**Prob 1**

**1.1**

**Verilog**

module cp\_4bit (

input [3:0] A ,

input [3:0] B ,

output A\_lt\_B,

output A\_eq\_B,

output A\_gt\_B

);

wire X3; // X3 = A[3] xnor B[3]

wire X2; // X2 = A[2] xnor B[2]

wire X1; // X1 = A[1] xnor B[1]

wire X0; // X0 = A[0] xnor B[0]

wire outl1; // outl1 = (~A[3]) & B[3]

wire outl2; // outl2 = X3 & (~A[2]) & B[2]

wire outl3; // outl3 = X3 & X2 & (~A[1]) & B[1]

wire outl4; // outl4 = X3 & X2 & X1 & (~A[0]) & B[0]

wire outg1; // outg1 = A[3] & (~B[3])

wire outg2; // outg2 = X3 & A[2] & (~B[2])

wire outg3; // outg3 = X3 & X2 & A[1] & (~B[1])

wire outg4; // outg4 = X3 & X2 & X1 & A[0] & (~B[0])

// X3 = A[3] xnor B[3]

xnor x0(X3, A[3], B[3]);

// X2 = A[2] xnor B[2]

xnor x1(X2, A[2], B[2]);

// X1 = A[1] xnor B[1]

xnor x2(X1, A[1], B[1]);

// X0 = A[0] xnor B[0]

xnor x3(X0, A[0], B[0]);

// outl1 = (~A[3]) & B[3]

and a0(outl1, ~A[3], B[3]);

// outl2 = X3 & (~A[2]) & B[2]

and a1(outl2, X3, ~A[2], B[2]);

// outl3 = X3 & X2 & (~A[1]) & B[1]

and a3(outl3, X3, X2, ~A[1], B[1]);

// outl4 = X3 & X2 & X1 & (~A[0]) & B[0]

and a4(outl4, X3, X2, X1, ~A[0], B[0]);

// A\_lt\_B = outl1 | outl2 | outl3 | outl4

or o0(A\_lt\_B, outl1, outl2, outl3, outl4);

// A\_eq\_B = X3 & X2 & X1 & X0

and a5(A\_eq\_B, X3, X2, X1, X0);

// outg1 = A[3] & (~B[3])

and a6(outg1, A[3], ~B[3]);

// outg2 = X3 & A[2] & (~B[2])

and a7(outg2, X3, A[2], ~B[2]);

// outg3 = X3 & X2 & A[1] & (~B[1])

and a8(outg3, X3, X2, A[1], ~B[1]);

// outg4 = X3 & X2 & X1 & A[0] & (~B[0])

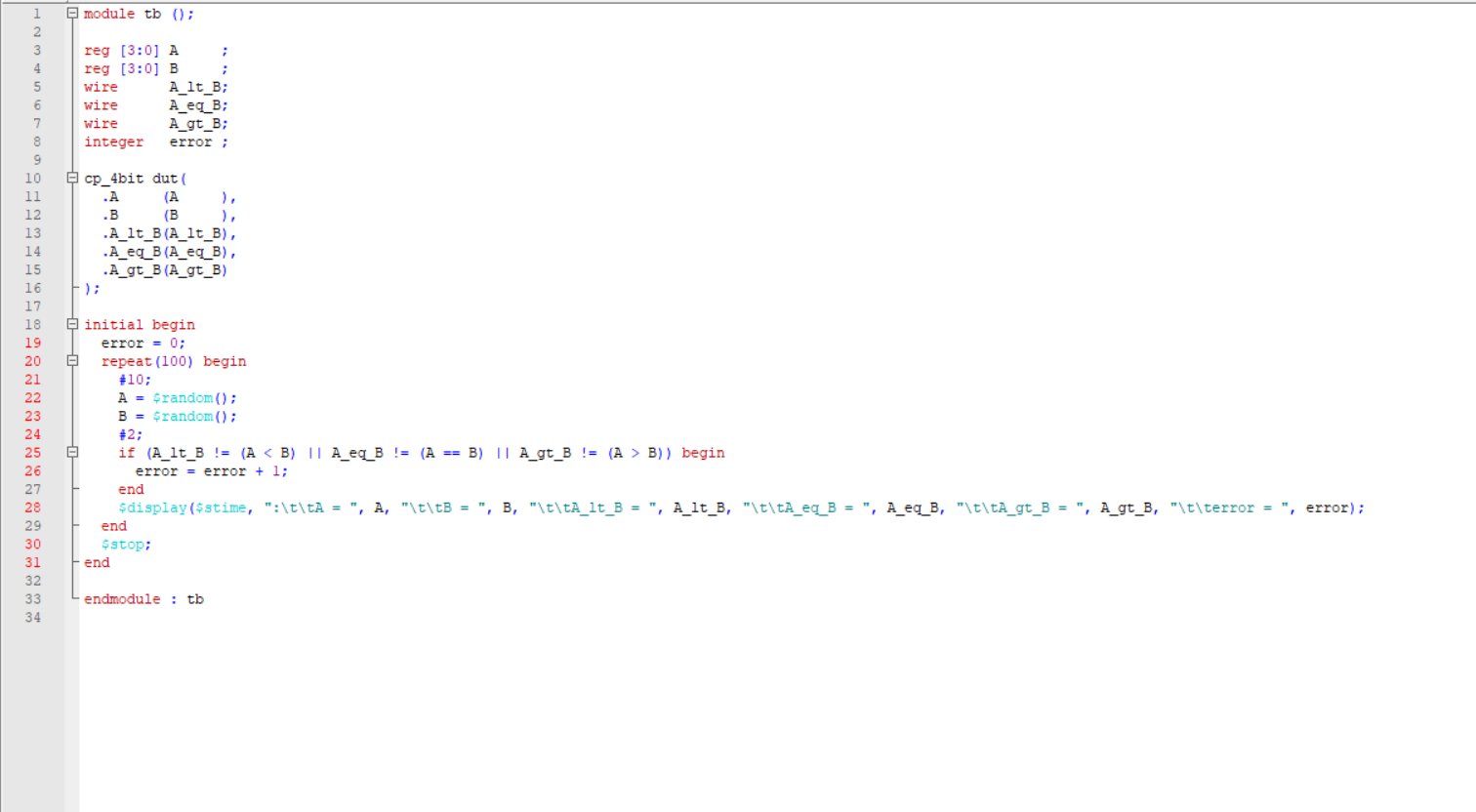
and a9(outg4, X3, X2, X1, A[0], ~B[0]);

// A\_gt\_B = outg1 | outg2 | outg3 | outg4

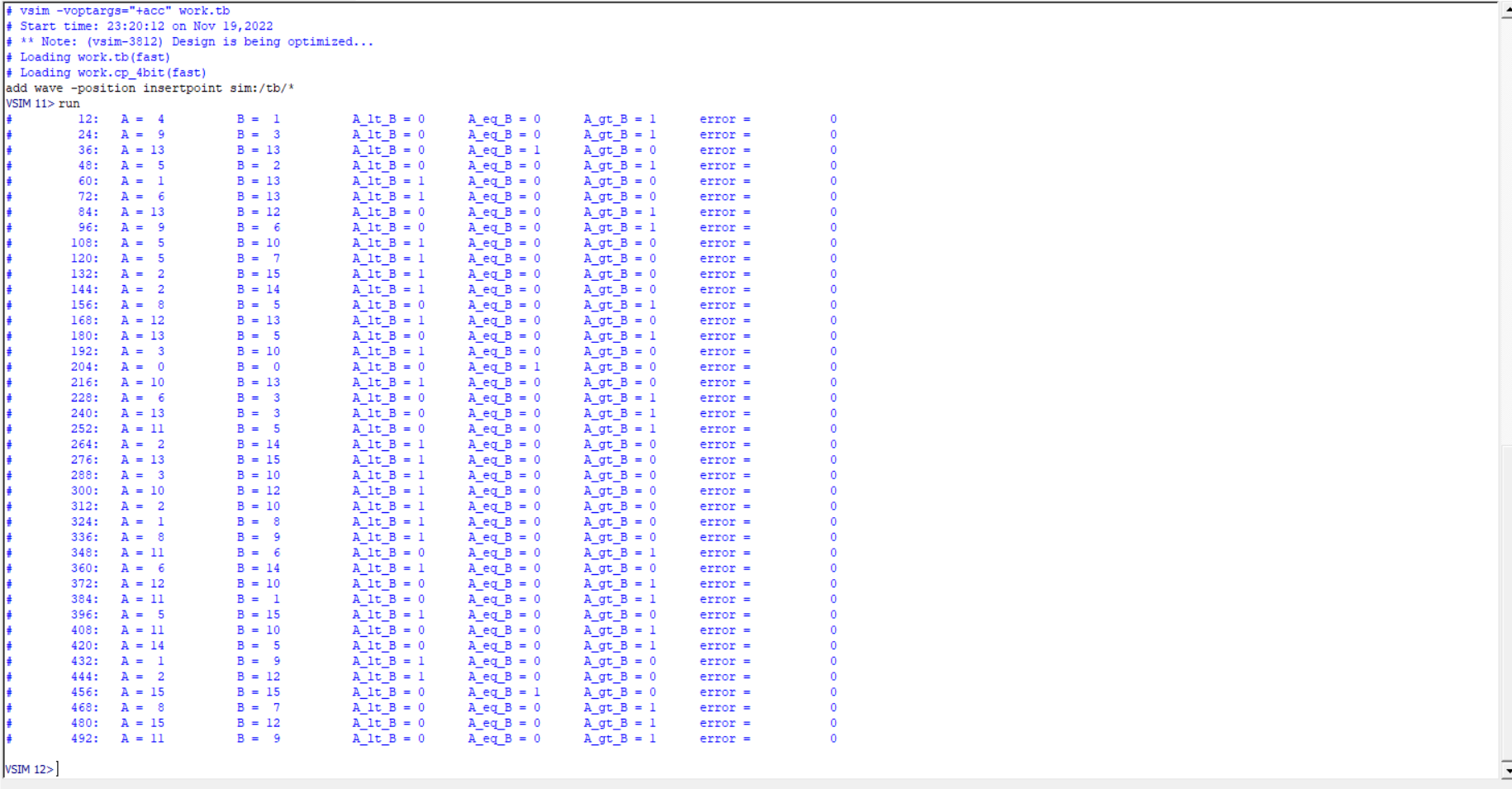
or o1(A\_gt\_B, outg1, outg2, outg3, outg4);

endmodule : cp\_4bit

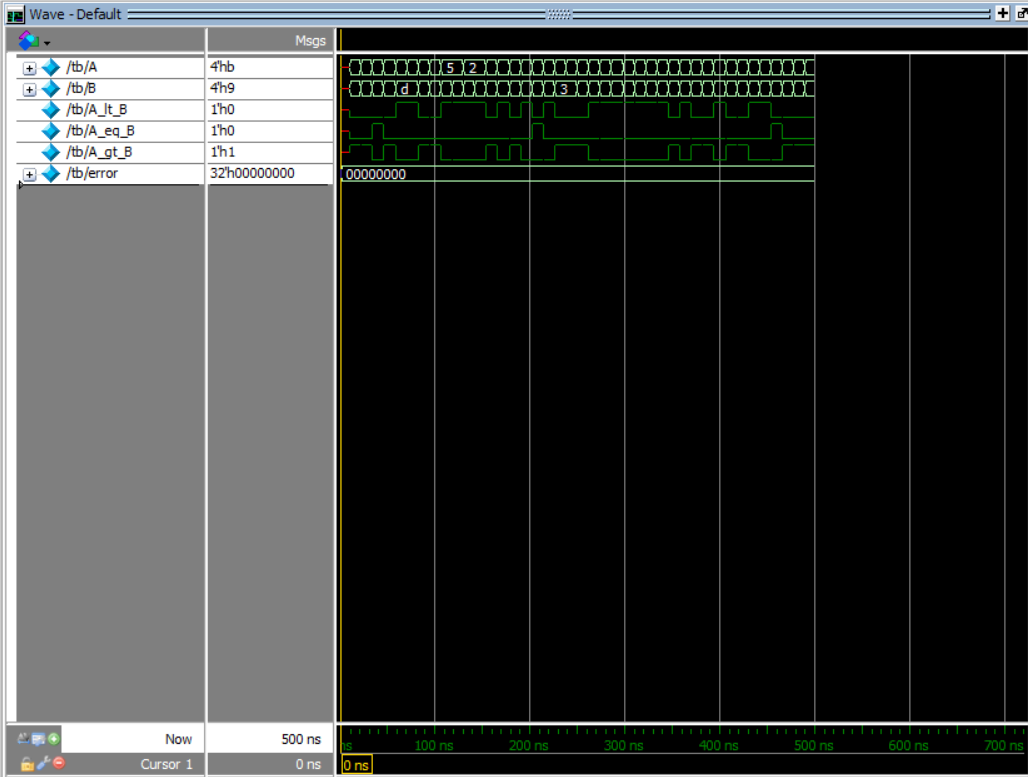
**Testbench**



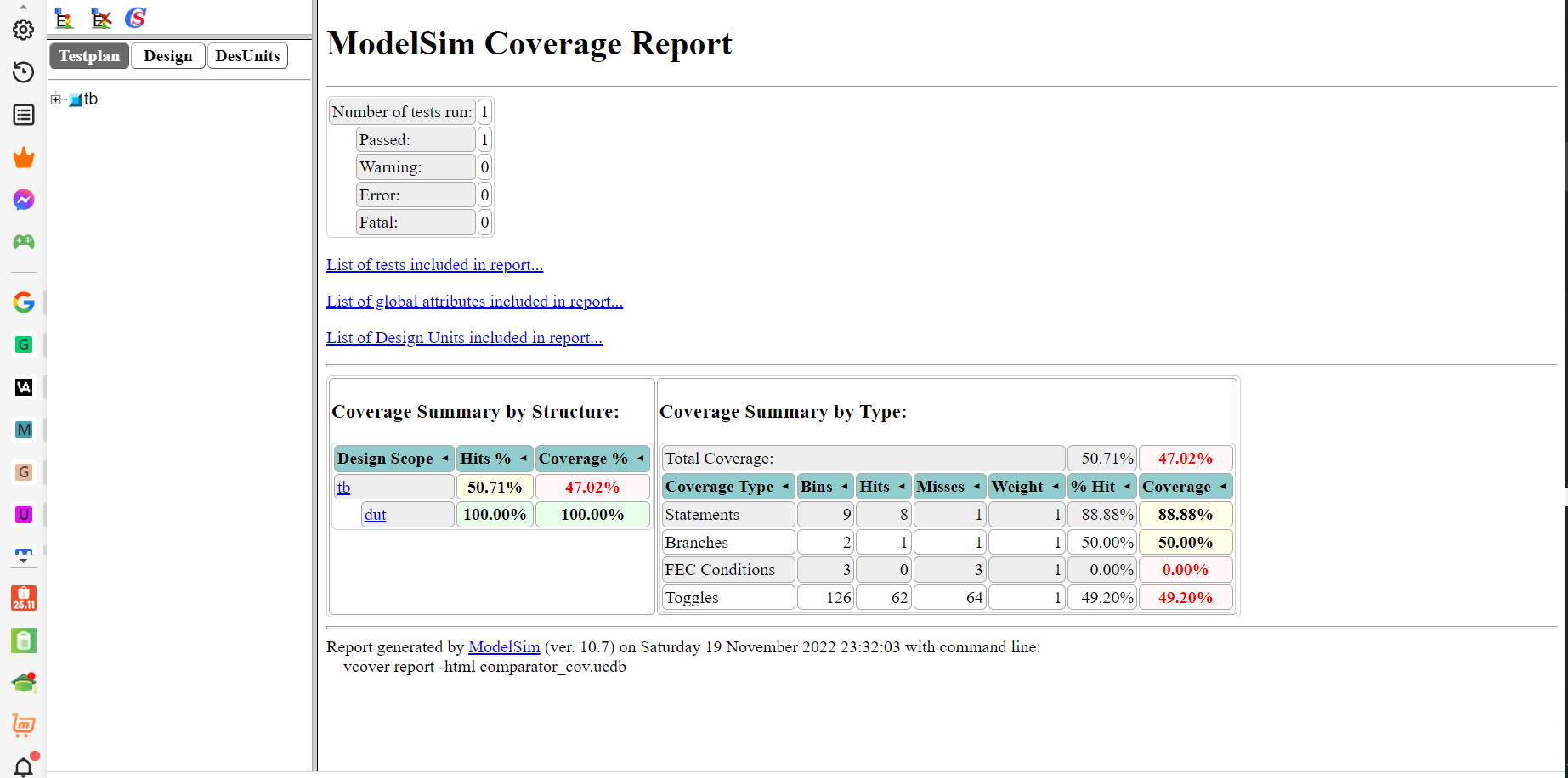
**Script**



**Wave**



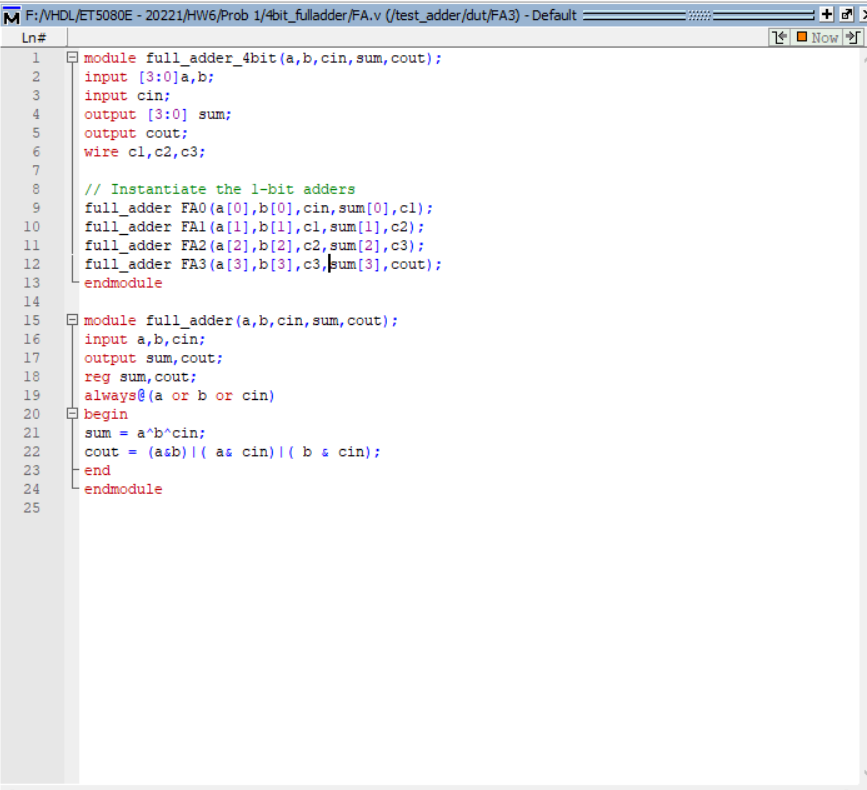
**Report**

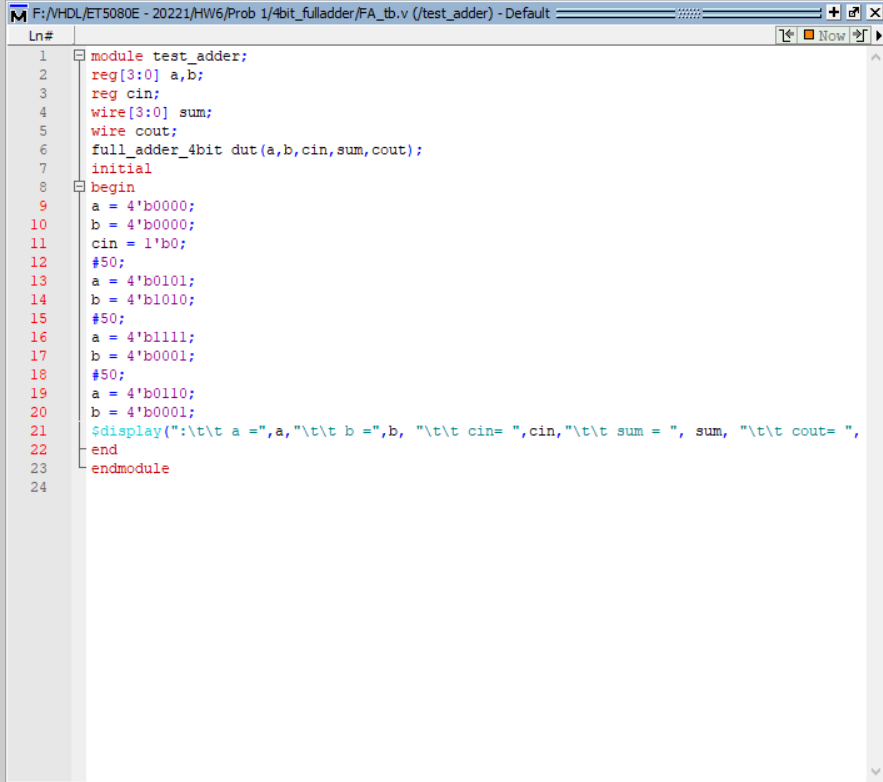
****

**1.2**

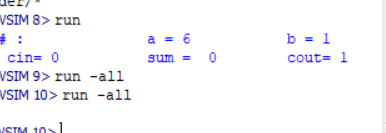
**Full adder 4 bit**

**Verilog**

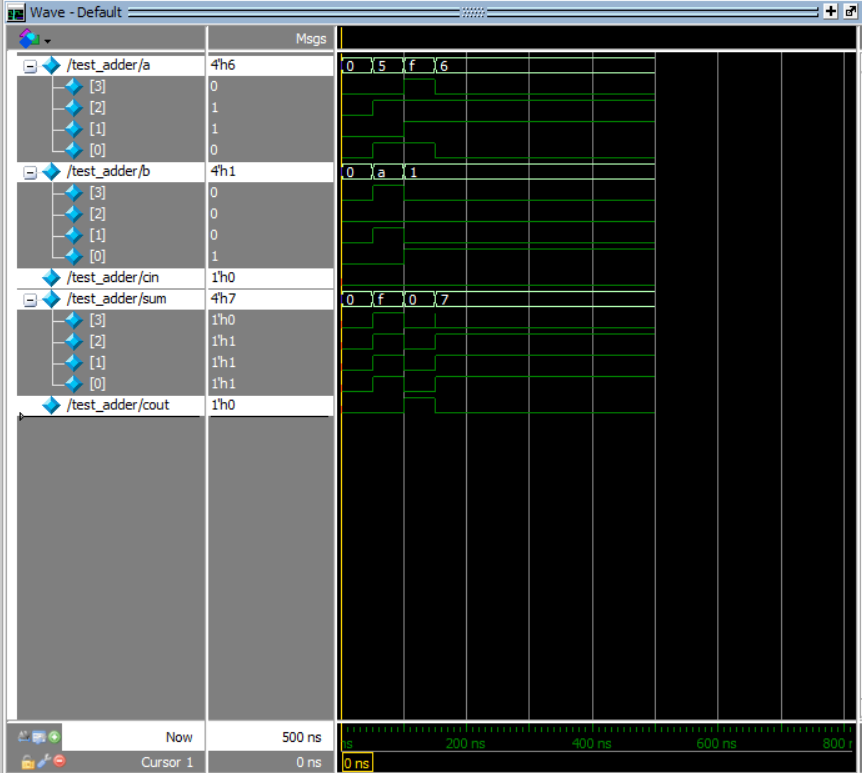


**Testbench**

**Script**



**Wave**

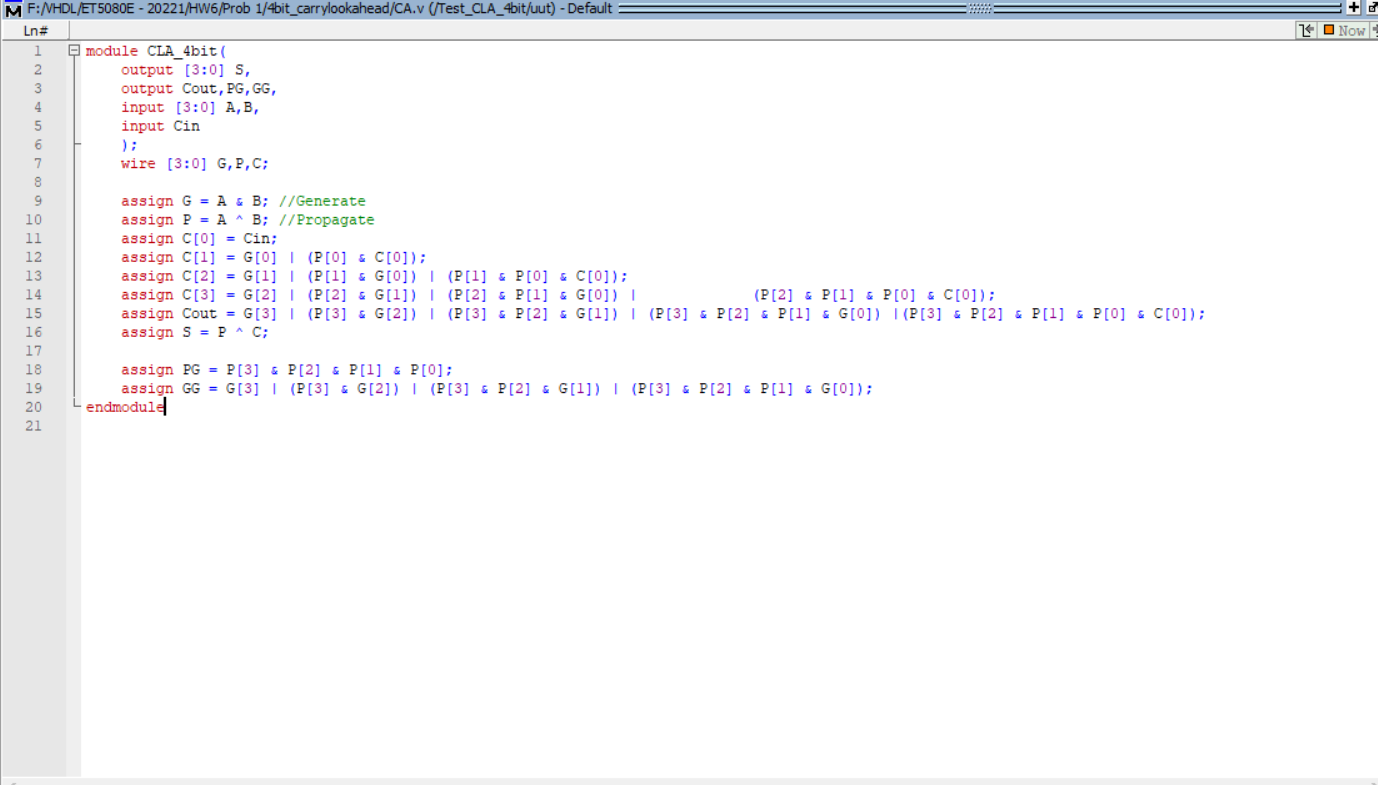


**Report**

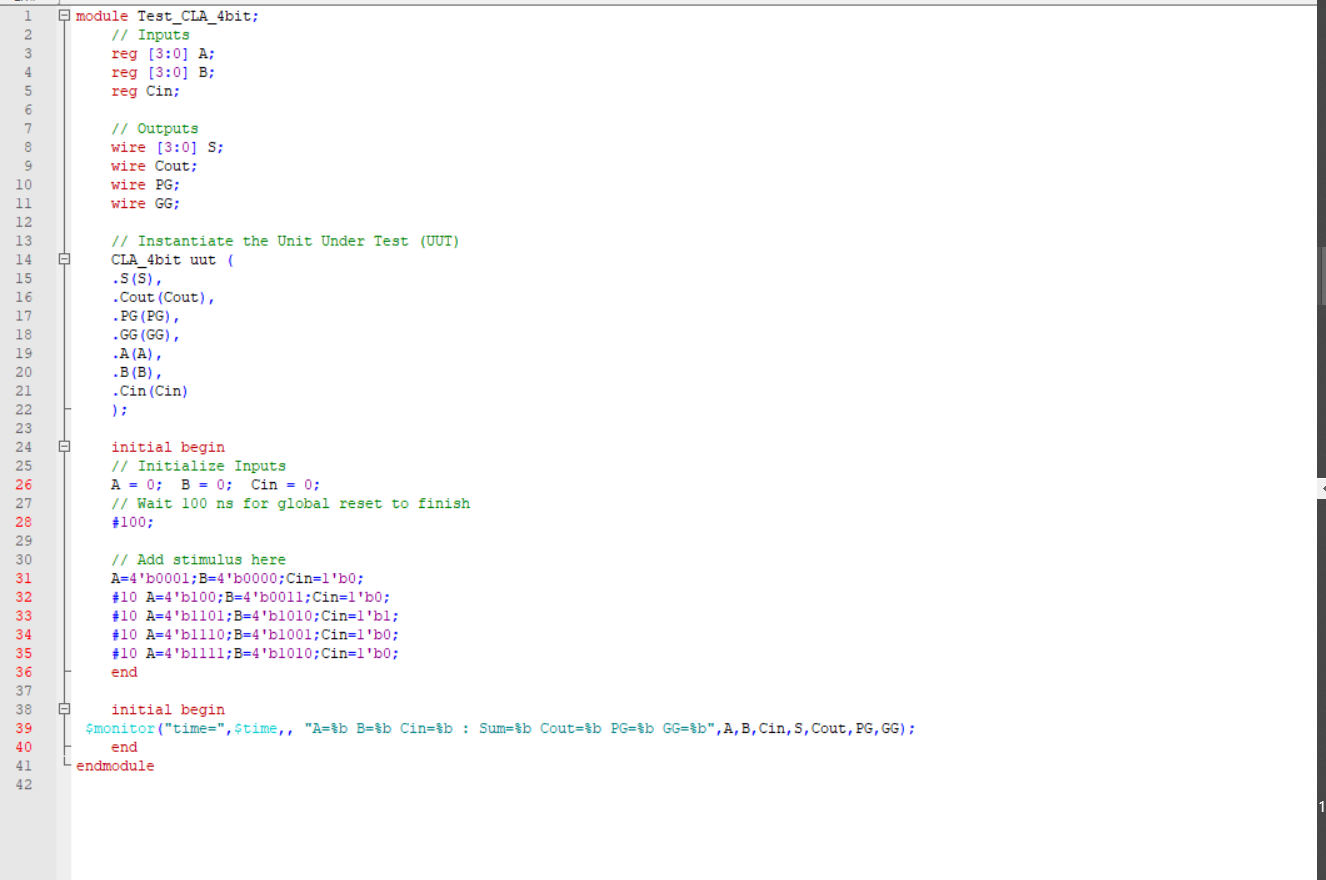


**Carry look ahead adder 4 bit**

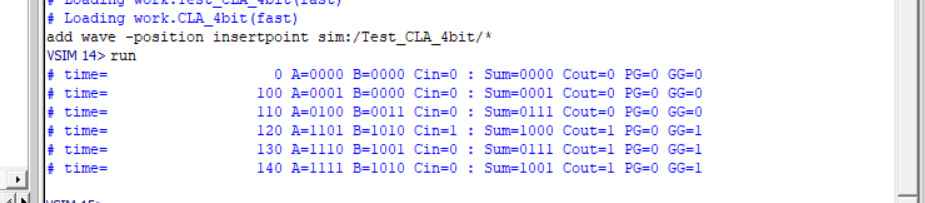
**Verilog**



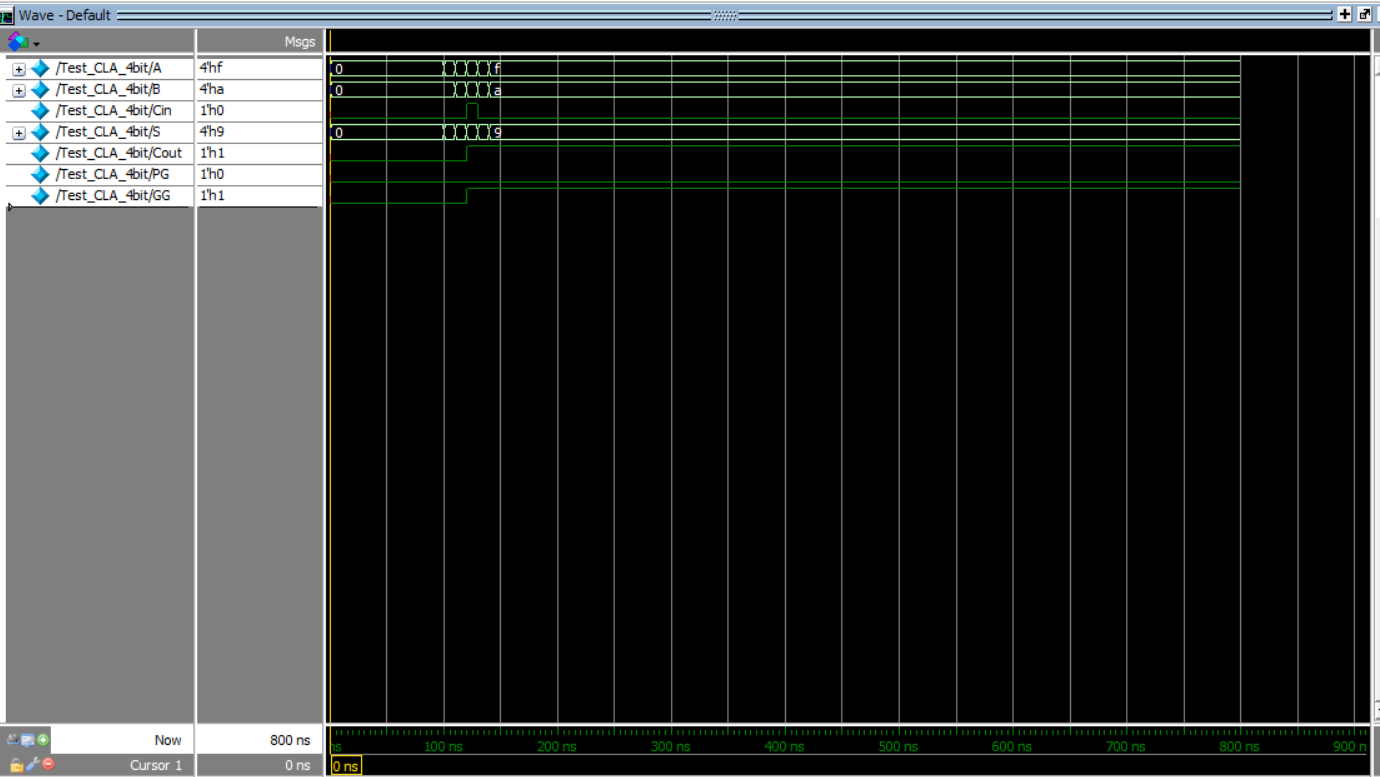
**Testbench**



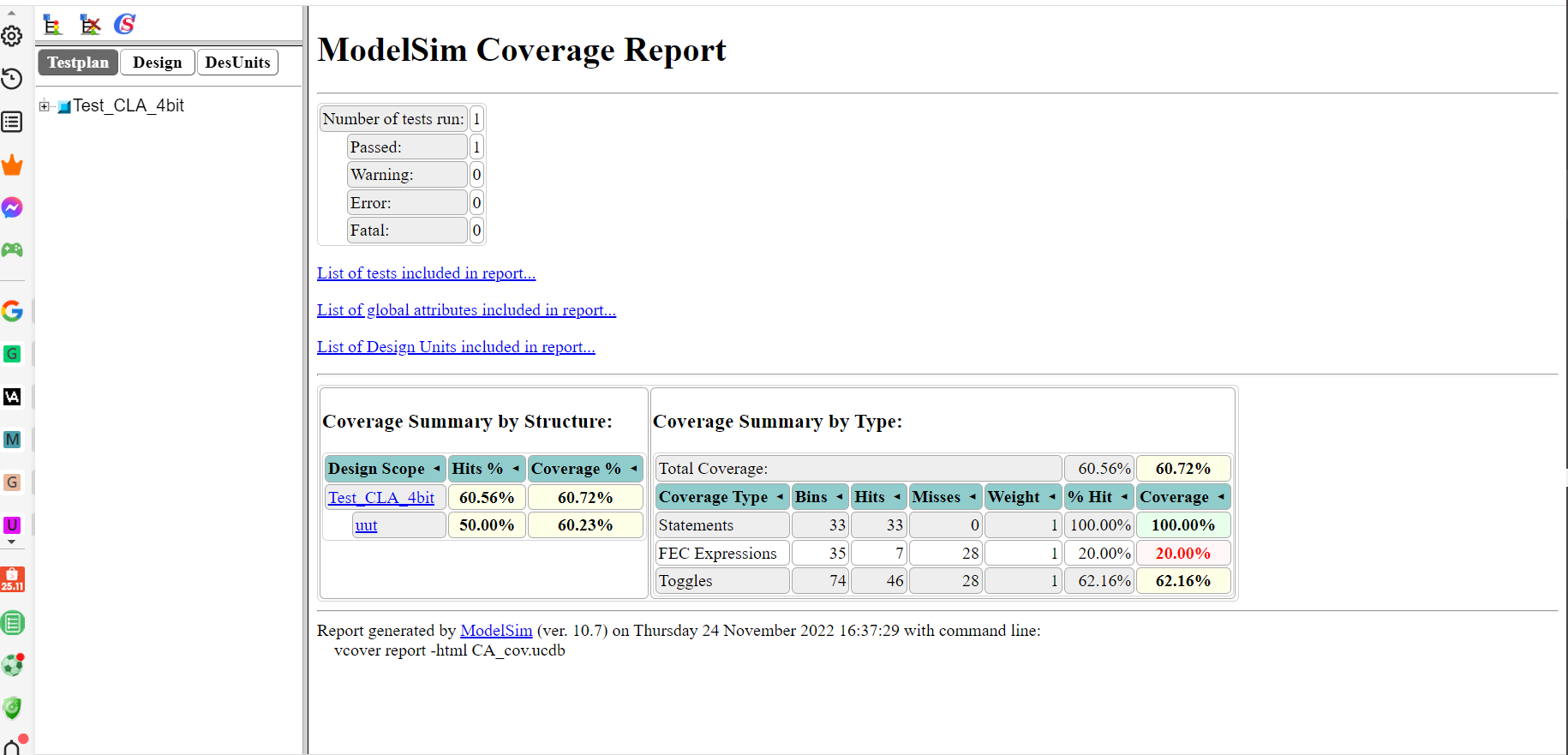
**Script**



**Wave**



**Report**



Structure, delay and speed

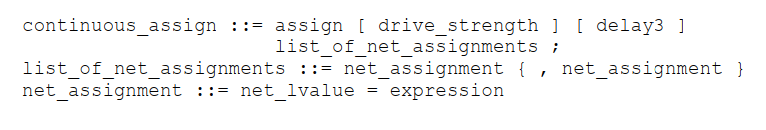
|  |  |  |
| --- | --- | --- |
|  | 4-bit Full Adder | 4-bit Carry Look Ahead Adder |
| Circuit |  | Carry-lookahead adder - Wikipedia |
| Structure | + Including n full adders connecting in series  + Each full adder has to wait for its carry-in from its prev stage full adder  + Thus, n th full adder has to wait until all (n – 1) full adders have completed their operations | + The carry-in of any full adder is independent of the carry bits generated during intermediate stages  + Known carry-in provided at the beginning and bits being added in the prev stages. Which enables the ability to evaluate the carry-in of any stages at the instant of time  + No need for waiting the carry-in generated by its prev stage full adder |
| Delay | High delay as n increases | Low delay |
| Speed | Extremely slow as n increases | Faster than 4-bit Full adder |

**Prob 2**

Prob 2:

1. **Statement assign**

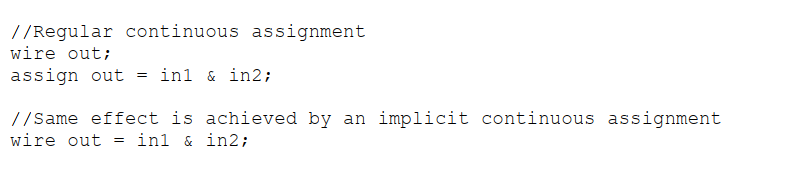
A continuous assignment is the most basic statement in dataflow modelling, used to drive a value onto a net. This assignment replaces gates in the description of the circuit and describes the circuit at a higher level of abstraction. The assignment statement starts with the keyword assign



Some further notations for the assignment:

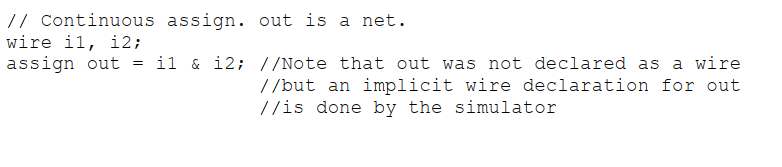
* The left hand side of an assignment must always be a scalar or vector net or a concatenation of scalar and vector nets. It cannot be a scalar or vector register
* Continuous assignments are always active. The assignment expression is evaluated as soon as one of the right-hand-side operands changes and the value is assigned to the left hand side net
* The operands on the right-hand side can be registers or nets or function calls. Registers or nets can be scalars or vectors.
* Delay values can be specified for assignments in terms of time units. Delay value are used to control the time when a net is assigned the evaluated value. This feature is similar to specifying delays for gates. It is very useful in modeling timing behavior in real circuits.
  1. Implicit Continuous Assignment

Instead of declaring a net and then writing a continuous assignment on the net, Verilog provides a shortcut by which a continuous assignment can be placed on a net when it is declared. There can be only 1 implicit declaration assignment per net because a net is declared only once.



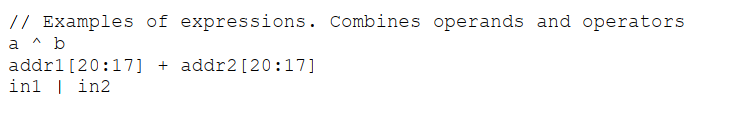
* 1. Implicit Net Declaration

If a signal name is used to the left of the continuous assignment, an implicit net declaration will be inferred for that signal name. If the net is connected to a modul port the width of the inferred net is equal to the width of the module port



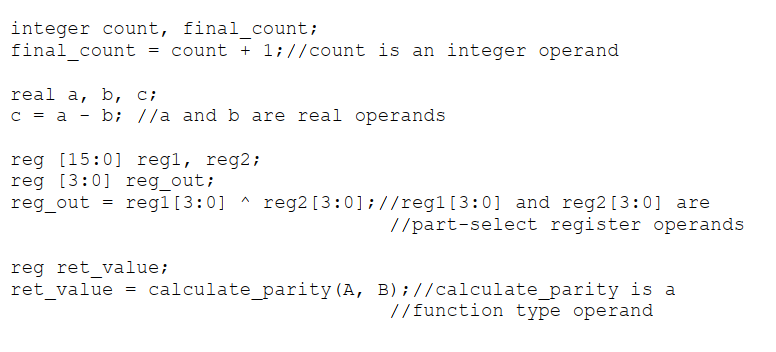
1. **Define, examples of expressions, operands, operator in Dataflow Modelling**
   1. Expressions

* Are constructs that combine operators and operands to produce a result



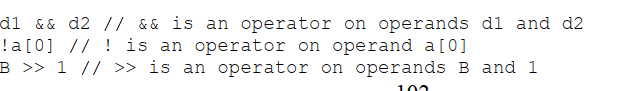
* 1. Operands

There are varied data types. Some constructs will take only certain types of operands. Operands can be constants, integers, real numbers, nets, registers, times, bit select, part select, memories or function calls



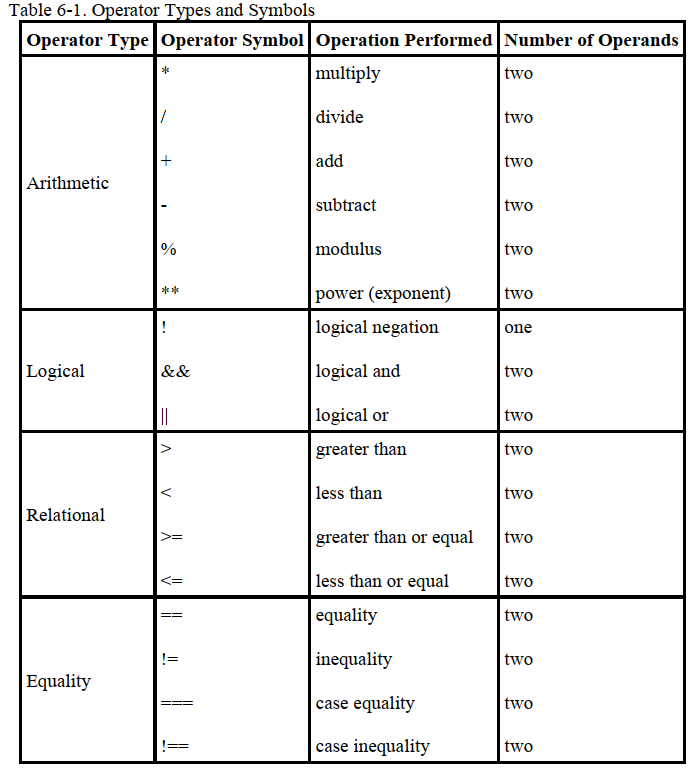
* 1. Operators

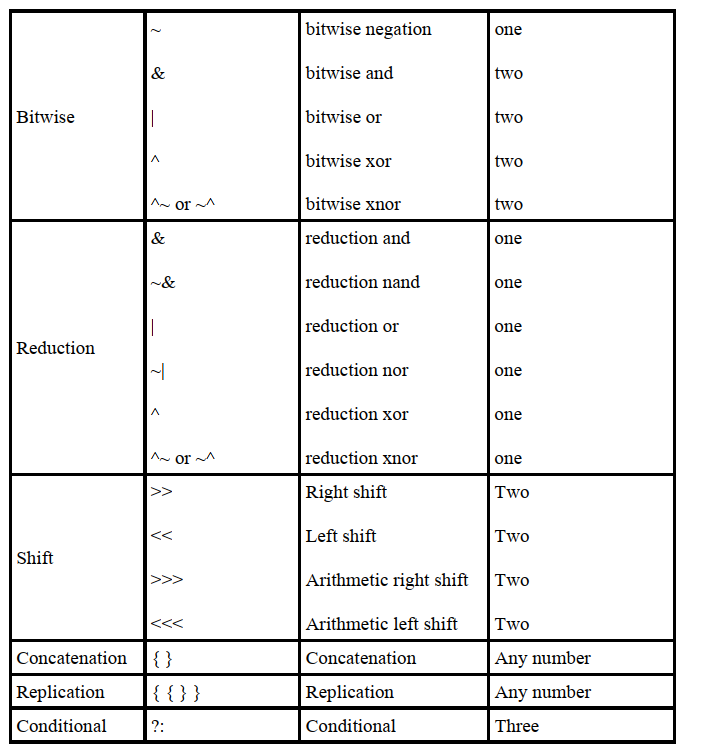
Act on the operands to produce desired results as various types of operators are provided in Verilog



1. **Dataflow Modelling operators**

This provides many different operator types. Operators can be arithmetic, logical, relational, equality, bitwise, reduction, shift, concatenation or conditional.





**Prob 3**

