**Name: DHVANI PATEL**

**Roll No: 21BCP116**

**Experiment 2**

**AIM: Implementation of basic gates using Verilog**

Theory:

Verilog is a Hardware Description Language (HDL). It is a language used for describing a digital system such as a network switch, a microprocessor, a memory, or a flip-flop. We can describe any digital hardware by using HDL at any level. Designs described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

Verilog was developed to simplify the process and make the HDL more robust and flexible. Today, Verilog is the most popular HDL used and practiced throughout the semiconductor industry.

HDL was developed to enhance the design process by allowing engineers to describe the desired hardware's functionality and let automation tools convert that behaviour into actual hardware elements like combinational gates and sequential logic.

The following are some keyword or components which are used in Verilog:

1. Wire:

A wire is used to represent a physical wire in a circuit and it is used for connection of gates or given modules. The value of a wire can only be read and not assigned in a function.

1. Register:

A reg (register) is an object, which is holding the value from one given assignment to next one to one and are used only in different functions also in procedural blocks. A reg is a simple Verilog, variable-type register to store data or number.

1. Assignment:

An assign keyword is used for assignment of variable to implement the given following gates. Where we point variable to combination of logic gates.

1. Output:

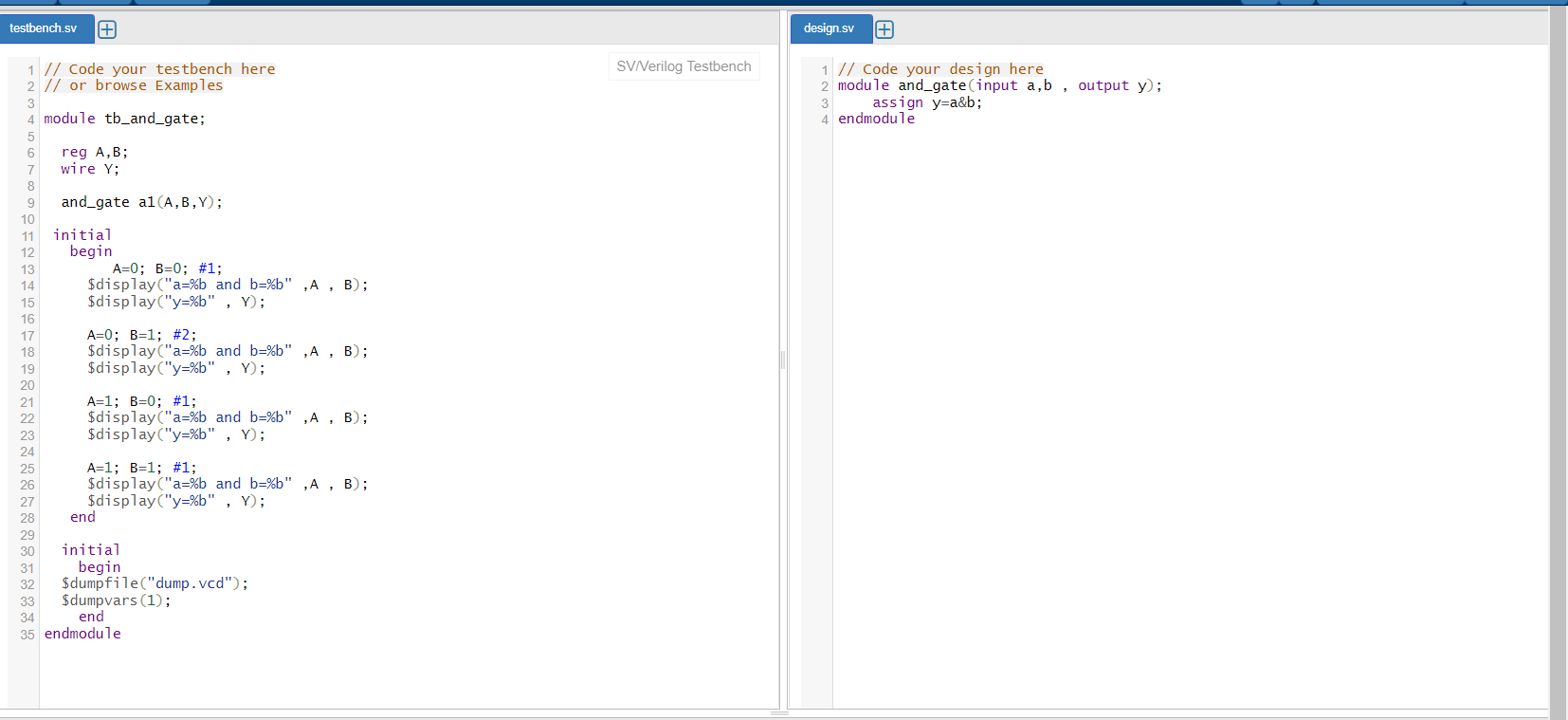
We use $display for the output in the Verilog system.

1. Module:

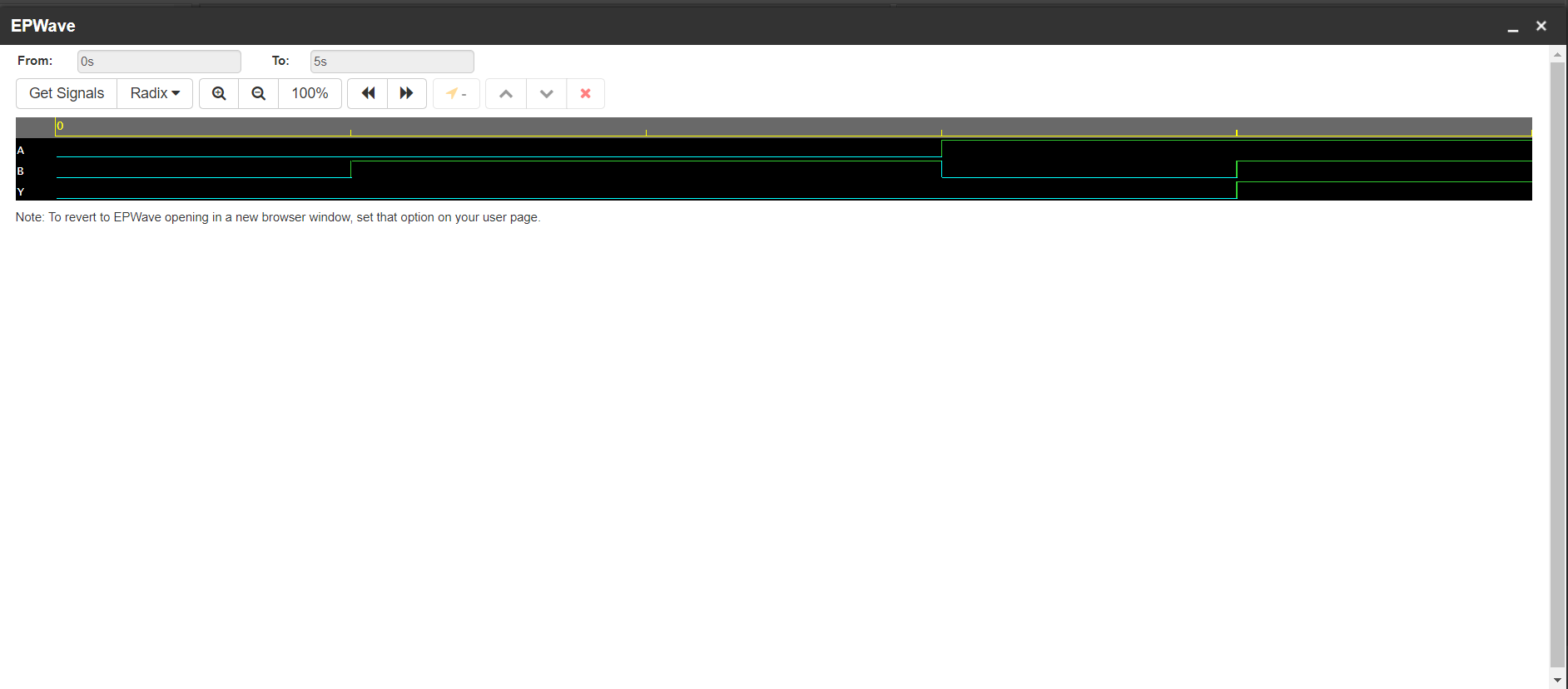
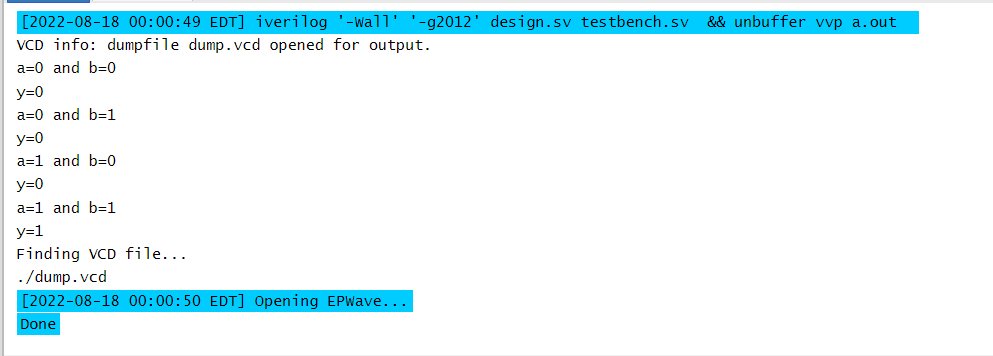
A module is used to import design of circuit in the output system , where we degin.sv and then import that module in following to testbench.

**Verification of AND Gate :**

Code :

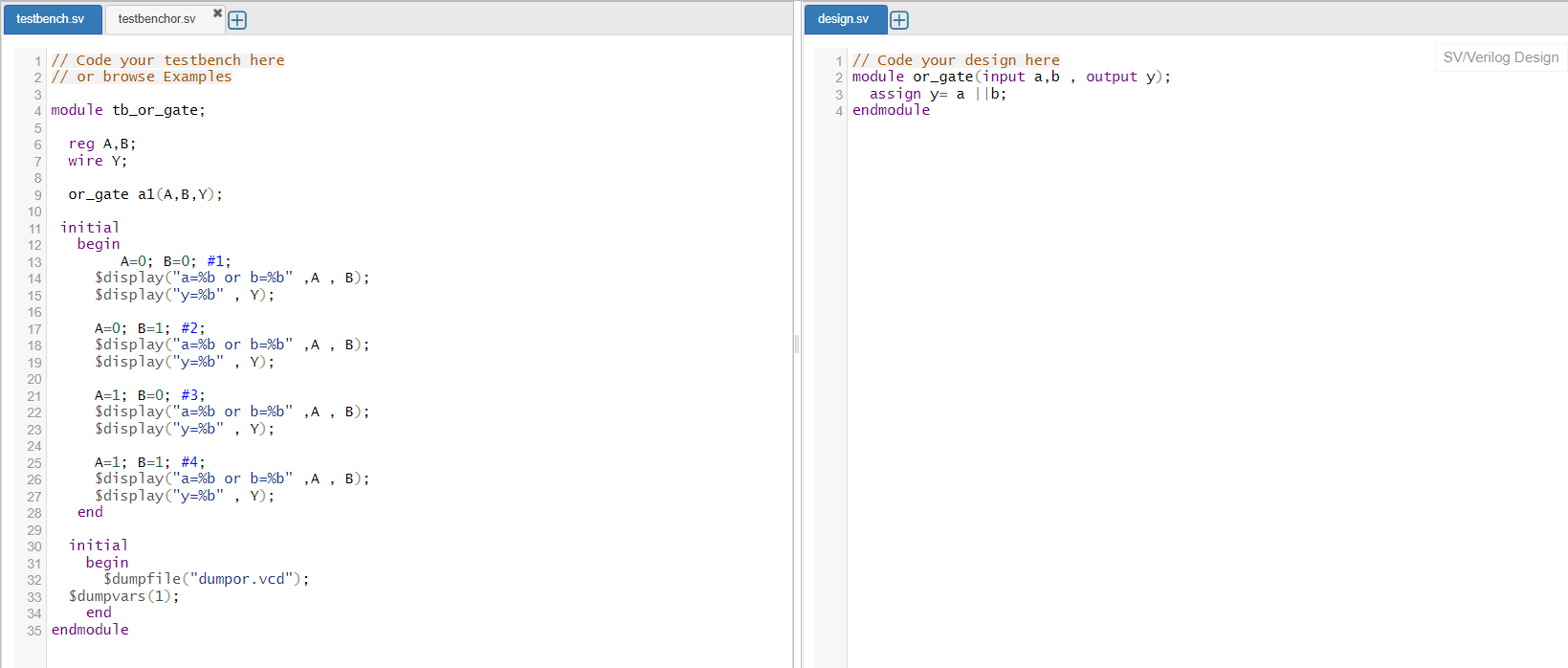


Output:

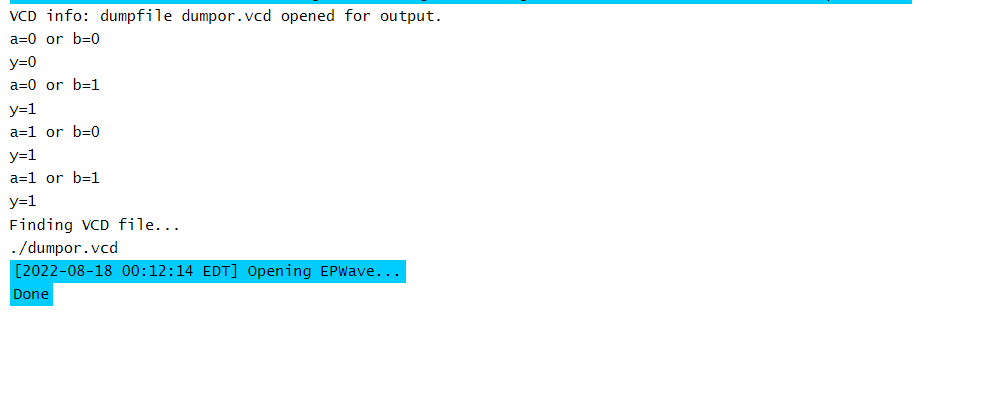


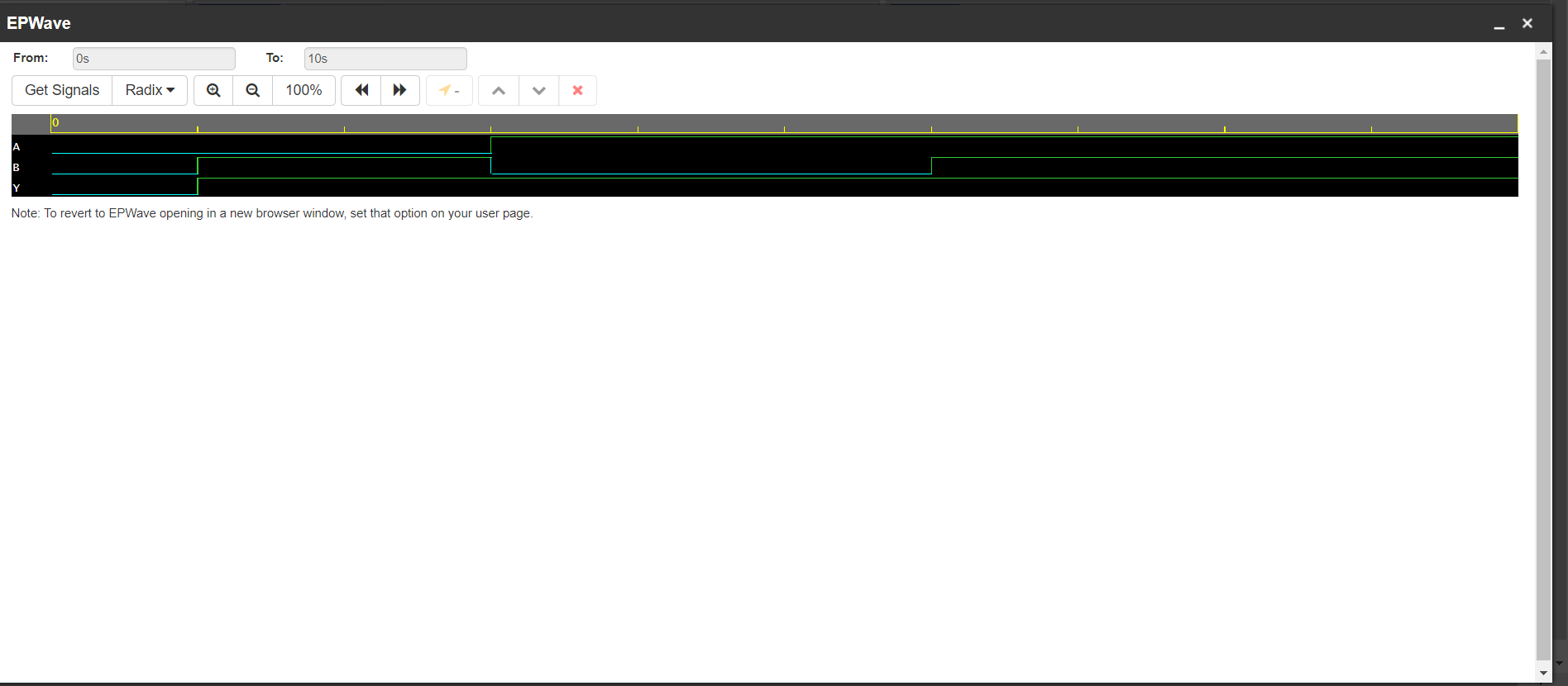
**Verification of OR Gate :**

Code :



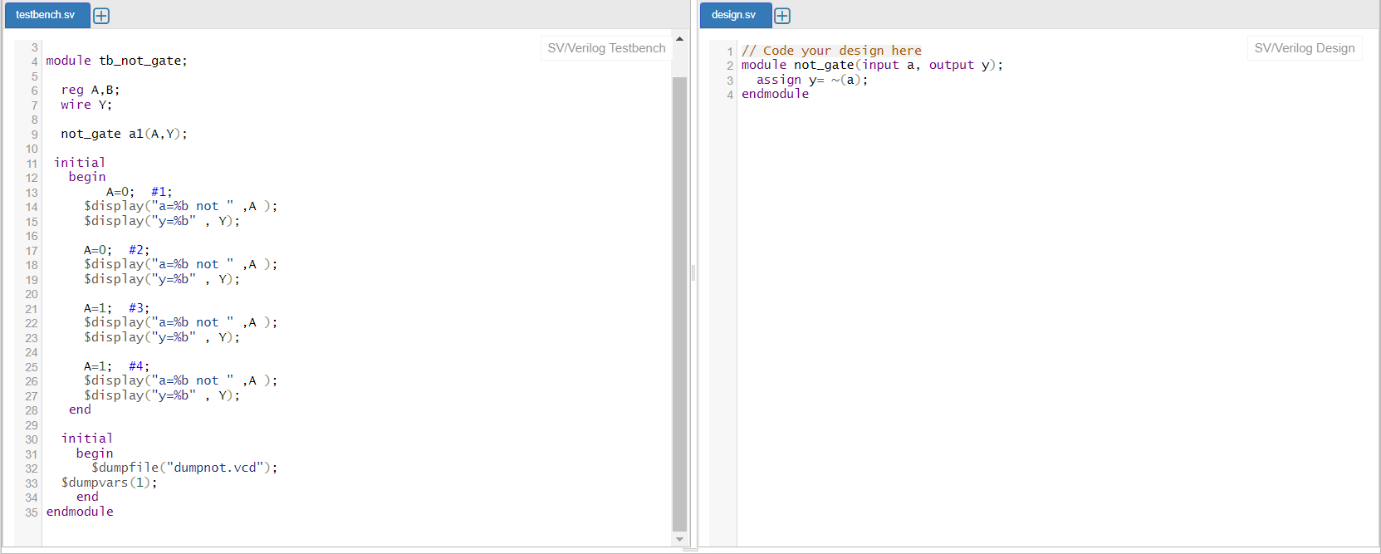
Output:



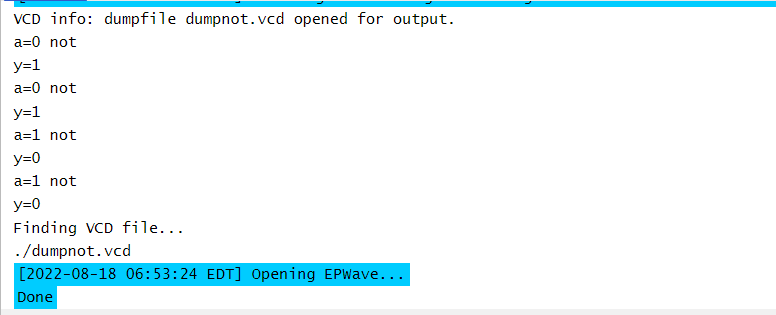


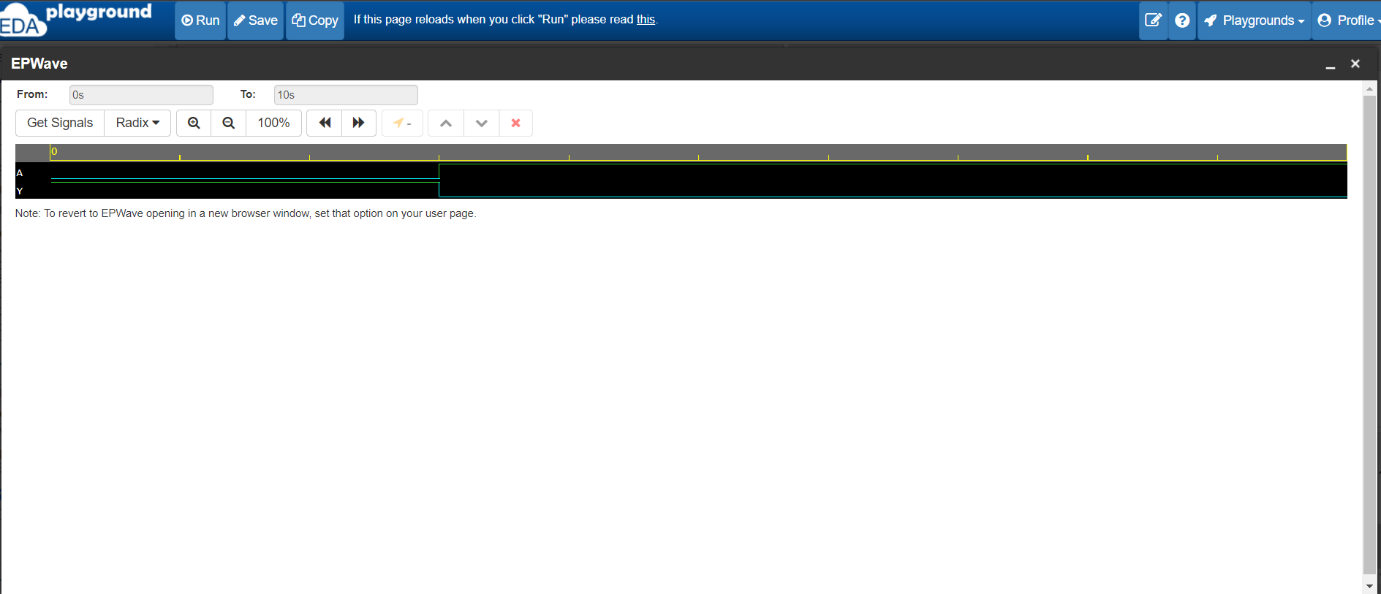
**Verification of NOT Gate :**

Code :



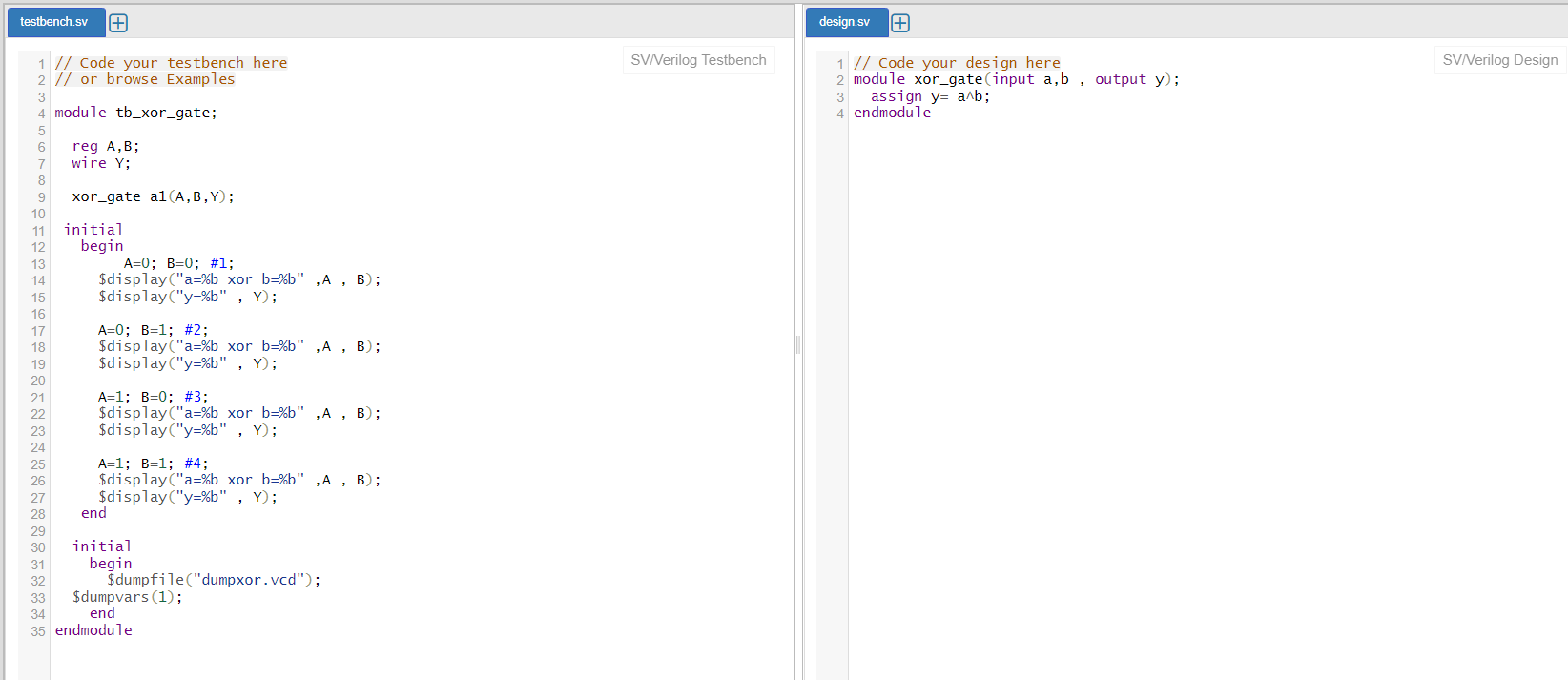
Output:



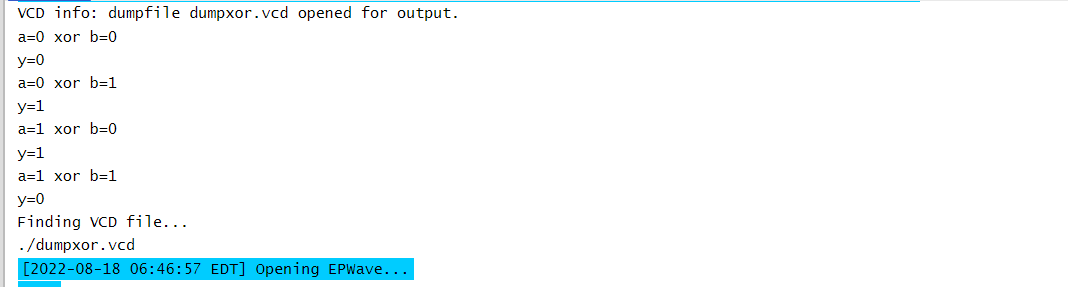


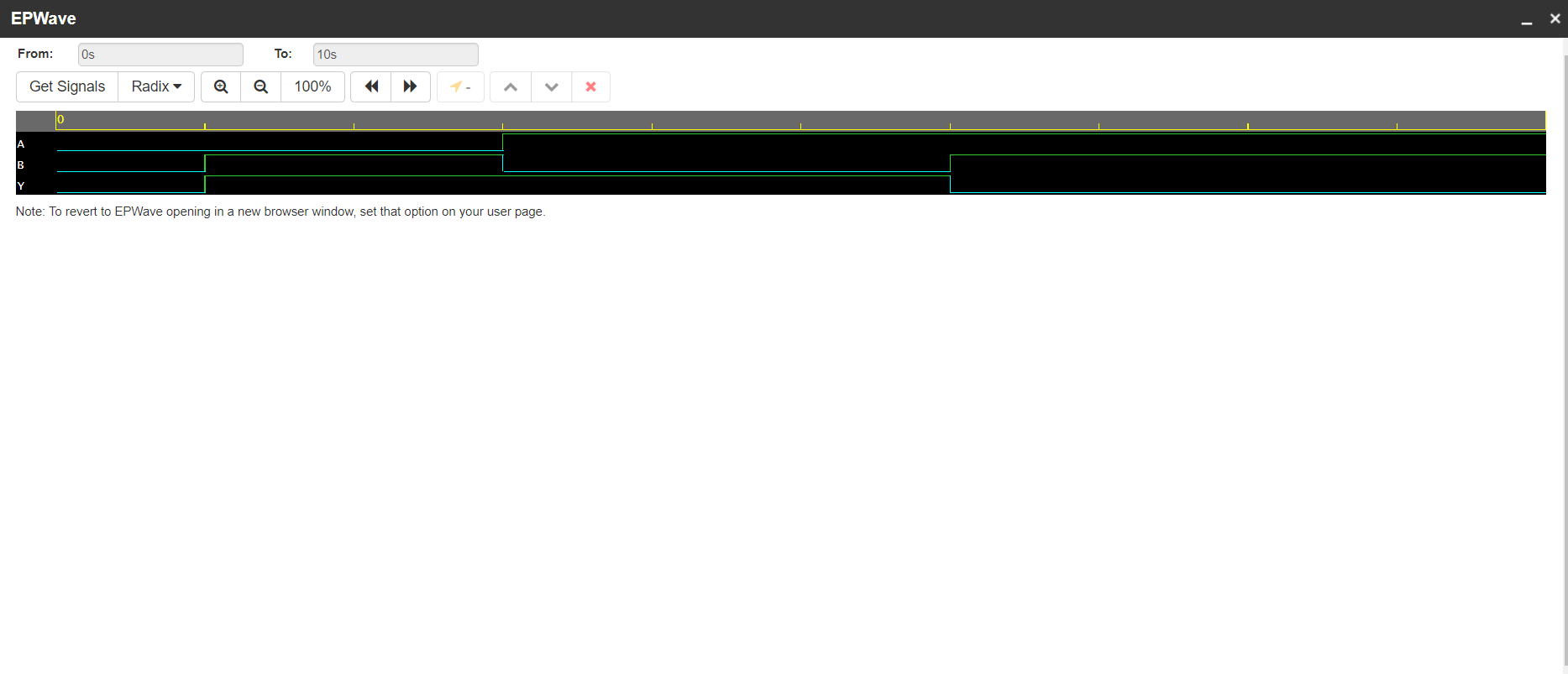
**Verification of XOR Gate :**

Code :



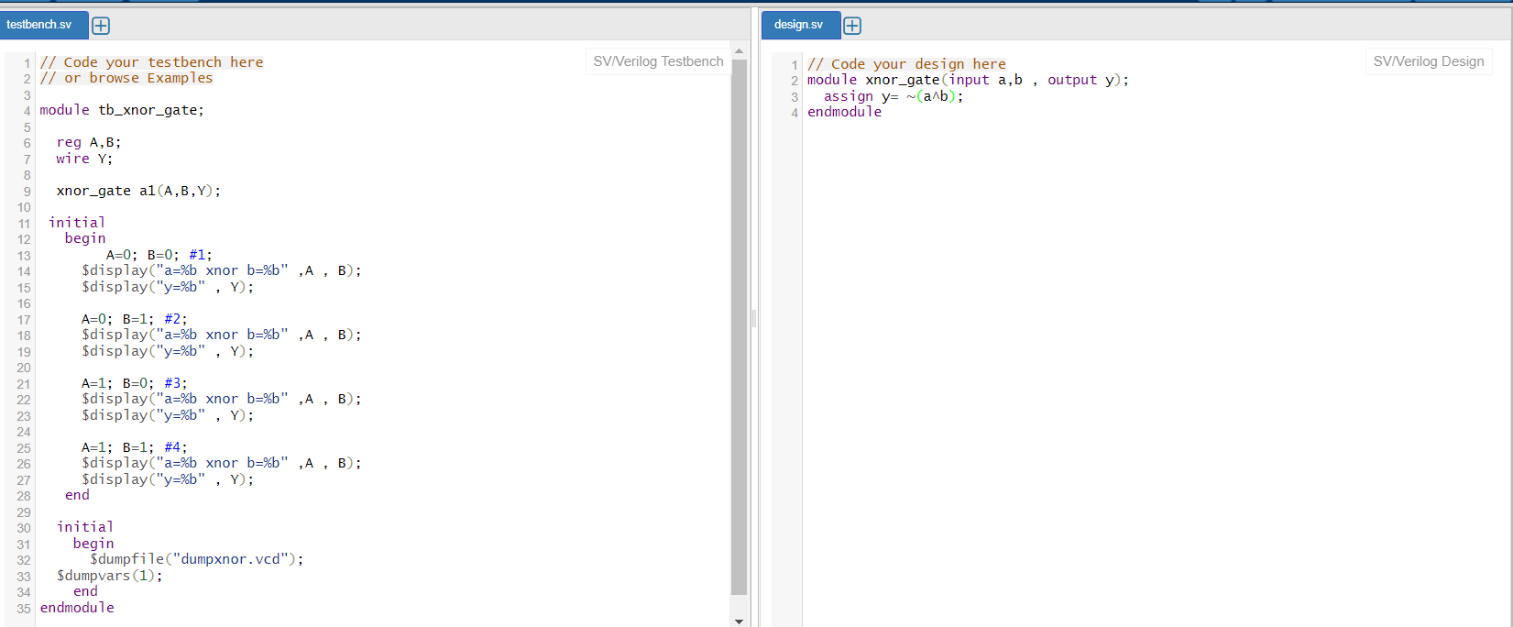
Output:



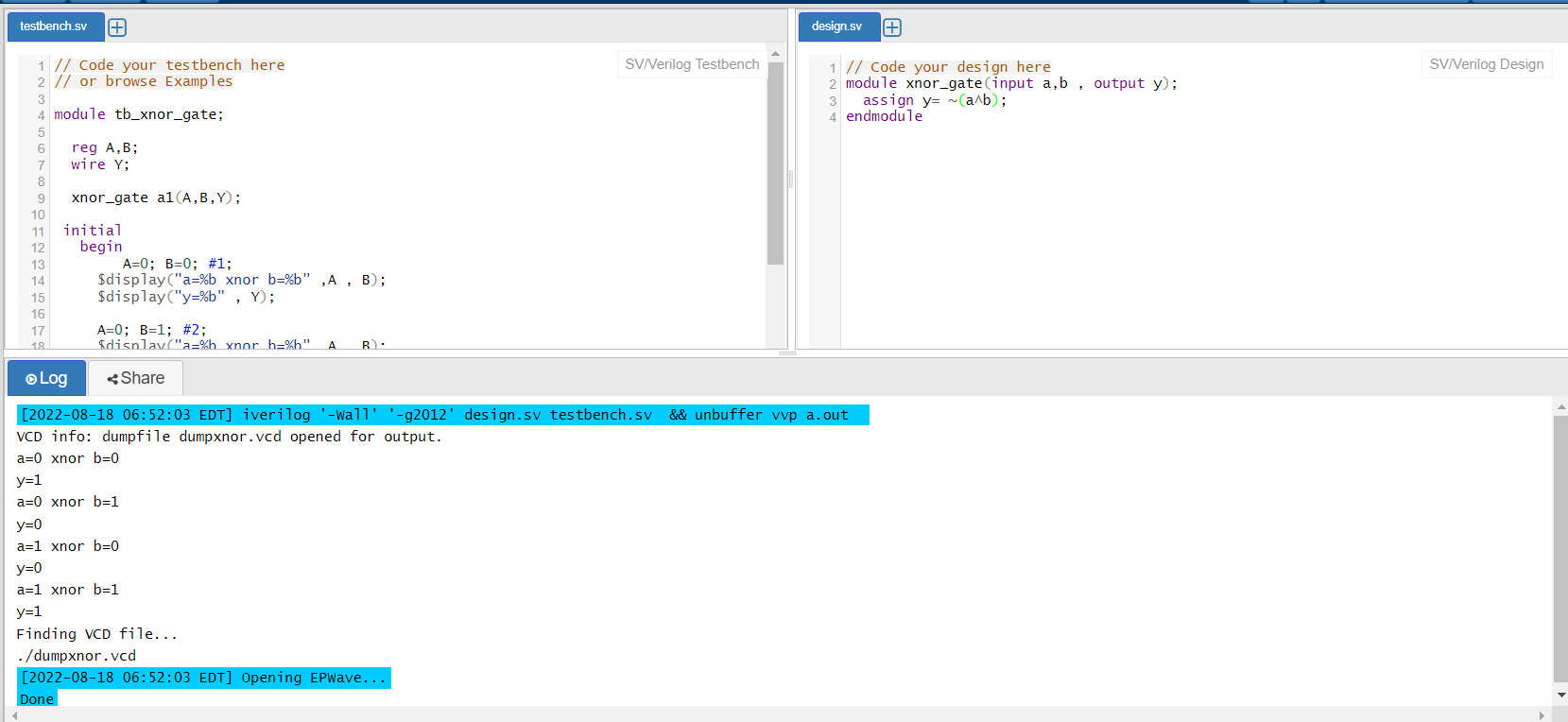


**Verification of XNOR Gate :**

Code :



Output:

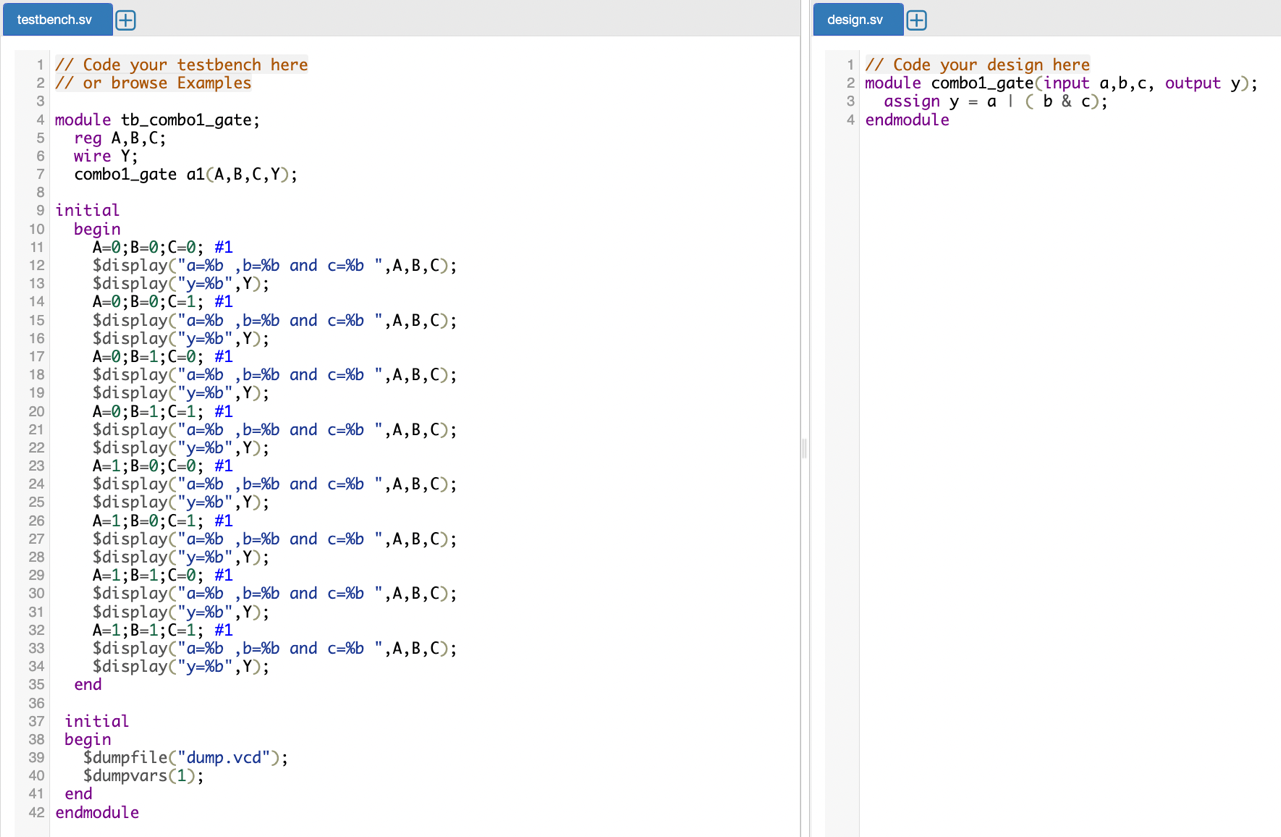




**Questions:**

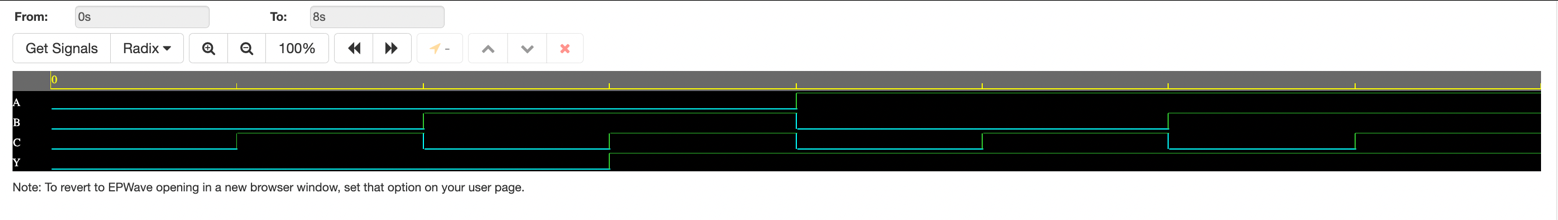
1. **A + (B . C)**

Code:



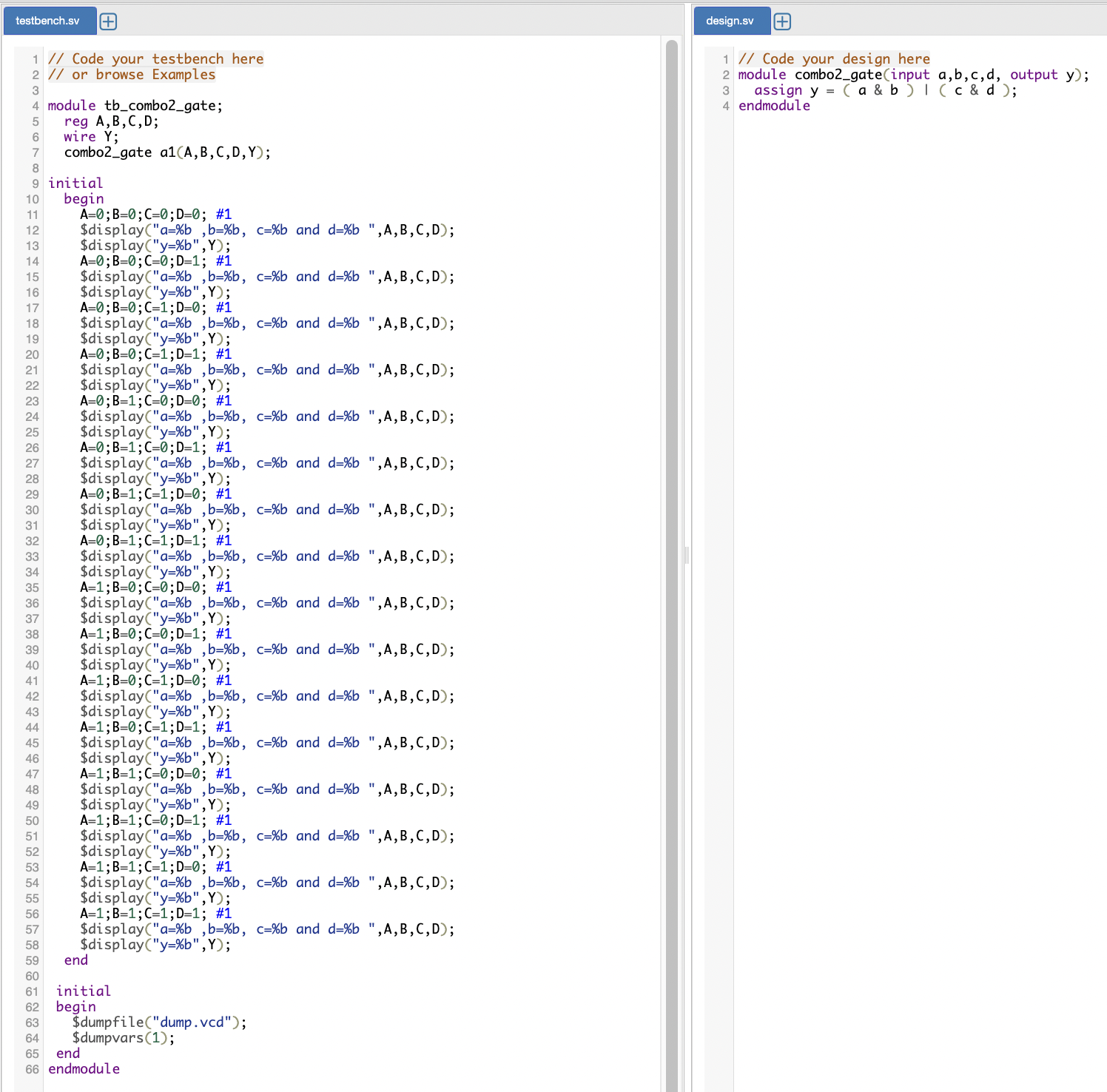
Output:



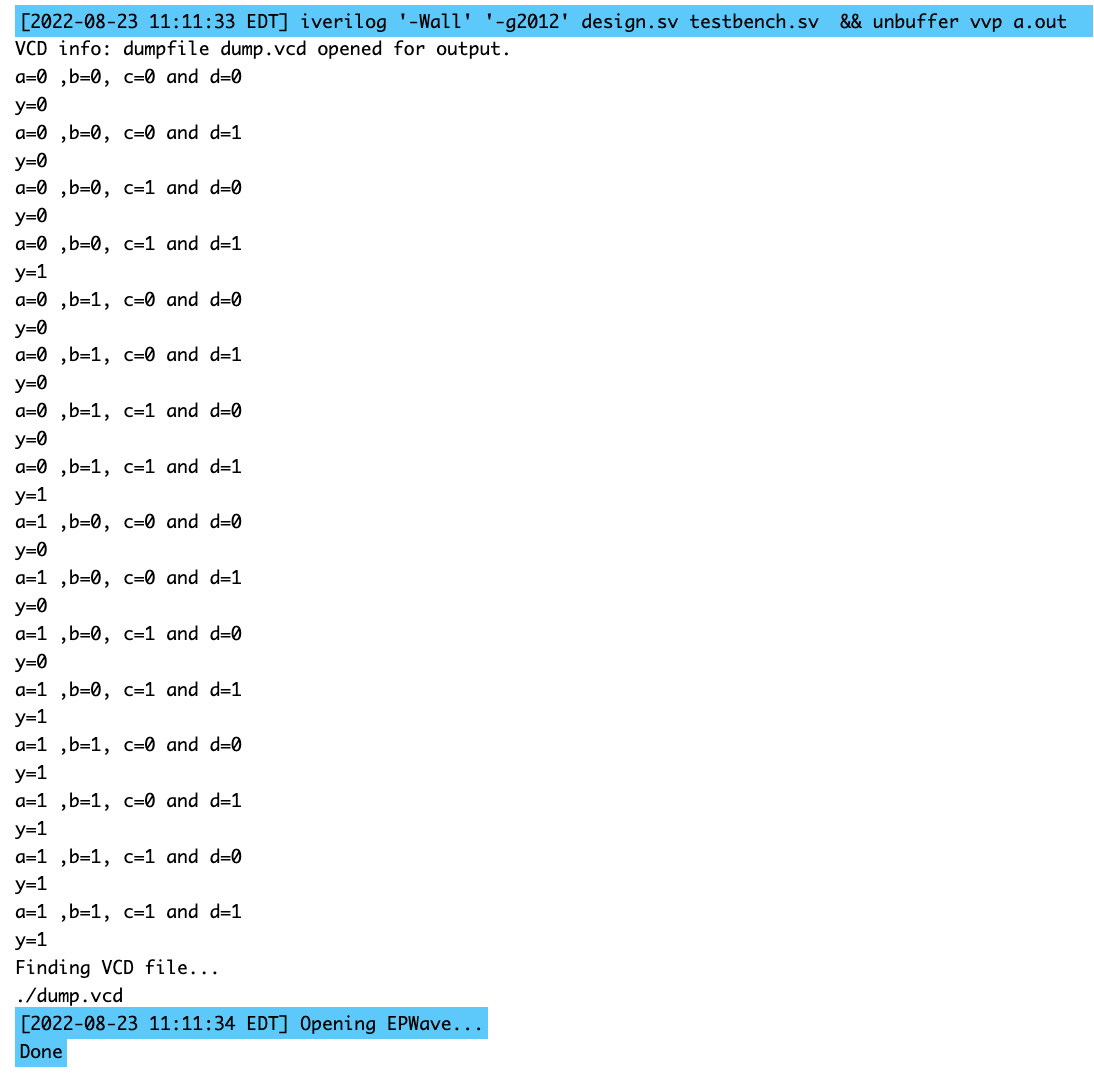


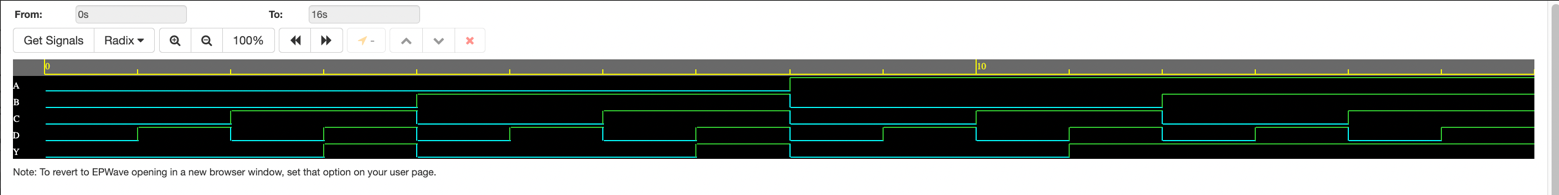
1. **(A . B) + (C . D)**

Code:



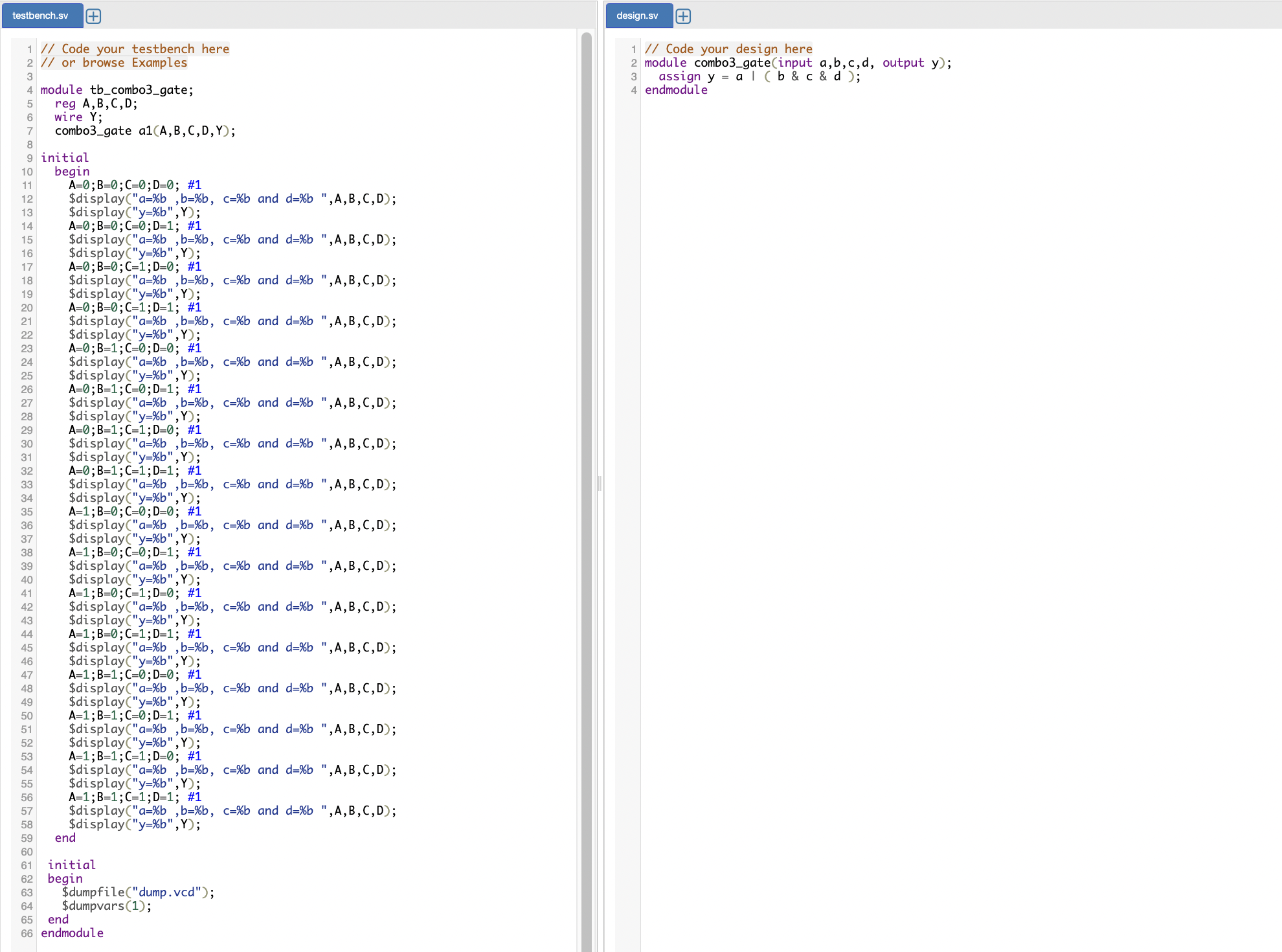
Output:





1. **A + (B . C . D)**

Code:



Output:

