Module #10 : Verilog HDL Functions

10.1: Verilog HDL Function :

- Functions are declared within a module, and can be called from continuous assignments, always blocks, or other functions.
- In a continuous assignment, they are evaluated when any of its declared inputs change.
- In a procedure, they are evaluated when invoked.
- Functions describe combinational logic, and do not generate latches.
- Functions are a good way to reuse procedural code, since modules cannot be invoked from within a procedure.

Module #10 : Verilog HDL Functions

10.2: Function Declaration :

- A function declaration specifies the name of the function, the width of the function return value, the function input arguments, the variables (reg) used within the function, and the function local parameters and integers

Syntax:

function *[msb:lsb] function_name;*
input [msb:lsb] input_arguments;
reg [msb:lsb] reg_variable_list;
parameter [msb:lsb] parameter_list;
integer [msb:lsb] integer_list;
... statements ...
endfunction

Example 10.1:

function *[7:0] my_func; // function return 8-bit value*
input [7:0] i;
reg [4:0] temp;
integer n;
temp= i[7:4] | (i[3:0]);
my_func = {temp, i[[2:0]};
endfunction

Module #10 : Verilog HDL Functions

10.3: Function Return Value:

- When we declare a function, a variable is also implicitly declared with the same name as the function name, and with the width specified for the function name (The default width is 1-bit).
- At least one statement in the function must assign the function return value to this variable.

10.4. Function Call:

- A function call is an operand in an expression. A function call must specify in its terminal list all the input parameters.

Example 10.2:

```
wire [7:0] x;
wire [7:0] y;
assign y = my_func[x];
```

Module #10 : Verilog HDL Functions

10.5: Function Return Value:

- Functions must contain at least one input argument.
- Functions cannot contain an inout or output declaration.
- Functions cannot contain time controlled statements (#, @, wait).
- Functions can invoke other functions, but not themselves (not recursive).
- Functions cannot invoke tasks.
- Functions must contain a statement that assigns the return value to the implicit function name register.

Module #10 : Verilog HDL Functions

10.6: Function Example:

- A Function has only one output. If more than one return value is required, the outputs should be concatenated into one vector before assigning it to the function name.
- The calling module program can then extract (unbundle) the individual outputs from the concatenated form.

Example 10.3:

```
module simple_processor (instruction, outp);
input [31:0] instruction;
output [7:0] outp;
reg [7:0] outp;; // so it can be assigned in always block
reg func;
reg [7:0] opr1, opr2;
```

Module #10 : Verilog HDL Functions

Continued . . .

```
function [16:0] decode_add (instr)
// returns 1 1-bit plus 2 8-bits
input [31:0] instr;
reg add_func;
reg [7:0] opcode, opr1, opr2;
begin
opcode = instr[31:24];
opr1 = instr[7:0];
case (opcode)
8'b10001000: begin // add two operands
add_func = 1;
opr2 = instr[15:8];
end
8'b10001001: begin // subtract two operands
add_func = 0;
opr2 = instr[15:8];
end
8'b10001010: begin // increment operand
add_func = 1;
opr2 = 8'b00000001;
end
default: begin; // decrement operand
add_func = 0;
opr2 = 8'b00000001;
end
endcase
decode_add = {add_func, opr2, opr1}; // concatenated
into 17-bits
end
endfunction
```

Module #10 : Verilog HDL Functions

Continued . . .

```
always @(instruction) begin
{func, op2, op1} = decode_add (instruction); //
outputs unbundled
if (func == 1)
outp = op1 + op2;
else
outp = op1 - op2;
end
endmodule
```