这是 I²C 总线控制器为 Master 时的发送和接收流程，两者的主要差别是，接收时，μC 需要在接收将近结束时配置 I²C 总线控制器 MBCR 寄存器，以便 I²C 总线控制器在最后一个字节不发出响应比特，让 Slave 结束发送。Testbench 向 EEPROM 写入测试数据的代码如下：

```
//*********** write data to EEPROM ***************
task current_write_eeprom;
begin
`mbcr=8'b0;
men=1'b1;
mbb=1'b1;
txak=1'b0;  //ack
uc_write_i2c(I2C_MBCR_ADDR,`mbcr);  //enable I2C
while(mbb==1)   // =1, bus busy
      begin
        uc_read_i2c(read_out,I2C_MBSR_ADDR);
        `mbsr=read_out;
      end

uc_write_i2c(I2C_MBDR_ADDR,{EEPROM_Address,1'b0}); //write i2c header
in MBDR; 1'b0 means master write data

mtx=1;  //necessary
msta=1; //Set MSTA in MBCR to Generate START
uc_write_i2c(I2C_MBCR_ADDR,`mbcr);

@(posedge mcf_wire);

for(count=255;count>=1;count=count-1)
      begin
        @(posedge mcf_wire)
        uc_write_i2c(I2C_MBDR_ADDR,count);  //write data to MBDR
        $display("write data: %d",count);
    end

@(posedge mcf_wire);
msta=1'b0;//reset MSTA in MBCR to Generate stop
uc_write_i2c(I2C_MBCR_ADDR,`mbcr);
end
endtask
```

由于调用了 BFM 里的 task，因此代码结构清晰，阅读、维护都非常容易，整个 Testbench 代码可以参见请见本书附带光盘的“Example-10-1”目录。

需要注意的是，受到篇幅的限制，本 Testbench 的编写并不完善，覆盖率、结果自动检查等很多问题也未能涉及，笔者将会在以后出版的书籍中详细论述验证相关的问题，有兴趣的读者请关注人民邮电出版社的出版动态。

在复杂 FPGA/IC 项目中，HDL 代码设计只占工作量的 30%，60％～70％的时间会用在仿真验证工作上。在基于 IP 复用的设计中，由于用了比较多的 IP，设计的工作量大大减少，验证工作量却依然巨大。写好验证计划和 Testbench 是非常有挑战性的工作，也是项目成败的关键，必须给予足够的重视。

10.4.4.3 BFM 模块编写

这里向读者介绍 FPGA/IC 验证技术的一个重要概念——BFM 模块，BFM 模块可以理解为 Testbench 的物理层，可以把验证所需的操作，例如 μC 对 I²C 总线控制器的读、写操作转换为 μC 和 I²C 总线控制器之间的信号变化，从而驱动 I²C 总线控制器执行相应的操作。使用 BFM 构建 Testbench 的优点如下：

(1) Testbench 结构清晰，使用 BFM 后，Testbench 可以简单划分为激励生成和驱动两大部分，前者产生验证所需的数据和控制整个验证的流程，是抽象层次比较高的部分；后者也就是 BFM，和 DUT 直接相连，负责 DUT 的驱动和控制，是抽象层次比较低的部分，采用周期精确（cycle-accurate）的编写风格。

(2) 可维护性好，采用 BFM 后，如果 DUT 接口信号或时序发生变化，只需要修改 BFM，不用改动 Testbench 的其他部分。如果想改变送到 DUT 的数据或流程，则只需要修改激励生成部分。

I²C 接口验证环境中，BFM 主要包含 μC 读写 I²C 总线控制器的 task，Verilog 代码如下：

```
task uc_write_i2c;
input [23:0] addr;
input [7:0] data;
begin
@(negedge clk)
r_w=1'b0;
addr_bus=addr;
as=1'b0;
data_bus=data;
ds=1'b0;
data_bus_drive=1;
@(negedge dtack)
as=1'b1;
ds=1'b1;
data_bus=8'hzz;
```

```
r_w=1'b1;           //reset the r_w line
data_bus_drive=0;
#1000;          //2clk, wait for uc_interface to transit to idle state
end
endtask


task uc_read_i2c;
output [7:0] data;
input [23:0] addr;
begin
@(negedge clk)      //write i2c header in MBDR
r_w=1'b1;
addr_bus=addr;
as=1'b0;
ds=1'b0;            //wait for state transition from addr to data_trs
data_bus_drive=1'b0;
@(negedge dtack)
as=1'b1;
ds=1'b1;
data=data_bus_tri;
#1000 ;         //2clk, wait for uc_interface to transite to idle state
end
endtask
```

读者可以发现，以上代码的编写风格比较特别，这是周期精确的行为级描述，是编写 BFM 的最常用方式，如果不是和硬件交互，应尽量避免使用周期精确的行为级描述，因为它描述了对象在每个时钟周期的行为，因此比一般的行为级描述复杂，在多数场合是不必要的。

有了 BFM，读写 I²C 总线控制器中的寄存器时，只需要调用相应的 task，不用逐一驱动相关的引线，非常方便。例如写控制寄存器 MBCR：

uc_write_i2c(I2C_MBCR_ADDR,`mbcr);

读状态寄存器 MBSR：

uc_read_i2c(read_out,I2C_MBSR_ADDR);

如果 μC 和 I²C 总线控制器之间的接口发生变化，例如改变了引脚名称，那么只需要改变 BFM 里相关的代码，非常简便，大大降低了发生错误的机会。

10.4.4.4 仿真验证

I²C 总线控制器的仿真验证步骤包括功能仿真、综合后仿真和布局布线后仿真，仿真时使用同一个 Testbench。如果 Testbench 和 DUT 使用的是同一种 HDL 语言，可以直接把 Testbench

也加入到 ISE 中，通过 ISE 调用 ModelSim 完成各个仿真验证步骤。对于混合语言仿真，ISE 不允许在一个工程中存在两种 HDL 语言，因此只能单独使用 ModelSim 进行仿真。

做功能仿真时，在 ModelSim 中建立工程，把 I²C 的所有 VHDL 文件，以及 Testbench 用到的 Verilog 文件加入到工程中，然后调整编译顺序，保证首先编译 Verilog 代码，然后才编译 VHDL 代码，如图 10-30 所示。

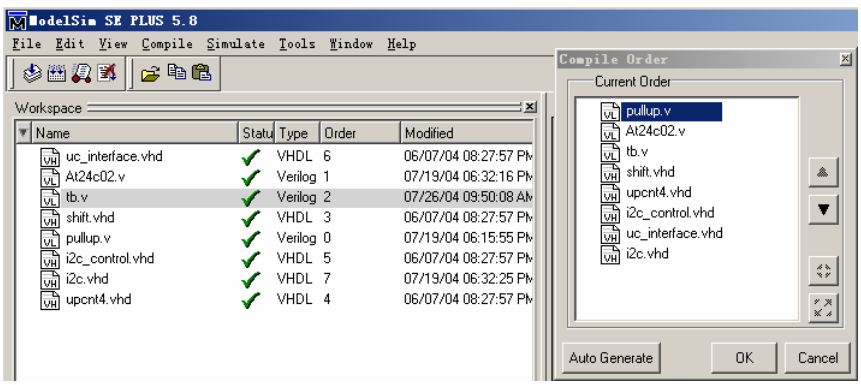

图10-30　功能仿真使用的文件及编译顺序

仿真波形如图 10-31 所示。

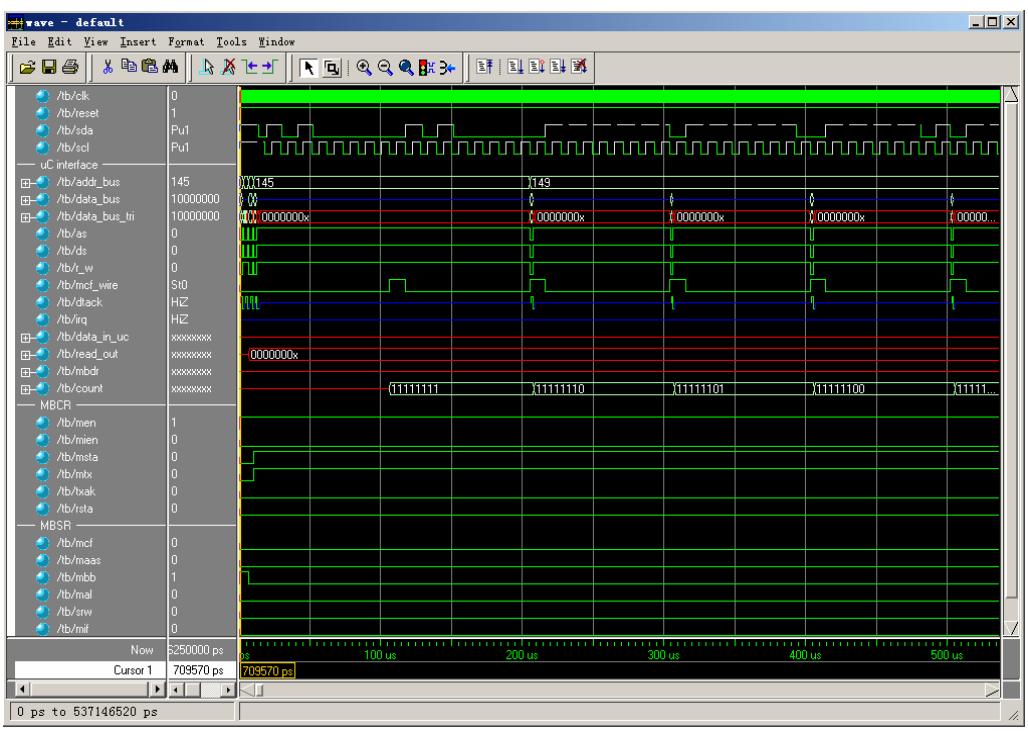

图10-31　功能仿真波形

在仿真调试的过程中，波形观察器是必不可少的工具，它能提供信号状态和变化的详细信息，但是依靠观察波形来判断 DUT 工作正常与否是非常繁重和容易出错的，在编写 Testbench 应尽量实现响应检查自动化。

响应检查自动化的实现方法主要有波形比较、数据文件比较和自检查 Testbench 等，其中波形比较使用仿真器的波形比较功能比较 DUT 仿真的波形和参考波形是否一致，从而判断 DUT 功能是否正确。在功能仿真过程中，假设某个版本的 RTL 代码通过了验证，得到了正确的仿真波形，可以把这个波形保存起来，如果代码需要修改，那么可以把修改后的 RTL 仿真结果和该波形进行比较，波形观察器会自动显示两者之间的差异，设计者可以从中判断修改后 DUT 功能是否正确。把综合/布局布线后的仿真波形和 RTL 代码的仿真波形做比较，可以判断综合/布局布线后 DUT 的功能是否发生了改变。

数据文件比较的方法比较适合以数据处理为主的设计，例如图像处理器、编码/译码器、数字信号处理器等，方法是首先使用其他工具、语言产生参考激励和参考响应，分别保存到数据文件中，然后使用参考激励驱动 DUT，Testbench 在仿真过程中比较 DUT 的响应和参考响应是否一致，或者把处理结果也存到数据文件中，然后使用文件比较命令比较 DUT 输出的数据文件和参考响应数据文件是否一致，以判断 DUT 的功能是否正确。

总的来说，波形比较和数据文件比较需要较多的人为干预，自动化程度不够高，在大多数情况下，使用自检查 Testbench 是更好的选择。仿真时，自检查 Testbench 根据实际的激励产生参考响应，同时比较 DUT 响应和参考响应是否相同，然后给出提示信息，以便设计者判断。典型的自检查 Testbench 的结构如图 10-32 所示。

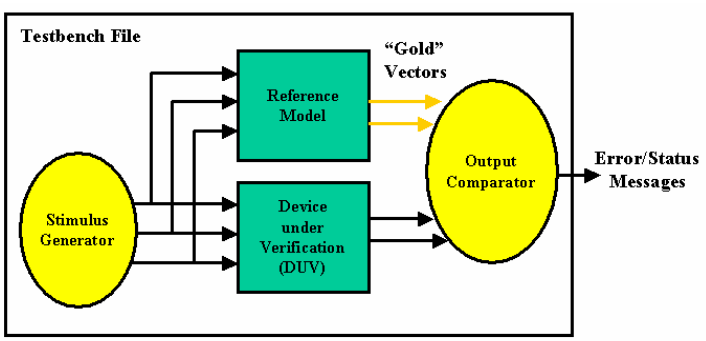

图10-32　自检查 Testbench 的结构

图中“Reference Model”是参考模型，其行为和 DUT 相同，一般使用行为描述实现，并经过一定的验证（或者自身足够简单，正确性容易得到保证），可以为结果比较模块提供参考输出（即 Gold Vectors）。有了自检查 Testbench 后，仿真工作会变得非常轻松，设计者只需要观察提示的信息就可以判断 DUT 是否正常，不再需要分析繁琐的波形了。

我们为 I²C 模块的验证编写了自检查 Testbench，仿真时 Modelsim 的提示信息如图 10-33 所示，从中可以清楚地看出整个验证过程，可以很方便地判断 I²C 接口控制器工作是否正确。

为了做综合后仿真以及布局布线后仿真，需要生成相应的仿真模型，方法是在 ISE 中建立工程，把 RTL 代码加入到工程中，根据使用的仿真器设置“Simulation Model Target”，如图 10-34 所示。然后双击“Generate Post-Map Simulation Model”，ISE 生成的 i2c_map.vhd 就是综合后仿真使用的 VHDL 文件。点击“Generate Post-Place & Route Simulation Model”生成的 i2c_timesim.vhd 和 i2c_timesim.sdf 为后仿真使用的仿真模型和反标文件。在

ModelSim 中，分别用这些文件代替功能仿真所用的 VHDL 代码就可以进行综合后仿真和布局布线后仿真了。

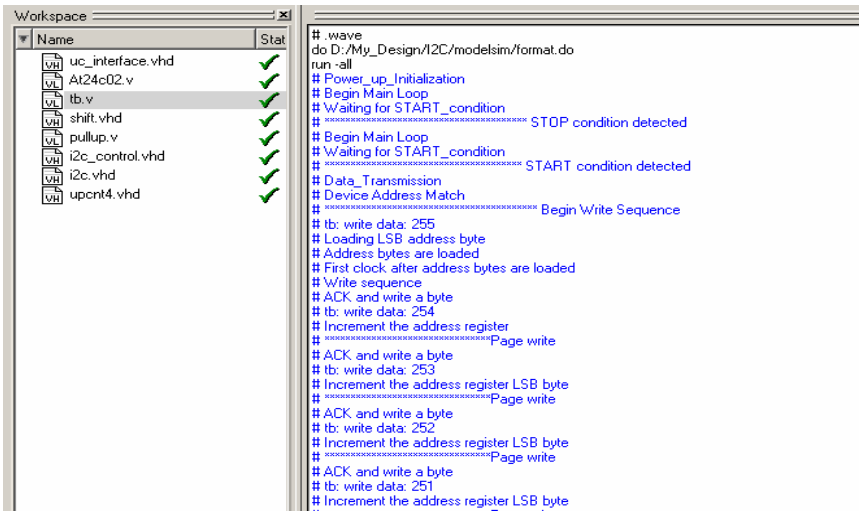

图10-33　功能仿真结果

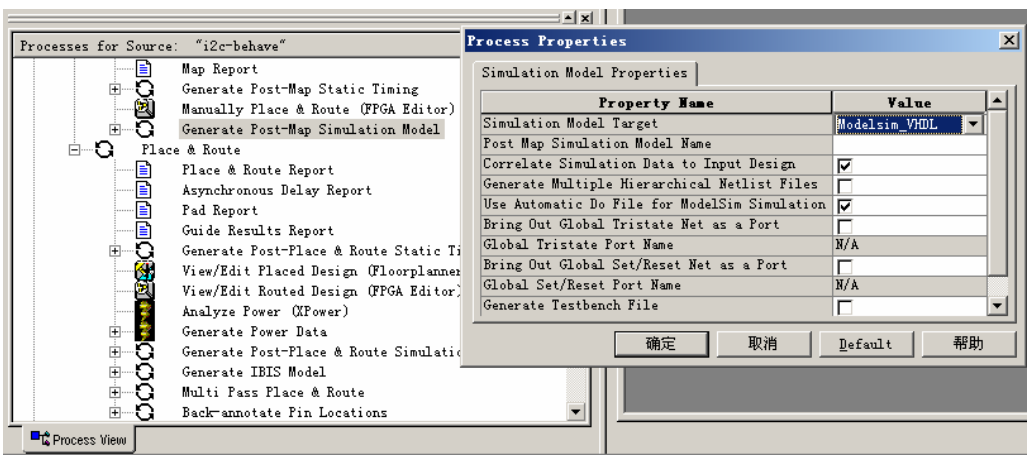

图10-34　产生 Post-map 仿真模型

如果使用 Synplify pro 做综合，在“Implementation Results”中选中“Write Mapped VHDL Netlist”后，会生成*.vhm 文件，该文件就是综合后的仿真模型，把后缀改为.vhd 后，就可以代替 RTL 代码在 ModelSim 中做综合后仿真了。

仿真所用到的文件、工程和仿真库请参考本书附带光盘的“Example-10-1”目录，读者可以把整个目录拷贝到自己的计算机上运行。后仿真要用到 SIMPRIM 和 VITAL2000 库，请读者检查当前工程是否已经引用了这两个库。

由于后仿真速度很慢，特别是设计比较大的时候，因此设计者要根据实际情况决定是否需要做后仿真。后仿真的目的是验证实现后的网表以及时序的正确性，一般而言，实现后的网表出问题往往是 RTL 代码风格不严谨所致，只要有好的代码设计风格，工具能正确产生网表。而时序的正确性可以通过静态时序分析解决，效率要比后仿真高得多。

10.4.5　上板调试

I²C 总线控制器仿真通过之后，就可以进行 FPGA 验证，即上板调试了。也许读者会问，后仿真通过了，还需要上板调试吗？答案是肯定的，主要原因是编写 Testbench 覆盖所有功能点有时候非常困难，可能也需要很多仿真时间。这也是在复杂设计中，验证工作要消耗 60%～70%时间的原因，而且有时候 Testbench 并没有真实地模拟设计的工作环境，在这种情况下很难保证验证结果的正确性，因此做上板调试一般也是必要的。

I²C 总线控制器上板调试之前，首先要综合、布局布线和生成 BIT 文件，综合工具建议使用 Synplify Pro（创建 I²C 总线控制器工程时进行选择），功能强大、使用方便，可以用于多个 FPGA 厂家的产品，由于工作频率不高，不用加约束。

EFX-SP200 实验开发系统板上有 80C51 单片机，可以和 I²C 微处理器接口相连，以便对 I²C 总线控制器进行控制，如果读者的板子上没有微处理器和 FPGA 相连，那么使用起来就会非常不方便。用于调试 I²C 总线控制器的单片机程序并不复杂，可以说是把 Testbench 转换为 80C51 的 C 语言或汇编语言代码而已，详细的代码请到专业 EDA 论坛（http://www.edacn.net）下载。

如果读者没有接触过单片机，那应该找个机会学习一下。单片机非常有用，可以设计很多产品，可以为以后学习 DSP（如 TI 的 6416）、Microprocessor（如 Motorola 的 8260）打下坚实的基础，而 80C51 单片机是最常用的。市场上有非常多的教材可以参考，掌握单片机知识后，就可以用 EFX-SP200 实验开发系统设计包含硬件（FPGA，做高速处理）和软件（80C51 控制器，做较低速处理及控制）的复杂系统，逐步迈入 SOC 设计的殿堂。

10.5　小结

FPGA/IC 设计一般包含协议学习、结构定义、代码编写、仿真、调试等过程，一般的初学者由于缺乏实践机会，往往只关注 HDL 代码的编写，而轻视协议学习和深入细致的仿真调试，因而不能全面的掌握 FPGA/IC 设计技能。本章以 I²C 总线控制器的设计为例，详细介绍了 I²C 总线协议、I²C 控制器的编码、仿真和调试。通过本章的学习，读者可以比较清楚地了解 FPGA/IC 的设计流程，掌握微控制器接口设计、状态机设计、混合语言仿真、后仿真、EDIF 设计流程等技术，完成一个有工程实践意义的设计，成为一个真正的 FPGA/ASIC 设计工程师。

10.6　问题与思考

1. I²C 总线与其他并行、串行总线相比，有什么优点和不足？
2. I²C 总线控制器的作用是什么？
3. 用状态机完成设计有什么优缺点？什么样的设计适合用状态机完成？
4. 为什么要使用混合仿真环境？实现混合仿真验证的关键步骤是什么？
5. 使用 BFM 构建 Testbench 有什么优点？