PARAMETERS

- Parameters are Verilog constructs that allow a module to be reused with a different specification
- For example, a 4-bit adder can be parameterized to accept a value for the number of bits and new parameter values can be passed in during module instantiation
- So, an N-bit adder can become a 4-bit, 8-bit or 16-bit adder
- They are like arguments to a function that are passed in during a function call
- Parameters are basically constants and hence it's illegal to modify their value at runtime
- It is illegal to redeclare a name that is already used by a net, variable or another parameter
- There are two major types of parameters, *module* and *specify* and both accepts a range specification
- But, they are normally made as wide as the value to be stored requires them to be and hence a range specification is not necessary

1. **parameter** MSB = 7; // MSB is a parameter with a constant value 7

2. **parameter REAL** = 4.5; // REAL holds a real number

3. **parameter** FIFO_DEPTH = 256, MAX_WIDTH = 32; // Declares two parameters

4. **parameter** [7:0] f_const = 2'b3; // 2 bit value is converted to 8 bits; 8'b3

- Module parameters can be used to override parameter definitions within a module and this makes the module have a different set of parameters at compile time
- A parameter can be modified with the defparam statement or in the module instance statement
- It is a common practice to use uppercase letters in names for the parameter to make them instantly noticeable
- The module shown below uses parameters to specify the bus width, data width and the depth of FIFO within the design, and can be overriden with new values when the module is instantiated or by using defparam statements

```
// Verilog 1995 style port declaration
module design_ip ( addr,
                   wdata,
                   write,
                   sel,
                   rdata);
parameter BUS_WIDTH = 32,
          DATA_WIDTH = 64,
          FIFO_DEPTH = 512;
input addr;
input wdata;
input write;
input sel;
output rdata;
wire [BUS_WIDTH-1:0] addr;
wire [DATA_WIDTH-1:0] wdata;
reg [DATA_WIDTH-1:0] rdata;
reg [7:0] fifo [FIFO_DEPTH];
// Design code goes here ...
endmodule
```

In the new ANSI style of Verilog port declaration, you may declare parameters as show below.

```
module design_ip
  #(parameter BUS_WIDTH=32,
    parameter DATA_WIDTH=64) (

    input [BUS_WIDTH-1:0] addr,
     // Other port declarations
   );
```

- Parameters can be overridden with new values during module instantiation.
- The first part instantiates the module called **design_ip** by the name **d0** where new parameters are passed in within **#()**
- The second part uses a Verilog construct called **defparam** to set the new parameter values
- The first method is the most commonly used way to pass new parameters in RTL designs
- The second method is commonly used in testbench simulations to quickly update the design parameters without having to reinstantiate the module

```
module tb;

    // Module instantiation override
    design_ip  #(BUS_WIDTH = 64, DATA_WIDTH = 128) d0 ( [port list]);

    // Use of defparam to override
    defparam d0.FIFO_DEPTH = 128;

endmodule
```

- The module counter has two parameters N and DOWN declared to have a default value of 2 and 0 respectively

- N controls the number of bits in the output effectively controlling the width of the counter. By default it is a 2-bit counter

- Parameter DOWN controls whether the counter should increment or decrement. By default, the counter will decrement because the parameter is set to 0

```
module counter
  #(   parameter N = 2,
       parameter DOWN = 0)

  ( input              clk,
    input              rstn,
    input              en,
     output   reg [N-1:0] out);

  always @ (posedge clk) begin
    if (!rstn) begin
      out <= 0;
    end else begin
      if (en)
        if (DOWN)
          out <= out - 1;
        else
            out <= out + 1;
      else
          out <= out;
    end
  end
endmodule
```

- The module counter is instantiated with N as 2 even though it is not required because the default value is anyway 2

- DOWN is not passed in during module instantiation and hence takes the default value of 0 making it an up-counter

```
module design_top (    input clk,
                       input rstn,
                       input en,
                       output [1:0] out);

   counter #(.N(2)) u0 (  .clk(clk),
                          .rstn(rstn),
                          .en(en));
endmodule
```

- See that default parameters are used to implement the counter where N equals two making it a 2-bit counter and DOWN equals zero making it an up-counter

- The output from counter is left unconnected at the top level

EXAMPLE-2: 4-BIT DOWN-COUNTER

- In this case, the module counter is instantiated with N as 4 making it a 4-bit counter
- DOWN is passed a value of 1 during module instantiation and hence an down-counter is implemented

```
module design_top (    input clk,
                       input rstn,
                       input en,
                       output [3:0] out);

   counter #(.N(4), .DOWN(1))
      u1 (  .clk(clk),
            .rstn(rstn),
            .en(en));
endmodule
```

- In Verilog HDL, parameters are constants and do not belong to any other data type such as net or register data types
- You are not allowed to modify parameter values at runtime, but you can modify a parameter value using the **defparam** statement
- The defparam statement can modify parameters only at the time of compilation
- Parameter values can also be modified using #delay specification with module instantiation
- In Verilog there are two ways to override a module parameter value during a module instantiation
- The first method is by using the ***defparam*** keyword and
- the second method is called *module instance parameter value assignment*

- A parameter is defined by Verilog as a constant value declared within the module structure. The value can be used to define a set of attributes for the module which can characterize its behavior as well as its physical representation

- Defined inside a module
- Local scope
- Maybe overridden at instantiation time
  - If multiple parameters are defined, they must be overridden in the order they were defined. If an overriding value is not specified, the default parameter declaration values are used.
- Maybe changed using the defparam statement

PARAMETER OVERRIDE USING DEFPARAM

```
1.  module secret_number;
2.  parameter my_secret = 0;
3.
    initial begin
4.    $display("My secret number is %d", my_secret);
5.  end
6.
7.  endmodule
8.
9.  module defparam_example();
10.
11. defparam U0.my_secret = 11;
12. defparam U1.my_secret = 22;
13.
14. secret_number U0();
15. secret_number U1();
16.
17. endmodule
```

PASSING MORE THAN ONE PARAMETER

```
1 module  ram_sp_sr_sw (
2 clk         , // Clock Input
3 address     , // Address Input
4 data        , // Data bi-directional
5 cs          , // Chip Select
6 we          , // Write Enable/Read Enable
7 oe            // Output Enable
8 );
10 parameter DATA_WIDTH = 8 ;
11 parameter ADDR_WIDTH = 8 ;
12 parameter RAM_DEPTH = 1 << ADDR_WIDTH;
13 // Actual code of RAM here
14
15 endmodule
```

```
1.  module secret_number;
2.  parameter my_secret = 0;
3.
    initial begin
4.    $display("My secret number in module is %d", my_secret);
5.  end
6.
    endmodule
7.
8.  module param_overide_instance_example();
9.
    secret_number #(11) U0();
10. secret_number #(22) U1();
11.
12. endmodule
```

- When instantiating more than the one parameter, parameter values should be passed in the order they are declared in the sub module

```
1 module  ram_controller ();//Some ports
2
3 // Controller Code
4
5 ram_sp_sr_sw #(16,8,256)  ram(clk,address,data,cs,we,oe);
6
7 endmodule
```

- In Verilog 2001, the code above will work, but the new feature makes the code more readable and error free

```
1 module  ram_controller ();//Some ports
2
3 ram_sp_sr_sw #(
4 .DATA_WIDTH(16),
5 .ADDR_WIDTH(8),
6 .RAM_DEPTH(256))  ram(clk,address,data,cs,we,oe);
7
8 endmodule
```