Chapter 5: Tasks, Functions, and UDPs

Prof. Ming-Bo Lin

Department of Electronic Engineering

National Taiwan University of Science and Technology

Syllabus

- Objectives
- Tasks
- Functions
- Combinational UDPs
- Sequential UDPs

Objectives

After completing this chapter, you will be able to:

- Describe the features of tasks and functions
- Describe how to use tasks and functions
- Describe the features of dynamic tasks and functions
- Describe the features of UDPs (user-defined primitives)
- Describe how to use combinational and sequential UDPs

Syllabus

- Objectives
- Tasks
  - Definition and call
  - Types of tasks
- Functions
- Combinational UDPs
- Sequential UDPs

Task Definition and Calls

task [automatic] task_identifier(task_port_list); … endtask

```
// port list style
task [automatic] task_identifier;
[declarations] // include arguments
procedural_statement
endtask

// port list declaration style
task [automatic] task_identifier ([argument_declarations]);
[other_declarations]  // exclude arguments
procedural_statement
endtask
```

A Task Example

```
// count the zeros in a byte
module zero_count_task (data, out);
input   [7:0] data;
output reg [3:0] out;
always @(data)
    count_0s_in_byte(data, out);
// task declaration from here
task count_0s_in_byte(input [7:0] data, output reg [3:0] count);
integer i;
begin   // task body
   count = 0;
   for (i = 0; i <= 7; i = i + 1)
         if (data[i] == 0) count= count + 1;
end endtask
endmodule
```

Syllabus

- Objectives
- Tasks
  - Definition and call
  - Types of tasks
- Functions
- Combinational UDPs
- Sequential UDPs

Types of Tasks

- (static) task

  task … endtask
- automatic (reentrant, dynamic) task

  task automatic … endtask

A Dynamic Task Example

```
// task definition starts from here
task automatic check_counter;
reg [3:0] count;
// the body of the task
begin
    $display ($realtime,,"At the beginning of task, count = %d", count);
    if (reset) begin
        count = 0;
        $display ($realtime,,"After reset, count = %d", count);
end
endmodule
```

Syllabus

- Objectives
- Tasks
- Functions
  - Definition and call
  - Types of functions
- Combinational UDPs
- Sequential UDPs

Function Definition and Calls

function [automatic] [signed] [range_of_type] … endfunction

```
// port list style
function [automatic] [signed] [range_or_type] function_identifier;
input_declaration
other_declarations
procedural_statement
endfunction

// port list declaration style
function [automatic] [signed] [range_or_type]
function_identifier (input_declarations);
other_declarations
procedural_statement
endfunction
```

A Function Example

```
// count the zeros in a byte
module zero_count_function (data, out);
input  [7:0] data;
output reg [3:0] out;
always @(data)
    out = count_0s_in_byte(data);
// function declaration from here.
function [3:0] count_0s_in_byte(input [7:0] data);
integer i;
begin
   count_0s_in_byte = 0;
   for (i = 0; i <= 7; i = i + 1)
       if  (data[i] == 0) count_0s_in_byte = count_0s_in_byte + 1;
end
endfunction
endmodule
```

Syllabus

- Objectives
- Tasks
- Functions
  - Definition and call
  - Types of functions
- Combinational UDPs
- Sequential UDPs

Types of Functions

- (static) function

  function … endfunction

- automatic (recursive, dynamic) function

  function automatic … endfunction

Automatic (Recursive) Functions

```
// the use of reentrant function
module factorial(input  [7:0] n, output [15:0] result);
// instantiate the fact function
    assign result = fact(7);
// define fact function
   function automatic [15:0]  fact;
   input [7:0] N;

   // the body of function
       if (N == 1) fact = 1;
       else fact = N * fact(N - 1);
   endfunction
endmodule
```

Constant Functions

```
module RAM (addr_bus, data_bus);
parameter RAM_depth = 1024;
input [count_log_b2(RAM_depth)-1:0] addr_bus;
output reg [7:0] data_bus;
// function declaration from here
    function integer count_log_b2(input integer depth);
        begin   // function body
            count_log_b2 = 0;
            while (depth) begin
                count_log_b2 = count_log_b2 + 1;
                depth = depth >> 1;
            end
        end
    endfunction
endmodule
```

Syllabus

- Objectives
- Tasks
- Functions
- Combinational UDPs
  - Definition
  - Instantiation
- Sequential UDPs

Definition of Combinational UDPs

```
// port list style
primitive udp_name(output_port, input_ports);
output output_port;
input input_ports;
// UDP state table
table // the state table starts from here
<table rows>
endtable
endprimitive
```

Definition of Combinational UDPs

- State table entries

  <input1><input2>……<input*n*>:<output>;

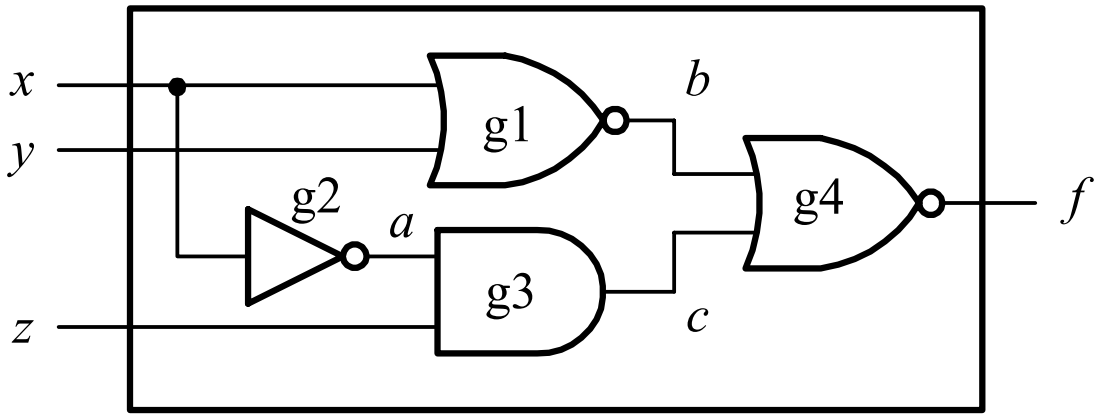
  - The <input#> values must be in the same order as in the input list

A Primitive UDP --- An AND Gate

```
primitive udp_and (out, a, b);
output out;
input a, b;
table
//    a   b   :   out;
      0   0   :   0;
      0   1   :   0;
      1   0   :   0;
      1   1   :   1;
endtable
endprimitive
```

Another UDP Example

```
primitive prog253 (output f, input x, y, z);
table   // truth table for f(x, y, z,) = ~(~(x | y) | ~x & z);
//   x y z : f
     0 0 0 : 0;
     0 0 1 : 0;
     0 1 0 : 1;
     0 1 1 : 0;
     1 0 0 : 1;
     1 0 1 : 1;
     1 1 0 : 1;
     1 1 1 : 1;
   endtable
endprimitive
```



Shorthand Notation for Don't Cares

```
primitive udp_and(f, a, b);
output f;
input a, b;
table
// a b : f;
   1 1 : 1;
   0 ? : 0;
   ? 0 : 0;   // ? expanded to 0, 1, x
endtable
endprimitive
```

Syllabus

- Objectives
- Tasks
- Functions
- Combinational UDPs
  - Definition
  - Instantiation
- Sequential UDPs

Instantiation of Combinational UDPs

```
// instantiations of UDPs
module UDP_full_adder(sum, cout, x, y, cin);
output sum, cout;
input x, y, cin;
wire s1, c1, c2;
// instantiate udp primitives
   udp_xor (s1, x, y);
   udp_and (c1, x, y);
   udp_xor (sum, s1, cin);
   udp_and (c2, s1, cin);
   udp_or (cout, c1, c2);
endmodule
```

Syllabus

- Objectives
- Tasks
- Functions
- Combinational UDPs
- Sequential UDPs
  - Definition
  - Instantiation

Definition of Sequential UDPs

```
// port list style
primitive udp_name(output_port, input_ports);
output output_port;
input input_ports;
reg output_port;      // unique for sequential UDP
initial output-port = expression; // optional for sequential
UDP
// UDP state table
table // keyword to start the state table
   <table rows>
endtable
endprimitive
```

Definition of Sequential UDPs

- State table entries

<input1><input2>……<input*n*>:<current_state>:<next_state>;

  - The output is always declared as a reg
  - An initial statement can be used to initialize output

Level-Sensitive Sequential UDPs

```
// define a level-sensitive latch using UDP
primitive d_latch(q, d, gate, clear);
output q;
input d, gate, clear;
reg q;
initial q = 0; // initialize output q
table
// d gate clear : q : q+;
   ?   ?    1   : ? : 0 ;   // clear
   1   1    0   : ? : 1 ;  // latch q = 1
   0   1    0   : ? : 0 ;  // latch q = 0
   ?   0    0   : ? : - ;   // no change
endtable
endprimitive
```

Edge-Sensitive Sequential UDPs

```
// define a positive-edge triggered T-type flip-flop using UDP
primitive T_FF(q, clk, clear);
output q;
input clk, clear;
reg q;
// define the behavior of edge-triggered T_FF
table
// clk clear : q : q+;
    ?      1   : ? : 0 ;  // asynchronous clear
    ?   (10) : ? : - ;   // ignore negative edge of clear
   (01)    0 : 1 : 0 ;  // toggle at positive edge
   (01)    0 : 0 : 1 ;  // of clk
   (1?)    0 : ? : - ;   // ignore negative edge of clk
endtable
endprimitive
```

Shorthand Symbols for Using in UDPs

Symbols	Meaning	Explanation
?	0, 1, x	Cannot be specified in an output field
b	0, 1	Cannot be specified in an output field
-	No change in state value	Can use only in a sequential UDP output field
r	(01)	Rising edge of a signal
f	(10)	Falling edge of a signal
p	(01), (0x), or (x1)	Potential rising edge of a signal
n	(10), (1x), or (x0)	Potential falling edge of a signal
*	(??)	Any value change in signal

Syllabus

- Objectives
- Tasks
- Functions
- Combinational UDPs
- Sequential UDPs
  - Definition
  - Instantiation

Instantiation of UDPs

```
// an example of sequential UDP instantiations
module ripple_counter(clock, clear, q);
input   clock, clear;
output [3:0] q;

// instantiate the T_FFs.
T_FF tff0(q[0], clock, clear);
T_FF tff1(q[1], q[0],  clear);
T_FF tff2(q[2], q[1],  clear);
T_FF tff3(q[3], q[2],  clear);
endmodule
```