTOR(3 downto 0));
END Bi_shift;
ARCHITECTURE Behavioral of Bi_shift IS
Signal Temp : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
    PROCESS (clk, clear)
    BEGIN
        IF clear = '0' THEN
            q <= "0000"; -- asynchronous clear
            Temp <= "0000";
        ELSIF (clk'EVENT and clk = '1') THEN
            CASE direction IS
                WHEN '0' =>
                    Temp <= Temp(2 downto 0) & lsi;   -- 左移
                    q <= Temp;
                WHEN '1' =>
                    Temp <= rsi & Temp(3 downto 1); -- 右移
                    q <= Temp;
                WHEN others =>
                    NULL;
            END CASE;
        END IF;
    END PROCESS;
END Behavioral;
```

为了对上面的移位寄存器模型进行仿真，可以定义如下的测试平台文件，对模型进行仿真，仿真输出波形如图 6-10 所示。

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY Bishift_TB_vhd IS
END Bishift_TB_vhd;

ARCHITECTURE behavior OF Bishift_TB_vhd IS
    COMPONENT bi_shift
    PORT(
        clk : IN std_logic;
        clear : IN std_logic;
        rsi : IN std_logic;
```

```
        lsi<='1';
        wait for 160 ns;
        lsi<='0';
        clear<='0';
        wait for 20 ns;
        clear<='1';
        wait for 20 ns;
        direction<='1';
        wait for 20 ns;
        rsi<='1';
        wait for 20 ns;
        rsi<='0';
        wait for 20 ns;
        lsi<='1';
        wait for 20 ns;
        lsi<='0';
        wait for 200 ns;
        rsi<='1';
        wait for 200 ns; -- 等待 200ns
    END PROCESS;
END;
```

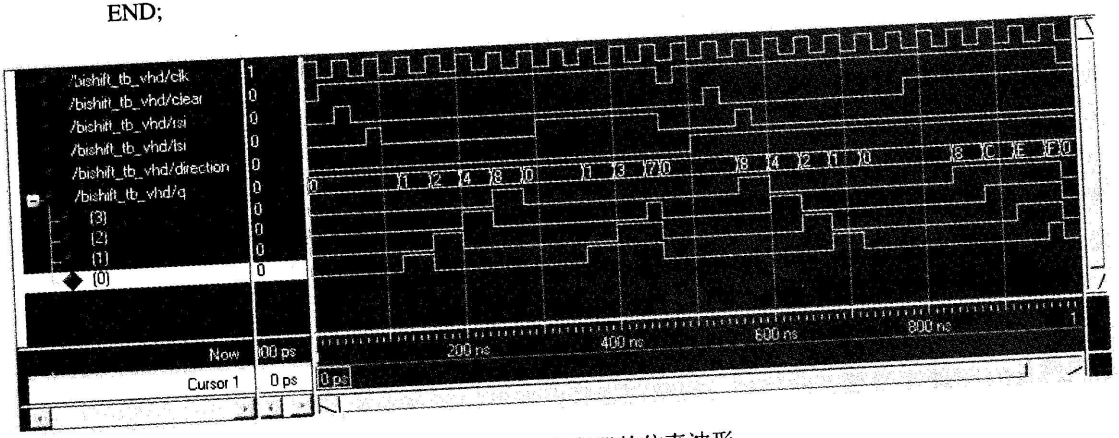

图 6-10　移位寄存器的仿真波形

6.2.3　使用仿真工具的波形编辑器

大部分仿真工具或综合工具提供了测试仿真用的波形编辑器，设计人员可以直接使用波形编辑器产生激励信号的波形。下面以 Xilinx 的 ISE7.1 为例来讲述直接使用波形编辑器来设置仿真的输入和时钟波形。当创建测试平台文件时，可以直接选择 Test Bench Waveform 选项，然后系统会弹出如图 6-11 所示的对话框，即为 Xilinx 的 ISE7.1 的时钟初始化对话框，此时可以定义时钟的波形，包括时钟周期、高电平和低电平时间等。

定义了时钟信号后，单击“OK”按钮，然后就会进入波形编辑界面，如图 6-12 所示。此时可以直接在输入信号的波形上设置其二进制值，设置需要的输入激励波形。

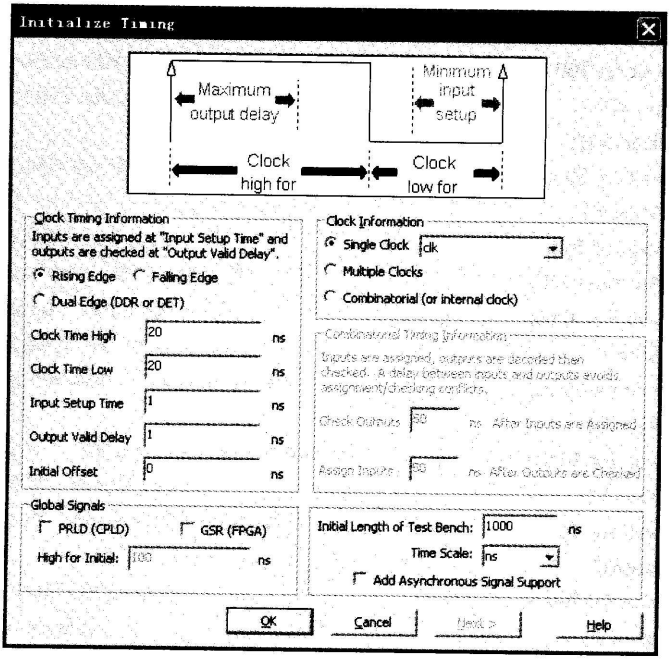

图 6-11　Xilinx 的 ISE7.1 的波形编辑器

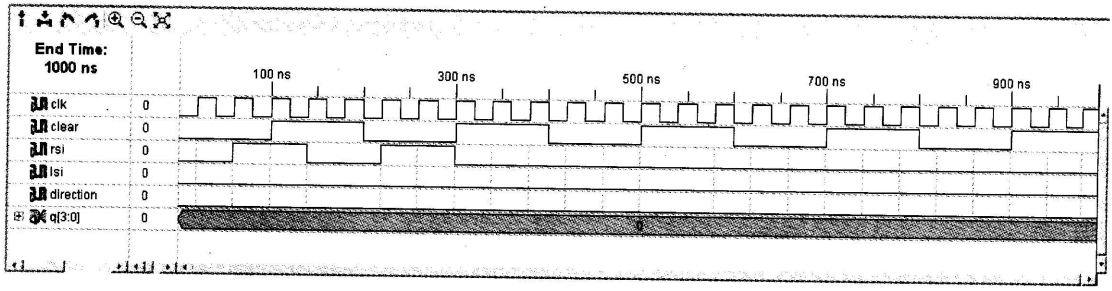

图 6-12　ISE7.1 波形编辑界面

6.2.4　使用测试矢量

另外一种产生激励信号的方法是使用测试矢量。即将一组固定的输入输出矢量值存储在一个常量表或一个 ASCII 文件中，然后将这些值应用到输入信号从而产生激励信号。矢量的值序列可以使用多维数组或使用多列记录来描述。例如下面的数据表存储了输入矢量。

```
CONSTANT NO_OF_BITS: INTEGER := 4;
CONSTANT NO_OF_VECTORS: INTEGER := 5;
TYPE TABLE_TYPE IS ARRAY (1 TO NO_OF_VECTORS) OF STD_LOGIC_VECTOR(1 TO
NO_OF_BITS);
CONSTANT VECTOR_PERIOD: TIME := 100 ns;
CONSTANT INPUT_VECTORS: TABLE_TYPE := ("1001", "1000", "0010", "0000", "0110");
SIGNAL INPUTS: STD_LOGIC_VECTOR (1 TO NO_OF_BITS);
SIGNAL A, B, C: STD_LOGIC;
```

```
SIGNAL D: STD_LOGIC_VECTOR (0 TO 1);
```

假设所测试的实体（DUT）具有 4 个输入：A、B、C 和 D 信号。如果以一般的时间间隔应用测试矢量，则可以使用一个 GENERATE（产生）语句，例如，

```
G1: for J in 1 to NO_OF_VECTORS generate
    INPUTS <= INPUT_VECTORS(J) after (VECTOR_PERIOD * J);
END GENERATE G1;
A<= INPUTS(1);
B <= INPUTS(4);
C<=INPUTS(1);
D<=INPUTS(2 to 3);
```

如果将矢量应用于任意时间间隔，则需要使用并行的信号赋值语句产生多个信号的波形。使用这种方法可以将一个矢量赋值给多个信号。例如下面的代码：

```
INPUTS <= INPUT_VECTORS(1) after 10 ns,
INPUT_VECTORS(2) after 25 ns,
INPUT_VECTORS(3) after 30 ns,
INPUT_VECTORS(4) after 32 ns,
INPUT_VECTORS(5) after 40 ns;
```

6.3 仿真响应

在仿真时，如果对执行的仿真没有任何控制，则仿真会一直持续到时间等于设定的仿真时间。如果想在某个时间终止仿真，则可以使用断言语句 ASSERT 来实现。另外，当对 VHDL 模型进行仿真时，如果想对某些值或行为做出响应，同样可以使用断言语句来实现。

断言语句 ASSERT 是最适合与执行仿真的自动响应的。一个断言语句可以检查一个条件并报告信息。根据所选择的严重级别和仿真工具的设置，在 ASSERT 语句报告了信息后，仿真可以继续执行（警告级别 WARNING）或者停止（错误 ERROR 和致命错误 FAILURE），默认的严重级别为 ERROR。详细的断言语句的语法和应用讲解请参考 3.2.13 节。

下面的实例语句为一个简单断言语句在仿真时的应用，它判断了仿真的时间，如果当前时间为 1000ns，则仿真完成，使用 ERROR 严重级终止仿真过程。

```
PROCESS
BEGIN
    ASSERT (NOW <= 1000 ns)
        REPORT "Simulation completed successfully"
        SEVERITY ERROR;
END PROCESS
```

断言语句判断条件时，如果条件的判断结果为 FALSE，则执行后面的报告以及严重级语句，否则仿真会忽略后面的报告和严重级语句并继续执行。

可以使用 ASSERT 语句设定一个判断条件，以便对仿真的某个结果或值做出响应，例如下面的实例程序，如果信号 q 等于“1001”，则终止仿真，并输出"The shifter gets the result!!"。

```
PROCESS(q)
BEGIN
    ASSERT (q = "1001")
        REPORT "The shifter gets the result!!"
        SEVERITY ERROR;
END PROCESS;
```

如果这段程序添加到实例 6-1 的测试平台文件中，则移位寄存器的输出结果等于"1001"，仿真就会被终止，不管仿真时间是否结束。

下面以一个简单的实例来讲述使用断言语句来响应一个仿真过程。

【例 6-2】 4 位上下计数器的仿真。

下面的程序为 4 位上下计数器的行为模型，计数的位数为 4 位。方向由信号 DIR 决定，如果 DIR 为高电平，则正向计数；如果 DIR 为低电平，则反向计数。计数结果保存在 Ct_Result 信号中。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY counter IS
    PORT(CLK, CLR, DIR : IN STD_LOGIC;
         Ct_result : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END counter;
ARCHITECTURE Behavioral OF counter IS
SIGNAL tmp: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
    PROCESS (CLK, CLR)
    BEGIN
        if (CLR='1') then
            tmp <= "0000";
        elsif (CLK'event and CLK='1') then
            if (DIR='1') then
                tmp <= tmp + 1;
            else
                tmp <= tmp - 1;
            end if;
        end if;
    END PROCESS;
    Ct_result <= tmp;
END Behavioral;
```

下面的程序为测试平台。在该程序中，一个断言语句用于判断计数结果是否等于"1001"。条件判断使用了“不等于（≠）”逻辑，如果判断条件为 FALSE，即等于"1001"，则报告信息，并终止仿真。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
USE IEEE.NUMERIC_STD.ALL;

ENTITY Counter_TB_vhd IS
END Counter_TB_vhd;
ARCHITECTURE behavior OF Counter_TB_vhd IS
    COMPONENT counter
    PORT(
        CLK : IN std_logic;
        CLR : IN std_logic;
        DIR : IN std_logic;
        Ct_result : OUT std_logic_vector(3 downto 0)
        );
    END COMPONENT;
    --输入信号
    SIGNAL CLK :   std_logic := '0';
    SIGNAL CLR :   std_logic := '0';
    SIGNAL DIR :   std_logic := '0';
    --输出信号
    SIGNAL Ct_result :   std_logic_vector(3 downto 0);
    --常量
    constant clk_period: TIME := 40 ns;
BEGIN
    uut: counter PORT MAP(
        CLK => CLK,
        CLR => CLR,
        DIR => DIR,
        Ct_result => Ct_result
    );
    clk_gen1: process       --产生对称的时钟信号，周期为 40ns
    BEGIN
    CLK <= '1';
        wait for clk_period/2;
        CLK <='0';
        wait for clk_period/2;
    END PROCESS;
    tb : PROCESS            --    激励信号定义
    BEGIN
        -- Wait 100 ns for global reset to finish
        CLR<='1';
        DIR<='1';
        wait for 20 ns;
        CLR<='0';
        wait for 280 ns;
        DIR<='0';
        wait for 320 ns;
        wait; -- will wait forever
```

```
    END PROCESS;
    PROCESS(Ct_result)          --仿真响应，使用断言语句 ASSERT
    BEGIN
        ASSERT (Ct_result /= "1001")
        REPORT "The counter gets to nine!!"
        SEVERITY ERROR;
    END PROCESS;
END;
```

此仿真波形如图 6-13 所示。当计数达到"1001"，仿真工具会在信息栏输出索要报告的信息，即如下信息：

```
# ** Error: The counter gets to nine!!
#    Time: 840 ns  Iteration: 3  Instance: /counter_tb_vhd
```

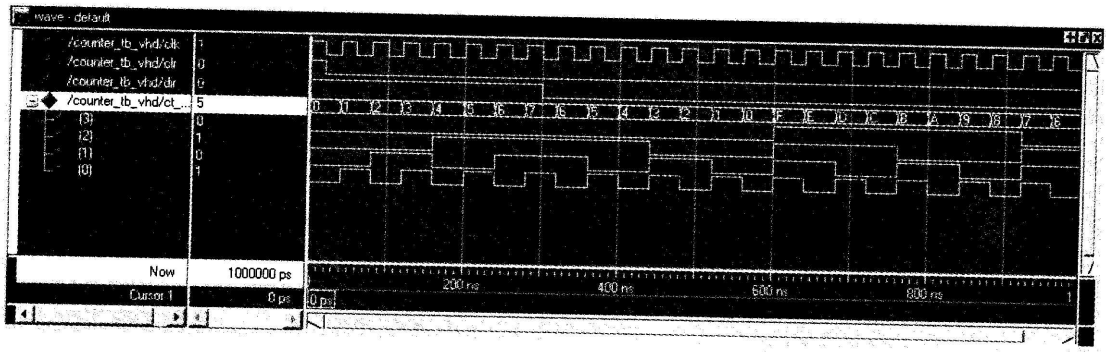

图 6-13　上下计数器的仿真波形

6.4　文件 I/O 的读写

仿真时，VHDL 允许设计人员从文件加载数据或将数据存储到文件中。比如用户定义的测试矢量可以保存在文件中，然后在仿真时从文件中读取这些测试矢量。另外，仿真的结果可以保存在文件中。VHDL'93 的文件 I/O 读写主要是用于仿真，综合工具并不支持文件 I/O 读写。

如果想在仿真时进行文件操作，必须包括标准库 STD 中的 TEXTIO 定义的程序库，该程序包中包含了文件输入输出所需的基本子程序（函数和过程）。VHDL'93 标准包括如下重要的文件 I/O 操作的子程序：

```
READLINE(...), READ(...), WRITELINE(...), WRITE(...), ENDFILE(...)
```

这几个函数或过程可以实现文件 I/O 访问，它们是标准库中预定义，包含了许多预定义的 VHDL 数据类型。文件 I/O 可以使 VHDL 功能和时序仿真更加灵活，因为如果使用文件 I/O 进行仿真操作，那么仿真的激励信号和响应分析数据都可以保存在文件，这样就可以不依赖于仿真工具软件进行仿真响应分析，并且激励信号也可以灵活在文件中定义，而不用去修改测试平台文件。