Reference Designer

UDP

User Defined Primitive

In the last page we saw how to create a single bit comparator using gate level modeling with predefined primitives. The use of the gates can becomes cumbursome if the number of gates are large. It also becomes hard to follow the code intuitively. Fortunately verilog also provide the concept of User Defined Primitives (UDPs). Using UDPs we define the function of a combinational logic using table.

Here is the 1 bit comparator example using the UDP

```
1.  `timescale 1ns / 1ps
2.  ////////////////////////////////////////////////////////////////////////////////
3.  // Example of comparator using UDP Table
4.  ////////////////////////////////////////////////////////////////////////////////
5.  module comparator(
6.          input x,
7.          input y,
8.          output z
9.  );
10. compare c0(z, x, y);
11. endmodule
12.
13. primitive compare(out, in1, in2);
14.         output out;
15.         input in1,in2;
16.
17. table
18. // in1  in2  : out
19.     0    0   : 1;
20.     0    1   : 0;
21.     1    0   : 0;
22.     1    1   : 1;
23. endtable
24. endprimitive
25.
```

This example does the same fuction as the previous example, but we have used primitive gates in this example. Notice how the verilog code gets simplified by the use of the udp tables/

Explanation

A new primitive is defined using the primitive keyword. It has an output and a list of inputs as its argument.

```
1. primitive compare(out, in1, in2);
2. output out;
3. input in1,in2;
```

The definition of the primitive is followed by a table definition. The Table definition starts with keyword table and ends with keyword endtable

```
1. table
2. // in1 in2 : out
3. 0 0 : 1;
4. 0 1 : 0;
5. 1 0 : 0;
6. 1 1 : 1;
7. endtable
```

Inside the table definition we define the primitive behavior with a number of rows. Each row has values of the inputs separated by whitespaces, followed by semicolon, followed by the output. The

input values should be in the same sequence as defined in the premitive definition.

Finally the prenitive definition ends with the keyword endpremitive.

We can instantiate the primitive with the primitive name followed by and identifier name and a list of the out and inputs as in

```
compare c0(z, x, y);
```

The stimulus stays the same and it produces the same result.

```
1.  `timescale 1ns / 1ps
2.
3.  /* Stimulus
4.  Example showing two bit comparator
5.  referencedesigner.com
6.  */
7.
8.  module stimulus1;
9.
10.         reg  x;
11.         reg  y;
12.     wire z;
13.
14.         // Instantiate the Unit Under Test (UUT)
15.         comparator uut (
16.                 .x(x),
17.                 .y(y),
18.                 .z(z)
19.         );
20.          initial begin
21.     // Initialize Inputs
22.     x = 0;
23.     y = 0;
24.
25.     // Wait 100 ns for global reset to finish
26.     #100;
27.     #50 x = 1;
28.     #60 y = 1;
29.     #70 y = 1;
30.     #80 x = 0;
31.     end
32.
33.
34. initial begin
35. $monitor("x=%d,y=%d,z=%d \n",x,y,z);
36. end
37.
38. endmodule
```

And it produces the same output

```
x=0,y=0,z=1
x=1,y=0,z=0
x=1,y=1,z=1
x=0,y=1,z=0
```

< Previous Next >