100 Days of RTL

DAY 5

Universal Gates:

Switch Level Modelling

Achutha Aswin Naick

NAND Gate

Introduction

Switch-level modelling in Verilog represents circuits at the transistor level using predefined switch primitives such as NMOS and PMOS. This technique closely mirrors actual transistor-based hardware implementation. In this project, a NAND gate was implemented using switch-level modelling in Verilog. A corresponding testbench was developed to simulate the circuit and verify its correctness.

Implementation Details

The NAND gate module, univ_nand, consists of two inputs (a and b) and a single output (out). The design uses supply0 (gnd) and supply1 (vdd) to represent logic low (0) and logic high (1), respectively. The circuit is defined using the built-in Verilog switch primitives nmos and pmos, which simulate the behavior of NMOS and PMOS transistors:

- nmos(someV, gnd, a); and nmos(out, someV, b);: The NMOS transistors are arranged in series to pull out low only when both a and b are high.
- pmos(out, vdd, a); and pmos(out, vdd, b);: The PMOS transistors are arranged in parallel to pull out high unless both a and b are high.

This configuration implements the NAND logic function.

Code Explanation

The implementation consists of two Verilog modules: univ_nand (design module) and nand_tb (testbench module). Below is a breakdown of their components and functionality.

NAND Gate Module (univ_nand)

The univ_nand module defines a simple NAND gate using NMOS and PMOS transistors.

- **Input and Output Declaration:**
  - input a, b; defines two input signals.
  - output out; defines the output signal.
  - The out signal is of type wire since it is directly connected to the transistor gates.
- **Supply Nets:**
  - supply0 gnd; assigns a constant logic 0 to gnd.
  - supply1 vdd; assigns a constant logic 1 to vdd.
- **Intermediate Wire:**
  - wire someV; is used as an intermediate connection between the NMOS transistors.
- **NMOS and PMOS Primitives:**
  - nmos(someV, gnd, a); and nmos(out, someV, b);: The NMOS transistors are arranged in series, meaning out is pulled to gnd only when both a and b are high.
  - pmos(out, vdd, a); and pmos(out, vdd, b);: The PMOS transistors are arranged in parallel, meaning out is pulled to vdd unless both a and b are high.

This combination results in a NAND gate function, where out is low only when both a and b are high.

Testbench Module (nand_tb)

The nand_tb module is used to verify the functionality of the univ_nand module.

- **Input Declaration:**
  - reg Vin1, Vin2; defines Vin1 and Vin2 as reg types, allowing them to be assigned values in the testbench.
- **Output Declaration:**

    - wire Vout; defines Vout as a wire, since it is driven by the DUT (Device Under Test).
- **Instantiation of DUT:**
    - univ_nand DUT(.a(Vin1), .b(Vin2), .out(Vout)); creates an instance of the univ_nand module, connecting its inputs and output to the testbench signals.
- **Simulation Execution:**
    - initial Vin1 = 1'b0; sets the initial value of Vin1 to 0.
    - initial Vin2 = 1'b0; sets the initial value of Vin2 to 0.
    - initial forever #10 Vin1 = ~Vin1; toggles Vin1 every 10 time units, generating a square wave input.
    - initial forever #20 Vin2 = ~Vin2; toggles Vin2 every 20 time units, generating a slower square wave input.
    - initial #200 $finish; stops the simulation after 200 time units.

Simulation and Results

- initial Vin1 = 1'b0; sets the initial value of Vin1 to 0.
- initial Vin2 = 1'b0; sets the initial value of Vin2 to 0.
- initial forever #10 Vin1 = ~Vin1; toggles Vin1 every 10 time units, generating a square wave input.
- initial forever #20 Vin2 = ~Vin2; toggles Vin2 every 20 time units, generating a slower square wave input.
- initial #200 $finish; stops the simulation after 200 time units.

Conclusion

The NAND gate module successfully demonstrates switch-level modelling using NMOS and PMOS transistors. The testbench verifies the design by simulating input changes and observing expected NAND behaviour.

Future Enhancements

- Implementing more complex transistor-level circuits, such as full adders and multiplexers.
- Simulating real-world effects such as propagation delay and power consumption.
- Extending the design to model other logic gates using transistor-level modelling.

NOR Gate

Introduction

Switch-level modelling in Verilog represents circuits at the transistor level using predefined switch primitives such as NMOS and PMOS. This technique closely mirrors actual transistor-based hardware implementation. In this project, a NOR gate was implemented using switch-level modelling in Verilog. A corresponding testbench was developed to simulate the circuit and verify its correctness.

Implementation Details

The NOR gate module, univ_nor, consists of two inputs (a and b) and a single output (out). The design uses supply0 (gnd) and supply1 (vdd) to represent logic low (0) and logic high (1), respectively. The circuit is defined using the built-in Verilog switch primitives nmos and pmos, which simulate the behavior of NMOS and PMOS transistors:

- nmos(out, gnd, a); and nmos(out, gnd, b);: The NMOS transistors are arranged in parallel to pull out low when either a or b is high.
- pmos(someV, vdd, a); and pmos(out, someV, b);: The PMOS transistors are arranged in series to pull out high only when both a and b are low. This configuration implements the NOR logic function.

This configuration implements the NOR logic function.

Code Explanation

The implementation consists of two Verilog modules: univ_nor (design module) and nor_tb (testbench module). Below is a breakdown of their components and functionality.

NOR Gate Module (univ_nor) The univ_nor module defines a simple NOR gate using NMOS and PMOS transistors.

- **Input and Output Declaration:**
  - input a, b; defines two input signals.
  - output out; defines the output signal.
  - The out signal is of type wire since it is directly connected to the transistor gates.
- **Supply Nets:**
  - supply0 gnd; assigns a constant logic 0 to gnd.
  - supply1 vdd; assigns a constant logic 1 to vdd.
- **Intermediate Wire:**
  - wire someV; is used as an intermediate connection between the PMOS transistors.
- **NMOS and PMOS Primitives:**
  - nmos(out, gnd, a); and nmos(out, gnd, b);: The NMOS transistors are arranged in parallel, meaning out is pulled to gnd when either a or b is high.
  - pmos(someV, vdd, a); and pmos(out, someV, b);: The PMOS transistors are arranged in series, meaning out is pulled to vdd only when both a and b are low. This combination results in a NOR gate function, where out is high only when both a and b are low.

Testbench Module (nor_tb) The nor_tb module is used to verify the functionality of the univ_nor module.

- **Input Declaration:**
  - reg Vin1, Vin2; defines Vin1 and Vin2 as reg types, allowing them to be assigned values in the testbench.

- **Output Declaration:**
    - wire Vout; defines Vout as a wire, since it is driven by the DUT (Device Under Test).

- **Instantiation of DUT:**
    - univ_nor DUT(.a(Vin1), .b(Vin2), .out(Vout)); creates an instance of the univ_nor module, connecting its inputs and output to the testbench signals.
- **Simulation Execution:**
    - initial Vin1 = 1'b0; sets the initial value of Vin1 to 0.
    - initial Vin2 = 1'b0; sets the initial value of Vin2 to 0.
    - initial forever #10 Vin1 = ~Vin1; toggles Vin1 every 10 time units, generating a square wave input.
    - initial forever #20 Vin2 = ~Vin2; toggles Vin2 every 20 time units, generating a slower square wave input.
    - initial #200 $finish; stops the simulation after 200 time units.

Simulation and Results

- initial Vin1 = 1'b0; sets the initial value of Vin1 to 0.
- initial Vin2 = 1'b0; sets the initial value of Vin2 to 0.
- initial forever #10 Vin1 = ~Vin1; toggles Vin1 every 10 time units, generating a square wave input.
- initial forever #20 Vin2 = ~Vin2; toggles Vin2 every 20 time units, generating a slower square wave input.
- initial #200 $finish; stops the simulation after 200 time units.

Conclusion

The NOR gate module successfully demonstrates switch-level modelling using NMOS and PMOS transistors. The testbench verifies the design by simulating input changes and observing expected NOR behaviour.

Future Enhancements

- Implementing more complex transistor-level circuits, such as full adders and multiplexers.
- Simulating real-world effects such as propagation delay and power consumption.
- Extending the design to model other logic gates using transistor-level modelling.