Introduction to Hardware Description Languages with Verilog

By Jake D. Karas

Introduction to Hardware Description Languages with Verilog

Jake D. Karas
U0000008780

10/14/2021

Objective:

This lab introduced the basics of the Verilog Hardware Description Language, a powerful tool used by digital logic designers to simulate a boolean function prior to purchasing any hardware components in order to reduce expenditures and improve the circuit's performance. In Modelsim, the software utilized to create and compile logic codes as well as simulate their outputs, three unique coding approaches were taken to simulate a particular logic circuit derived from a diagram. These approaches are the Gate Level Modeling Technique, Boolean Equation Programming, and Behavioral Modeling.

Equipment Used:

- Modelsim Software

Methodology:

Prior to typing anything into Modelsim, the boolean equation was derived from the given circuit schematic. Additionally, the truth table was derived, which was used later to check each simulation.

Gate Level Modeling is the action of programming each logic gate into a hardware description language to create a simulation of a particular boolean function. Within the .v file utilized for modeling the circuit in this manner, each individual gate's meaning was typed out (and/or/not) with its output expressed first followed by the two inputs. The gates that didn't connect directly to the output, but rather to higher-level gates within the same circuit, were given special "wire" outputs, and the higher-level gates took some of these as their inputs.

To run this file, another file, known as a testbench, was also programmed. Testbenches feed the main program file the input conditions and combinations at each particular timestamp within the simulation. This first testbench ran off of the monitor function, which states the timestamp in which each input changes between states.

Once the program and testbench files compiled successfully, a simulation was performed through the "Simulate" tab at the top of the screen. After adding all the signal waves from the inputs and outputs, the simulation was ran, and the signal waves were generated with respect to time. Whenever a particular wave was at its lower position, that corresponds to a logic value of '0', whereas when it was at its higher position, it resembled a logic '1'.

The second Verilog programming method, Boolean Equation programming, requires the fewest lines of code to create a circuit simulation. With this method, the output variable was assigned to equal the given boolean function of the inputs. Another testbench was created to test this version of the circuit, except this time, instead of utilizing the monitor function, it read the input combinations from a third file, known as a test vector. The simulation itself was then prepared and performed in the same manner as before.

The third and final programming method utilized in Modelsim was Behavioral Programming. Instead of treating the boolean function as a set of logic gates or an equation, it was now treated in a similar manner to a C/C++ program. This iteration of the program utilized an if/else statement to describe when the output should be a logic '1' and when it should be a '0'.

Introduction to Hardware Description Languages with Verilog
Jake D. Karas
U0000008780
10/14/2021

Equation and Truth Table:

$$F(A, B, C, D) = A'D' + (BC + BD') = A'D' + BC + BD'$$

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

ModelSim Circuit Codes and Output Waves:

Gate Level Modeling Technique Programming:

Main Program:

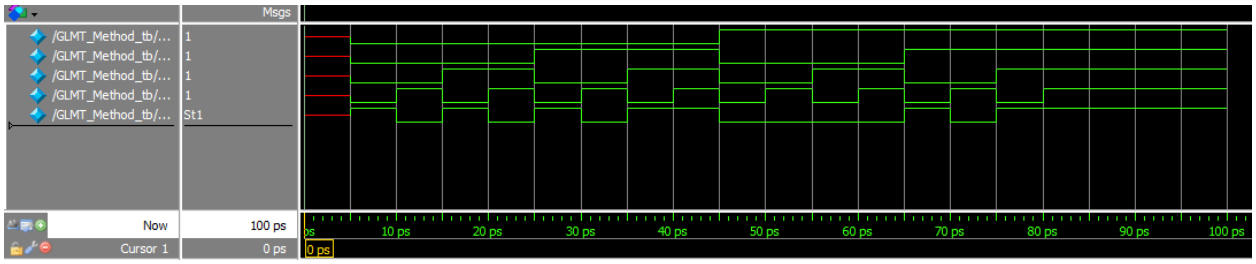
```
module GLMT(A,B,C,D,x); //Declare all input/output variables.
input A,B,C,D;  //Assign each variable to one of three types(input, output, or wire).
output x;
wire e,f,g,h,i; //Create additional wire variables.
not n1(e,A);    //Output of each not first, then its input.
not n2(f,D);
and a1(g,e,f);  //Output of each and/or first, then its inputs.
and a2(h,B,C);
and a3(i,f,B);
or o1(j,h,i);
or o2(x,g,j);
endmodule
```

TestBench:

```
module GLMT_Method_tb();
reg A_in, B_in, C_in, D_in;
wire x_out;

initial
begin
$monitor("simtime=%g, A_in=%b, B_in=%b, C_in=%b, D_in=%b, x_out=%b",
$time, A_in, B_in, C_in, D_in, x_out);

end

GLMT r1(.A(A_in), .B(B_in), .C(C_in), .D(D_in), .x(x_out));

initial
begin

#5 A_in=1'b0; B_in=1'b0; C_in=1'b0; D_in=1'b0;
#5 A_in=1'b0; B_in=1'b0; C_in=1'b0; D_in=1'b1;
#5 A_in=1'b0; B_in=1'b0; C_in=1'b1; D_in=1'b0;
#5 A_in=1'b0; B_in=1'b0; C_in=1'b1; D_in=1'b1;
#5 A_in=1'b0; B_in=1'b1; C_in=1'b0; D_in=1'b0;
#5 A_in=1'b0; B_in=1'b1; C_in=1'b0; D_in=1'b1;
#5 A_in=1'b0; B_in=1'b1; C_in=1'b1; D_in=1'b0;
#5 A_in=1'b0; B_in=1'b1; C_in=1'b1; D_in=1'b1;
#5 A_in=1'b1; B_in=1'b0; C_in=1'b0; D_in=1'b0;
#5 A_in=1'b1; B_in=1'b0; C_in=1'b0; D_in=1'b1;
#5 A_in=1'b1; B_in=1'b0; C_in=1'b1; D_in=1'b0;
#5 A_in=1'b1; B_in=1'b0; C_in=1'b1; D_in=1'b1;
#5 A_in=1'b1; B_in=1'b1; C_in=1'b0; D_in=1'b0;
#5 A_in=1'b1; B_in=1'b1; C_in=1'b0; D_in=1'b1;
#5 A_in=1'b1; B_in=1'b1; C_in=1'b1; D_in=1'b0;
#5 A_in=1'b1; B_in=1'b1; C_in=1'b1; D_in=1'b1;

end
endmodule
```

Introduction to Hardware Description Languages with Verilog
Jake D. Karas
U0000008780
10/14/2021

Output Waves:



Boolean Equation Programming:

Main Program:

```
module Bool_Eq(A,B,C,D,x);  //Declare all variables.
 input A,B,C,D;  //Define inputs.
 output x;  //Define outputs.
 assign x = (~A&~D) | ((B&C) | (~D&B));  //Assign the outpt variable in a similar manner to C or C++, but with logic operators.
endmodule
```

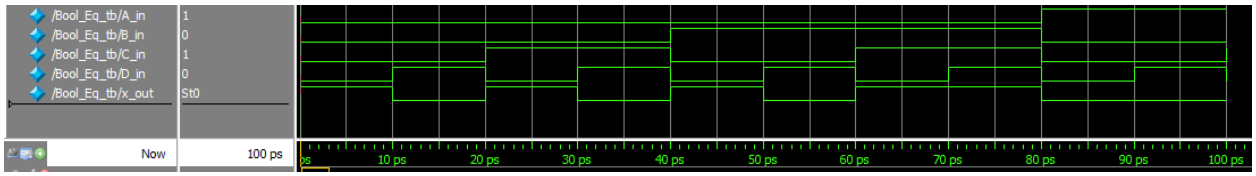
TestBench:

```
module Bool_Eq_tb();
reg A_in, B_in, C_in, D_in;
wire x_out;

reg [3:0] testvectors [15:0];   //Make a 4-bit register for 16 combinations.
integer i;      //For-loop control variable.

Bool_Eq r1 (.A(A_in), .B(B_in), .C(C_in), .D(D_in), .x(x_out));

initial begin
$readmemb ("C:/Users/Jake/Documents/ModelSim/Lab 4/Bool_Eq_testvector.tv", testvectors);       // Bool_Eq_testvecor.tvis a testvector which represents the truth table input combinations.
end

always begin
for (i=0; i<15; i=i+1) begin
{A_in, B_in, C_in, D_in} = testvectors[i];      //reads vectors from testvectors and assign those to A_in, B_in, C_in, and D_in.
#10;
end
end

endmodule
```

TestVector:

```
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
```

Output Waves:



Behavioral Modeling:

```
module BehavioralModeling(A, B, C, D, x);  //Declare all variables.
input A, B, C, D;  //Define inputs.
output x;  //Define outputs.
reg x;
always @ (A or B or C)
begin
        if ((A==0 && D == 0) || ((B==1 && C==1) || (B==1 && D==0)))  //Program an if/else statement similar to C/C++ to control the output.
                x = 1;
        else    x = 0;
end
endmodule
```

Conclusion:

These three hardware description language programming methods are important to logic circuit designers in a number of approaches. For instance, it allows one to easily test a logic circuit design prior to purchasing any physical components, saving on design costs. Also, it is easier to test simplifications of a particular circuit prior to building it with hardware, reducing both costs and internal delay between the time of input state changes and output state changes.