NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

LECTURE

Subject:

VERILOG

Hardware Design Language

Chapter8: Tasks and Functions

Lecturer: Lam Duc Khai

Agenda

1. Chapter 1: Introduction (Week1)
2. Chapter 2: Fundamental concepts (Week1)
3. Chapter 3: Modules and hierarchical structure (Week2)
4. Chapter 4: Primitive Gates – Switches – User defined primitives (Week2)
5. Chapter 5: Structural model (Week3)
6. Chapter 6: Behavioral model – Combination circuit & Sequential circuit (Week4 & Week5)
7. Chapter 7: State machines (Week6)
8. Chapter 8: Tasks and Functions (Week6)
9. Chaper 9: Testbench and verification (Week7)

Tasks and Functions in Verilog

- Module cannot be instantiated inside a behavioral codes.
- A designer is frequently required to implement the same functionality at many places in a behavioral design.
- Verilog provides task and functions to break up large behavioral designs into smaller pieces.

Tasks versus Functions in Verilog

- Procedures/Subroutines/Functions in SW programming languages
  - The same functionality, but put in many different places
- Verilog equivalence:
  - *Tasks* and *Functions*
  - function and task (~ function and subroutine)
  - Used in behavioral modeling
  - Part of design hierarchy ⇒ Hierarchical name

Differences between...

- Functions
  - Can enable (call) just another function (not task)
  - Execute in 0 simulation time
  - No timing control statements allowed (next slide)
  - At lease one input, none output
  - Return only a single value
  - Synthesizable
- Tasks
  - Can enable (call) other tasks and functions
  - May execute in non-zero simulation time
  - May contain any timing control statements
  - May have arbitrary (none, one or many) inputs, outputs, or inouts
  - Do not return any value
  - Not synthesizable

Timing Control

- **Delay #**

  Used to delay statement by specified amount of simulation time

```
always
begin
      #10 clk = 1;
      #10 clk = 0;
end
```

- **Event Control @**

  Delay execution until event occurs

  Event may be single signal/expression change

  Multiple events linked by or

```
always @(posedge clk)
begin
      q <= d;
end
```

```
always @(x or y)
begin
      s = x ^ y;
      c = x & y;;
end
```

Similarity

- Both
  - are defined in a *module*
  - are local to the *module*
  - can have local variables (registers, but not nets) and events
  - contain only procedural statements (NOT continuous statements)
  - do not contain *initial* or *always* statements, they are only blocked by “begin …end”.
  - are called from *initial* or *always* statements or others task and functions

Task and function usage

- Tasks can be used for common Verilog code
  - is used for both combinational and sequential logic
- Function are used when the common code
  - is used for purely combinational logic
- Functions are typically used for conversions and commonly used calculations

Tasks

- Keywords: *task*, *endtask*
- Must be used if the procedure has
  - any timing control constructs
  - zero or more than one output arguments

Tasks (cont'd)

- Task declaration and invocation
  - Declaration syntax

```
task <task_name>;
        <I/O declarations>
        <variable and event declarations>
        begin     // if more than one statement needed
          <statement(s)>
        end       // if begin used!
endtask
```

Tasks (cont'd)

- Task declaration and invocation
  - Task invocation syntax (call)

    <task_name>;

    <task_name> (<arguments>);
  - *input* and *inout* arguments are passed into the task
  - *output* and *inout* arguments are passed back to the invoking statement when task is completed

Tasks (cont'd)

- I/O declaration in modules vs. tasks
  - Both used keywords: *input*, *output*, *inout*
  - In modules, represent ports
    - connect to external signals
  - In tasks, represent arguments
    - pass values to and from the task

Task Examples: Use of input and output arguments

```
module operation;
parameter delay = 10;
reg [15:0] A, B;
reg [15:0] AB_AND, AB_OR, AB_XOR;

initial
    $monitor( …);

initial
begin
    …
end

always @(A or B)
begin
    bitwise_oper(AB_AND, AB_OR,
        AB_XOR, A, B);
end
```

```
task bitwise_oper;
output [15:0]  ab_and, ab_or,
        ab_xor;
input [15:0] a, b;
begin
    #delay ab_and = a & b;
    ab_or = a | b;
    ab_xor = a ^ b;
end
endtask


endmodule
```

Task Examples : Use of module local variables

```
module sequence;
reg clock;

initial
begin
    …
end

initial
    init_sequence;

always
    asymmetric_sequence;
```

```
task init_sequence;
begin
    clock = 1'b0;
end
endtask

task asymmetric_sequence;
begin
    #12 clock = 1'b0;
    #5 clock = 1'b1;
    #3 clock = 1'b0;
    #10 clock = 1'b1;
end
endtask

endmodule
```

Functions

- Keyword: *function*, *endfunction*
- Can be used if the procedure
  - does not have any timing control constructs
  - returns exactly a single value
  - does not have any output
  - has at least one input argument

Functions (cont'd)

- Function Declaration and Invocation
  - Declaration syntax:

```
function <range_or_type> <func_name>;
    <input declaration(s)>
    <variable_declaration(s)>
    begin // if more than one statement needed
      <statements>
    end     // if begin used
endfunction
```

Functions (cont'd)

- Function Declaration and Invocation
  - Invocation syntax (call):

    <func_name> (<argument(s)>);

Functions (cont'd)

- Semantics
  - much like *function* in *Pascal*
  - An internal implicit *reg* is declared inside the function with the same name
  - The return value is specified by setting that implicit *reg*
  - <range_or_type> defines width and type of the implicit *reg*
    - *type* can be *integer* or *real*
    - default bit width is 1

Function Examples: Parity Generator

```
module parity;
reg [31:0] addr;
reg parity;

Initial begin
    …
end

always @(addr)
begin
  parity = calc_parity(addr);
  $display("Parity calculated = %b",
           calc_parity(addr) );
end
```

reg 1 bit

```
function calc_parity;
input [31:0] address;
begin
    calc_parity = ^address;
end
endfunction

endmodule
```

Same name

The implicit reg
1 bit width (default)

Function Examples: Controllable Shifter

```
module shifter;
`define LEFT_SHIFT      1'b0
`define RIGHT_SHIFT     1'b1
reg [31:0] addr, left_addr, right_addr;
reg control;

initial
begin
      …
end

always @(addr)begin
  left_addr  =shift(addr, `LEFT_SHIFT);
  right_addr =shift(addr,`RIGHT_SHIFT);
end
```

```
function [31:0] shift;
input [31:0] address;
input control;
begin
  shift = (control==`LEFT_SHIFT)
      ?(address<<1) : (address>>1);
end
endfunction

endmodule
```

The implicit reg
32 bit width

Tasks and Functions Summary

- Tasks and functions in behavioral modeling
  - The same purpose as subroutines in SW
  - Provide more readability, easier code management
  - Are part of design hierarchy
  - Tasks are more general than functions
    - Can represent almost any common Verilog code
  - Functions can only model purely combinational calculations