Tasks, Functions, and Testbench

介绍

Verilog 允许您使用 tasks 和 functions 定义子程序。 它们用于提高可读性并开发可重用性代码。 functions 等同于组合逻辑，不能用于替换包含事件或延迟控制运算符的代码（如在顺序逻辑中使用的那样）。 tasks 比 functions 更通用，并且可能包含时序控件。 Testbench 是一种用任意语言编写的程序或模块，用于在模拟过程中执行和验证硬件模型的功能正确性。 Verilog 主要用于硬件建模（模拟），该语言包含各种资源，用于格式化，读取，存储，动态分配，比较和写入模拟数据，包括输入激励和输出结果。

在本实验中，您将学习如何编写 tasks，functions 和 testbenchs。 您将了解 testbench 的组件以及可用于验证底层硬件模型正确性的语言构造。 请参阅 Vivado 教程，了解如何使用 Vivado 工具创建项目和验证数字电路。

目标

完成本实验后，您将能够：

•开发用于对组合电路建模的 tasks

•开发用于对组合电路建模的 functions

•开发 testbenchs 以测试和验证测试中的设计

Tasks

Task 就像一个过程，它提供了从模型中的几个不同位置执行常见代码的能力。 Task 可以包含时序控件，它可以调用其他 tasks 和 functions（在下一部分中描述）。 在模块定义中，Task 被定义为：

```
task task_id;
[declarations]
procedure statements
endtask
```

Task 可以包含零个，一个或多个参数。 值通过参数传递给任务和从任务传递。 参数可以是输入，输出或输出。 以下是 task 定义和用法的示例。

```
module HAS_TASK;
    parameter MAXBITS = 8;
    task REVERSE_BITS; // task definition starts here
        input [MAXBITS – 1 : 0] DIN;
        output [MAXBITS – 1 : 0] DOUT;
        integer k;
```

```
        begin
            for (k=0; k < MAXBITS; k = k +1)
                DOUT[MAXBITS-k] = DIN[k];
        end
    endtask // task definition ends here
    reg [MAXBITS – 1] REG_X, NEW_REG;
    always @ REG_X)
        REVERSE_BITS(REG_X,NEX_REG); // task being called
endmodule
```

Verilog HDL 也提供少量 system tasks。system task 名称前面带有$。例如，$ display – 将指定信息打印到带有行尾字符的标准输出。 例如 $ display（“At Simulation time %t，x_var is %d”）；将以十进制格式打印 x_var 值，以当前时间格式打印 time。 $ write-与 display task 类似，但它不打印行尾字符。 $ monitor – 持续监视参数。 只要参数列表中的值发生更改，就会显示整个参数列表。

例如

```
initial
$monitor(“At %t, D= %d, CLK = %d”, $time, D, CLK, “and Q is %b”, Q);
```

请注意，当调用$display task 时，其中的参数列表值会被打印，而在$monitor 中，只要其中一个参数的值发生更改，它就会被打印。

System tasks 不可综合，即它们不能在真实硬件中实现。

1-1。 编写一个名为 add_two_values 的 task，它将获取两个 4 位参数，将它们相加，并输出一个 4 位和和一个进位。 编写一个名为 add_two_values_task 的模块，该模块使用通过输入端口接收的操作数调用 task 并输出结果。 使用提供的 testbench，add_two_values_tb.v 对设计进行仿真，并验证功能。
1-1-1。 打开 Vivado 并创建一个名为 lab4_1_1 的空白项目。
1-1-2。 创建并添加名为 add_two_values_task 的 Verilog 模块，该模块定义了一个名为 add_two_values 的 task。 该任务将采用两个 4 位参数，将它们相加，并输出一个 4 位和和一个进位。 模块将使用通过输入端口接收的操作数调用任务并输出结果。
1-1-3。 使用提供的 testbench， add_two_values_task_tb.v 仿真设计，并验证设计是否有效。 在仿真器控制台窗口中查找$display 显示的消息。
1-1-4。 根据需要调整仿真时间。
1-2。 编写一个名为 calc_even_parity 的 task，该任务输入为一个 8 位数字，并计算并返回奇偶性。 编写一个名为 calc_even_parity_task 的模块，该模块使用通过输入端口接收的操作数调用 task 并输出结果。使用提供的 testbench calc_even_parity_task_tb.v，用$display system task 显示结果。 仿真设计并验证功能。

Functions

Functions 在父模块中使用关键字 function 和 endfunction 声明。 如果满足以下所有条

件，则使用 functions：
•没有存在的延迟，计时或事件控制结构
•它返回单个值
•至少有一个输入参数
•没有输出或 inout 参数
•没有非阻塞任务

简而言之，functions 可以仅实现组合行为，即它们基于输入参数的当前值计算值并返回单个值。 它们用在赋值语句的右侧。 这是 functions 定义和调用的示例。

```
module HAS_FUNCTION(X_IN, REV_X);
    parameter MAXBITS = 8;
        output reg [MAXBITS – 1 : 0] REV_X;
        input [MAXBITS – 1 : 0] X_IN;
    function [MAXBITS – 1 : 0] REVERSE_BITS; // function definition starts
                                             //here
        input [MAXBITS – 1 : 0] DIN;
        integer k;
        begin
            for (k=0; k < MAXBITS; k = k +1)
                REVERSE_BITS[MAXBITS-k] = DIN[k];
        end
    endfunction // function definition ends here
    always @ (X_IN)
        REV_X = REVERSE_BITS(X_IN); // function being called
endmodule
```

2-1。 编写一个名为 add_two_values 的 function，它将获取两个 4 位参数，将它们相加，并返回一个 5 位和。 编写一个名为 add_two_values_function 的模块，带有两个 4 位输入端口和一个 5 位输出端口，并调用该 function。 使用提供的 testbench add_two_values_function_tb.v 仿真设计，并验证功能。
2-1-1。 打开 Vivado 并创建一个名为 lab4_2_1 的空白项目。
2-1-2。 创建并添加一个名为 add_two_values_function 的 Verilog 模块，该模块定义了一个 function，称为 add_two_values，它接受两个 4 位参数，将它们相加，并返回一个 5 位和。 该模块将具有两个 4 位输入端口和一个 5 位输出端口。 它将调用该 function。
2-1-3。 使用提供的 add_two_values_function_tb.v 仿真 50 ns 以验证设计是否有效
2-2。 编写一个名为 calc_ones 的 task，它将采用一个 8 位数字，计算并返回一的个数。 编写一个名为 calc_ones_function 的模块，带有一个 8 位输入端口和一个 3 位输出端口，并调用该 function。 使用提供的 calc_ones_function_tb.v 模拟设计，并验证功能。

Testbench

Testbench 的主要组件是：

o`时间表声明
指定所有延迟的时间单位
o Module，它定义了测试平台的顶级结构
测试平台通常没有端口
o 内部信号，它将驱动激励信号进入 UUT 并监控 UUT 的响应
信号驱动和监控
o UUT 实例化
o 激励生成
编写语句以创建激励和程序块
o 响应监控和比较
自我测试语句，能报告数值，错误和警告
$ display，$ write，$ strobe 和/或$ monitor 系统任务

以下是我们在 Vivado Tutorial lab 中使用的 testbench 示例。

```
1 `timescale 1ns / 1ps
2 module tutorial_tb(
3     );
4 
5     reg [7:0] switches;
6     wire [7:0] leds;
7     reg [7:0] e_led;
8 
9     integer i;
10 
11    tutorial tut1(.led(leds),.swt(switches));
12 
13    function [7:0] expected_led;
14       input [7:0] swt;
15    begin
16       expected_led[0] = ~swt[0];
17       expected_led[1] = swt[1] & ~swt[2];
18       expected_led[3] = swt[2] & swt[3];
19       expected_led[2] = expected_led[1] | expected_led[3];
20       expected_led[7:4] = swt[7:4];
21    end
22    endfunction
23 
24    initial
25    begin
26        for (i=0; i < 255; i=i+2)
27        begin
28            #50 switches=i;
29            #10 e_led = expected_led(switches);
30            if(leds == e_led)
31                $display("LED output matched at", $time);
32            else
33                $display("LED output mis-matched at ",$time,": expected: %b, actual: %b", e_led, leds);
34        end
35    end
36 endmodule
```

第 1 行定义了`timescale 指令。第 2 行和第 3 行定义了测试平台模块名称。请注意，通常，testbench 模块的端口列表中不列出端口。第 5 行将开关定义为 reg 数据类型，因为它将用于提供激励。它连接到被测试（tutorial）的实例化设备。第 11 行使用实例名称 tut1 和输入/输出端口实例化测试（tutorial）中的设计。第 13 行到第 22 行定义了计算预期输出的 function。第 24 至 35 行使用 initial 过程描述来定义激励。您将在仿真器控制台窗口中使用 system task $ display 来查看第 31 行和第 33 行生成的消息。第 29 行通过传递开关参数调用函数 expected_led，并将返回的（计算过的）输出分配给 e_led。 e_led 在第 7 行

定义为 reg 类型，因为它在过程语句（初始）中接收函数调用的输出。第 28 和 29 行还分别定义惯性延迟 50 和 10，以模拟延迟。

Verilog 支持两种类型的延迟建模：（i）惯性和（ii）传输。 惯性延迟是门（gate）或电路由于其物理特性而可能经历的延迟。 根据所使用的技术，它可以是 ps 或 ns。 惯性延迟还用于确定输入是否对门或电路有影响。 如果输入至少在初始延迟时没有保持变化，则忽略输入变化。 例如，5 ns 的惯性延迟意味着无论何时输入发生变化，它都应保持至少 5 ns 的变化，以使其被视为已更改，否则将忽略该变化（被视为噪声尖峰）。 传输延迟是传输电路导线的信号的飞行时间。 以下是运输和惯性延迟的一些示例：

```
wire #2 a_long_wire; // transport delay of 2 units is modeled
xor #1 M1(sum, a, b); // inertial delay of 1 unit exerted by xor gate
```

初始语句在 testbenchs 中用于生成激励和控制仿真执行。
这是一个例子：

```
initial begin
    #100 $finish; // run simulation for 100 units
end
initial begin
    #10 a=0; b=0; // a, b zero after 10 units delay. Between 0 and 10,
                      //it is x
    #10 b=1; // At 20, make b=1
    #10 a=1; // at 30, make a=1
    #10 b=0; // at 40, make b=0
end
```

下面是生成称为时钟的周期信号的初始语句用法的另一个示例。 它将产生 50％占空比的时钟信号，周期为 20 个单位。

```
reg clock;
parameter half_cycle = 10;
initial
    begin
        clock = 0;
        forever
            begin
                #half_cycle clock = 1;
                #half_cycle clock = 0;
            end
        end
```

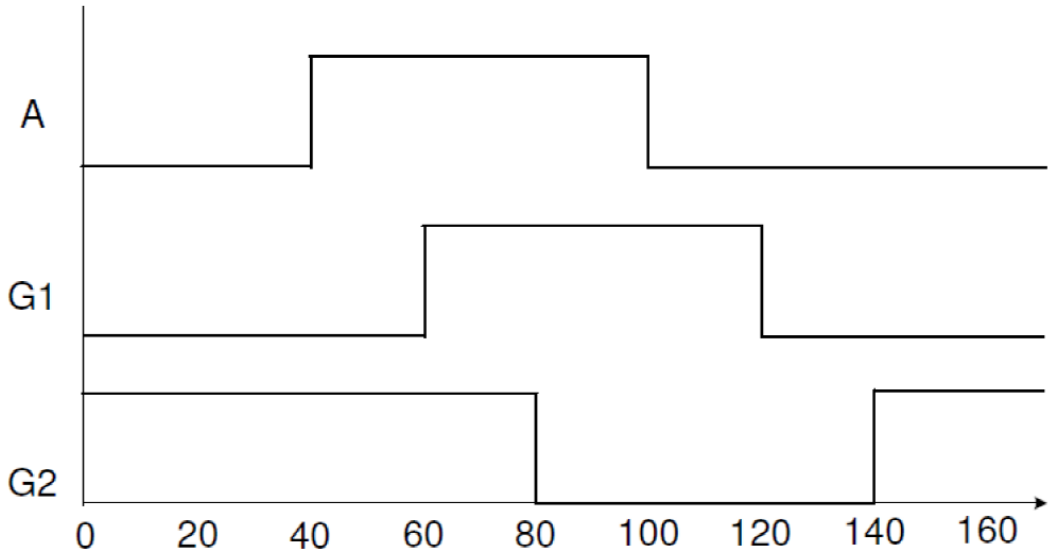
3-1。 开发 testbench 以验证 4 位纹波进位加法器（ ripple carry adder ）功能。 使用您在实验 2 的第 3-1 部分中开发的纹波进位设计(及其相关的 fulladder_dataflow 模块)。修改设计，为设计中的每个赋值语句包含 2 个惯性延迟单位。 开发一个 testbench，使其计算预期输出，比较结果，并将结果显示为“Test Passed”或“Test Failed”

3-1-1。 打开 Vivado 并创建一个名为 lab4_3_1 的空白项目。

3-1-2。 添加您在实验 2 的第 3-1 部分中使用的 Verilog 模块。修改设计，为设计中的每个赋值语句包含 2 个惯性延迟单位。 开发一个 testbench，使其计算预期输出，比较结果，并将结果显示为“Test Passed”或“Test Failed”。

3-1-3。 模拟设计所需的时间并验证设计是否有效。 您应该只需通过查看模拟器的控制台窗口来验证。

3-2。 开发一个 testbench，生成如下所示的波形。



3-2-1。 打开 Vivado 并创建一个名为 lab4_3_2 的空白项目。

3-2-2。 创建并添加输出上面显示的波形的 Verilog 模块。

3-2-3。 将设计模拟 150 ns 并验证是否生成了正确的输出。

结语

在本实验中，您学习了如何编写 function，task 和 testbench。 您还了解了 function 和 task 之间在定义和使用方面的差异，以及如何在 testbench 中使用 fuction 来计算预期输出。