怎样写 testbench

本文的实际编程环境：**ISE 6.2i.03**
ModelSim 5.8 SE
Synplify Pro 7.6
编程语言 **VHDL**
在 **ISE** 中调用 **ModelSim** 进行仿真

一、 基本概念和基础知识

Testbench 不仅要产生激励也就是输入，还要验证响应也就是输出。当然也可以只产生激励，然后通过波形窗口通过人工的方法去验证波形，这种方法只能适用于小规模的设计。

在 **ISE** 环境中，当前资源操作窗显示了资源管理窗口中选中的资源文件能进行的相关操作。在资源管理窗口选中了 **testbench** 文件后，在当前资源操作窗显示的 **ModelSim Simulator** 中显示了 **4** 种能进行的模拟操作，分别是：**Simulator Behavioral Model**（功能仿真）、**Simulator Post-translate VHDL Model**（翻译后仿真）、**Simulator Post-Map VHDL Model**（映射后仿真）、**Simulator Post-Place & Route VHDL Model**（布局布线后仿真）。如图 **1** 所示：

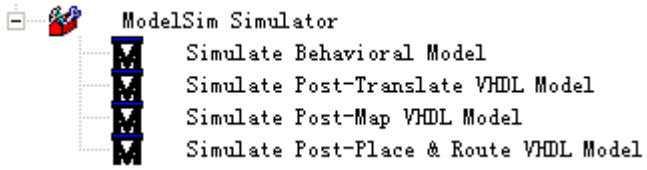

图 **1**

- **Simulator Behavioral Model** 也就是所说的功能仿真、行为仿真、前仿真。验证功能是否正确，这是设计的第一步。功能仿真正确的程序不一定能被正确综合，也就是硬件实现。有的在综合时报错误，有的虽然能综合但结果并不正确。当然，功能仿真如果都不能通过，以后的步骤也就无法进行。这是必做的仿真。

- **Simulator Post-translate VHDL Model** 也就是翻译后仿真。对源程序进行编译后首先排除了语法错误，对一些像类属命令（**Generic**）、生成语句（**Generate**）等进行了展开。不是必做的仿真。

- **Simulator Post-Map VHDL Model** 也就是映射后仿真。不同的器件内部结构也不尽相同，映射的作用就是将综合后产生的网表文件对应到实际的器件上去。由于映射不包含布线，也就是要用什么类型的逻辑单元虽然已经确定但要用哪个位置的还没有确定，因此，映射后仿真不包含布线延时。不是必做的仿真。

- **Simulator Post-Place & Route VHDL Model** 也就是所说的布局布线后仿真、时序仿真、后仿真。这是最完整的仿真，既包含逻辑延时又包含布线延时。在做布局布线后仿真时要用到一个叫 **SDF** 的文件。**SDF** 文件包含设计中每个单元（**Cell**）的延

时和时序约束数据。通过加载这个文件就能得到完整的时序情况。它是必做的仿真。

一般必须进行功能仿真和布局布线后仿真。

常见问题：

为什么有的 **testbench** 在进行功能仿真时能正确进行，而在进行布局布线后仿真时就不能运行。有两点要注意的地方：（**1**）、在做映射后仿真或布局布线后仿真时，都已经经过了综合工具的综合，源程序中的类属命令（**Generic**）、生成语句（**Generate**）等都已经进行展开。例如，如果用 **Generic** 定义了一个参数 **width**，综合工具进行综合时已经按照一个确定的 **width** 值进行了综合。它生成的电路已经具有一个确定的结构，不能再随意调整。所以在映射后仿真和布局布线后仿真的 **testbench** 中，往往不能出现 **Generic** 语句。（**2**）映射后仿真和布局布线后仿真都要用到 **SDF** 文件，并且要将 **SDF** 文件关联到设计中的实例。所以在映射后仿真和布局布线后仿真的 **testbench** 中，第一，要将你的设计声明成一个元件。第二，实例化你设计的元件并且实例名要取为 **UUT**（默认的，当然也可以改）。

关于断言语句

在仿真中为了能得到更多信息，经常要用到断言语句（**assert**）。其语法如下：

Assert<条件>
Report<消息>
Severity<出错级别>;

出错级别共有 **5** 种：

- **Note**
- **Warning**
- **Error**
- **Failure**
- **Fatal**

在 **VHDL** 模型的模拟过程中，一旦断言语句的条件为假，则发送消息并将出错级别发送给模拟器。通常可以设置一个中止模拟器运行的出错级别，一般默认的中止运行的出错级别为 **Failure**。

我们来看一个例子：

```
assert false
    report "********* " & IMAGE(DWIDTH) & "BIT DIVIDER SEQUENCE FINISHED
    AT " & IMAGE(now) & " !" & " *********"
severity note;
```

断言的条件不是一个条件表达式，而直接是 **false**。这说明只要程序执行到这里断言就一定会成立，送出消息。出错级别为 **note**，在模拟器的输出窗口将会显示：

```
# ** Note: ********* 8        BIT DIVIDER SEQUENCE FINISHED AT 52428800 ns        ! *********
```

图 **2**

再看一个例子：

```
assert (s_cyi((DWIDTH-1)/4) = '0')
```

```
        and (s_ovi = '0')
        and (s_qutnt = conv_std_logic_vector(v_quot,DWIDTH))
        and (s_rmndr = conv_std_logic_vector(v_remd,DWIDTH))
        report "ERROR in division!"
    severity failure;
```

断言的条件有 4 个并且是与的关系，只要其中一个条件不成立则整个表达式为假，断言成立。如果断言成立将输出“ERROR in division!“这个消息。并且通知模拟器出错级别为 failure，这一般会停止模拟。这个断言实际是在对结果进行验证。

二、实际 testbench 分析

下面将详细分析一个实际的 testbench，它是用来测试 8051 的 ALU 单元的除法功能的。8 位的除法器，被除数和除数的组合共有 256×256＝65536 种。我们采用的方法是穷举所有的输入组合，这样的代码覆盖率可以达到 100%。它的验证必须通过程序自动完成，否则通过人工方法工作量太大。

把要测试的程序当作一个元件，例如想象成一个 74 系列数字电路。Testbench 的作用是在被测试电路的输入端加上激励，然后比较被测试电路的输出和计算出来的期望值是否一致。对我们这个例子来说，在要仿真的 ALU 输入端产生 65536 种输入组合，然后将 ALU 产生的对应输出值和 testbench 算出的期望值相比较，如果有错误产生则停止模拟并输出信息。ALU 的除法单元的输入有 4 个，分别是被除数、除数、进位、溢出位；输出也有 4 个，分别是商、余数、新的进位、新的溢出位。

1、 testbench 的输出 s_dvdnd（被除数）、s_dvsor（除数）、s_cyo（进位）、s_ovo（溢出位）连接到 ALU 的输入 acc_i（被除数）、ram_data_i（除数）、cy_i（进位）、ov_i（溢出位）；

2、 testbench 的输出 s_qutnt（商）、s_rmndr（余数）、s_cyi（进位）、s_ovi（溢出位）连接到 ALU 的输出 result_a_o（商）、result_b_o（余数）、new_cy_o（进位）、new_ov_o（溢出位）。

3、 总之，testbench 驱动被测试单元，同时对被测试单元的输出进行验证。

```
4、assert (s_cyi((DWIDTH-1)/4) = '0')
        and (s_ovi = '0')
        and (s_qutnt = conv_std_logic_vector(v_quot,DWIDTH))
        and (s_rmndr = conv_std_logic_vector(v_remd,DWIDTH))
        report "ERROR in division!"
    severity failure;
```

根据 51 指令系统规定，除法运算的 cy 位固定为 0，如果除数为 0 则 ov 置 1，否则置 0。程序中

```
s_qutnt = conv_std_logic_vector(v_quot,DWIDTH)
s_rmndr = conv_std_logic_vector(v_remd,DWIDTH)
```

用来对运算结果进行比较。conv_std_logic_vector（）是类型转换函数。

——首先是对库的引用

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
library work;
use work.mc8051_p.all;
library STD;
use STD.textio.all;

--定义结构体，testbench 程序的结构体是空的。因为 testbench 是用来仿真的，不存在
--对外的接口，所以 entity 是空的。但是必须要有，这是语法的要求。
entity TBX_mc8051_alu is
end TBX_mc8051_alu;

------------------------------------------------------------------------------
architecture TBX_ARCH_DIV of TBX_mc8051_alu is
--定义元件，映射后仿真和布局布线后仿真要使用 SDF 文件，必须指定实例名。要实例
--化元件首先必须定义元件。
component mc8051_alu
    port (
        new_ov_o : out STD_LOGIC;                          --新的 ov 位，输出
        ov_i : in STD_LOGIC := 'X';                        --ov 位，输入
        new_cy_o : out STD_LOGIC_VECTOR ( 1 downto 0 );    --新的 cy 位，输出
        acc_i : in STD_LOGIC_VECTOR ( 7 downto 0 );        --acc，输入
        rom_data_i : in STD_LOGIC_VECTOR ( 7 downto 0 );   --rom_data，输入
        cmd_i : in STD_LOGIC_VECTOR ( 5 downto 0 );        --命令，输入
        ram_data_i : in STD_LOGIC_VECTOR ( 7 downto 0 );   --ram_data，输入
        cy_i : in STD_LOGIC_VECTOR ( 1 downto 0 );         --cy，输入
        result_b_o : out STD_LOGIC_VECTOR ( 7 downto 0 );  --结果 b，输出
        result_a_o : out STD_LOGIC_VECTOR ( 7 downto 0 )   --结果 a，输出
    );
end component;

--定义函数
------------------------------------------------------------------------------
    --
--  IMAGE - Convert a special data type to string                           --
    --
    This function uses the STD.TEXTIO.WRITE procedure to convert different --
--  VHDL data types to a string to be able to output the information via   --
--  a report statement to the simulator.                                   --
--  (VHDL'93 provides a dedicated predefinded attribute 'IMAGE)            --
------------------------------------------------------------------------------
--定义了一个函数 IMAGE，之所以有两个定义，是因为对 IMAGE 进行了重载，一个是把
--time 变量转换成 string，另一个是把 integer 变量转换成 string
function IMAGE (constant tme : time) return string is
    variable v_line : line;
    variable v_tme  : string(1 to 20) := (others => ' ');
```

```
begin
    write(v_line, tme);
    v_tme(v_line.all'range) := v_line.all;
    deallocate(v_line);
    return v_tme;
end IMAGE;

function IMAGE (constant nmbr : integer) return string is
    variable v_line : line;
    variable v_nmbr  : string(1 to 11) := (others => ' ');
begin
    write(v_line, nmbr);
    v_nmbr(v_line.all'range) := v_line.all;
    deallocate(v_line);
    return v_nmbr;
end IMAGE;

--定义过程，它产生所有的测试输入数据并将产生结果和期望值进行比较，如果有误
--产生，模拟将停止并输出一个错误信息。注意到 testbench 要产生被测试单元的输入
--信号，因此 testbench 的输出接到被测试单元的输入，被测试单元的输出接到
--testbench 的输入。
procedure PROC_DIV_ACC_RAM (
    constant DWIDTH     : in  positive;
    constant PROP_DELAY : in  time;
    signal   s_cyi      : in  std_logic_vector;
    signal   s_ovi      : in  std_logic;
    signal   s_qutnt    : in  std_logic_vector;
    signal   s_rmndr    : in  std_logic_vector;
    signal   s_cyo      : out std_logic_vector;
    signal   s_ovo      : out std_logic;
    signal   s_dvdnd    : out std_logic_vector;
    signal   s_dvsor    : out std_logic_vector;
    signal   s_dvdr_end : out boolean) is

    variable v_quot : integer;
    variable v_remd : integer;
    variable v_flags     : std_logic_vector((DWIDTH-1)/4+1 downto 0);

begin
    s_dvdr_end <= false;
    for j in 0 to 2**DWIDTH-1 loop
        s_dvdnd <= conv_std_logic_vector(j,DWIDTH); --产生被除数
        for i in 0 to 2**DWIDTH-1 loop
        s_dvsor <= conv_std_logic_vector(i,DWIDTH); --产生除数
```

```
            for f in 0 to 2**(((DWIDTH-1)/4)+2)-1 loop  ——产生 cy 和 ov
                v_flags := conv_std_logic_vector(f,((DWIDTH-1)/4)+2);
                s_cyo <= v_flags(((DWIDTH-1)/4) downto 0);  ——s_cyo 和 s_ovo 输出到
                s_ovo <= v_flags(v_flags'HIGH);          ——mc8051_alu 的 cy_i 和 ov_i
                wait for PROP_DELAY;                      ——等待 100ns
                if i /= 0 then
                v_quot := j/i;                            ——产生期待的商
                v_remd := j rem i;                        ——产生期待的余数
                assert (s_cyi((DWIDTH-1)/4) = '0')        ——对结果进行比较
                    and (s_ovi = '0')
                    and (s_qutnt = conv_std_logic_vector(v_quot,DWIDTH))
                    and (s_rmndr = conv_std_logic_vector(v_remd,DWIDTH))
                    report "ERROR in division!"
                    severity failure;
                else                                      —— 否则除数为 0
                assert (s_cyi((DWIDTH-1)/4) = '0')
                    and (s_ovi = '1')
                    report "ERROR in division by zero - flags not correct!"
                    severity failure;
                end if;
            end loop;  -- f
            end loop;  -- i
        end loop;  -- j
        assert false                          —— 程序执行到这里表示没有问题产生
            report "********* " & IMAGE(DWIDTH) & "BIT DIVIDER SEQUENCE FINISHED AT "
            & IMAGE(now) & " !" & " *********"
            severity note;
        s_dvdr_end <= true;
        wait;
    end PROC_DIV_ACC_RAM;
    ------------------------------------------------------------------------------
    ——定义常数和信号
    constant PROP_DELAY : time := 100 ns;

    signal rom_data_DIV_ACC_RAM : std_logic_vector(7 downto 0);
    signal ram_data_DIV_ACC_RAM : std_logic_vector(7 downto 0);
    signal acc_DIV_ACC_RAM : std_logic_vector(7 downto 0);
    signal hlp_DIV_ACC_RAM : std_logic_vector(7 downto 0);
    signal cmd_DIV_ACC_RAM : std_logic_vector(5 downto 0);
    signal cy_DIV_ACC_RAM : std_logic_vector(1 downto 0);
    signal ov_DIV_ACC_RAM : std_logic;
    signal new_cy_DIV_ACC_RAM : std_logic_vector(1 downto 0);
    signal new_ov_DIV_ACC_RAM : std_logic;
    signal result_a_DIV_ACC_RAM : std_logic_vector(7 downto 0);
```

```
signal result_b_DIV_ACC_RAM : std_logic_vector(7 downto 0);
signal end_DIV_ACC_RAM : boolean;
-------------------------------------------------------------------------------


begin
-------------------------------------------------------------------------------
-- Test the DIV_ACC_RAM command                                              --
-------------------------------------------------------------------------------
    rom_data_DIV_ACC_RAM <= conv_std_logic_vector(0,8);
    cmd_DIV_ACC_RAM      <= conv_std_logic_vector(43,6);    ——43 表示除法命令

    UUT : mc8051_alu                                        ——例化元件
        port map (
        rom_data_i => rom_data_DIV_ACC_RAM(7 downto 0),
        ram_data_i => ram_data_DIV_ACC_RAM(7 downto 0),
        acc_i      => acc_DIV_ACC_RAM(7 downto 0),
        cmd_i      => cmd_DIV_ACC_RAM,
        cy_i       => cy_DIV_ACC_RAM(1 downto 0),
        ov_i       => ov_DIV_ACC_RAM,
        new_cy_o   => new_cy_DIV_ACC_RAM(1 downto 0),
        new_ov_o   => new_ov_DIV_ACC_RAM,
        result_a_o => result_a_DIV_ACC_RAM(7 downto 0),
        result_b_o => result_b_DIV_ACC_RAM(7 downto 0));
——mc8051_alu 的被除数、除数、cyi、ovi 由 PROC_DIV_ACC_RAM 产生
——注意到 PROC_DIV_ACC_RAM 的商(s_qutnt)、余数(s_rmndr)的 Port 方向是 in,
——用来连接 mc8051_alu 的输出 result_a_DIV_ACC_RAM 和 result_b_DIV_ACC_RAM
    PROC_DIV_ACC_RAM (DWIDTH => 8,                          ——调用过程
                PROP_DELAY      => PROP_DELAY,
                s_cyi           => new_cy_DIV_ACC_RAM(1 downto 0),
                s_ovi           => new_ov_DIV_ACC_RAM,
                s_qutnt         => result_a_DIV_ACC_RAM(7 downto 0), ——商
                s_rmndr         => result_b_DIV_ACC_RAM(7 downto 0), ——余数
                s_cyo           => cy_DIV_ACC_RAM(1 downto 0),
                s_ovo           => ov_DIV_ACC_RAM,
                s_dvdnd         => acc_DIV_ACC_RAM(7 downto 0),      ——被除数
                s_dvsor         => ram_data_DIV_ACC_RAM(7 downto 0), ——除数
                s_dvdr_end      => end_DIV_ACC_RAM);
-------------------------------------------------------------------------------
end TBX_ARCH_DIV;
```

```
--配置，表示 TBX_mc8051_alu 这个 entity 使用 TBX_ARCH_DIV 这个 architecture
configuration TBX_CFG_mc8051_alu_TBX_ARCH_DIV of TBX_mc8051_alu is
for TBX_ARCH_DIV
end for;
end TBX_CFG_mc8051_alu_TBX_ARCH_DIV;
```