**1. Implement a 2:1 multiplexer using assign and case.**

**//using case**

module mux2x1\_case(

input a,b,s0,

output reg y

);

always@(\*)begin

case(s0)

0: y = a;

1: y = b;

endcase

end

endmodule

**//using assign**

module mux2x1\_assign(

input a,b,s0,

output y

);

assign y = (s0 == 0)? a : b;

end module

**//testbench**

module tb\_mux2x1\_case;

// Declare testbench signals with different names

reg in1, in2, sel;

wire out\_mux;

// Instantiate the module under test (MUX) with custom signal names

mux2x1\_case uut (

.a(in1), // Connect 'a' in module to 'in1' in testbench

.b(in2), // Connect 'b' in module to 'in2' in testbench

.s0(sel), // Connect 's0' in module to 'sel' in testbench

.y(out\_mux) // Connect 'y' in module to 'out\_mux' in testbench

);

// Test stimulus

initial begin

$display("Time\t in1 in2 sel | out\_mux");

$monitor("%g\t %b %b %b | %b", $time, in1, in2, sel, out\_mux);

// Apply test cases

in1 = 0; in2 = 0; sel = 0; #10;

in1 = 0; in2 = 1; sel = 0; #10;

in1 = 1; in2 = 0; sel = 0; #10;

in1 = 1; in2 = 1; sel = 0; #10;

in1 = 0; in2 = 0; sel = 1; #10;

in1 = 0; in2 = 1; sel = 1; #10;

in1 = 1; in2 = 0; sel = 1; #10;

in1 = 1; in2 = 1; sel = 1; #10;

// End simulation

$finish;

end

endmodule

**Simulation Result:**



**KEY NOTES:**

1. **initial begin ... end**

initial begin

$display("Time\t in1 in2 sel | out\_mux");

$monitor("%g\t %b %b %b | %b", $time, in1, in2, sel, out\_mux);

end

* The initial block **executes only once** at the **beginning of the simulation**.
* The **testbench stimulus (input changes and monitoring)** is placed inside this block.

**2. $display(...)**

$display("Time\t in1 in2 sel | out\_mux");

* **Purpose**: Prints a **header row** for better readability of simulation output.
* **\t (Tab)**: Inserts spaces to align columns neatly.

**3. $monitor(...)**

$monitor("%g\t %b %b %b | %b", $time, in1, in2, sel, out\_mux);

* **Purpose**: Automatically prints the values of specified variables **whenever they change**.
* **Format Specifiers:**
  + %g → Prints **simulation time ($time)** in generic format.
  + %b → Prints **binary values** of in1, in2, sel, and out\_mux.
* **How $monitor Works:**
  + Unlike $display, which prints **once**, $monitor updates **whenever a monitored variable changes**.

**Key Differences Between $display and $monitor**

| **Feature** | **$display** | **$monitor** |
| --- | --- | --- |
| **When it executes** | Executes once when encountered | Runs whenever a variable changes |
| **Printing behavior** | Prints only once per call | Automatically updates values |
| **Use case** | Headers, debug messages, one-time info | Continuous signal tracking |

**Why reg?**

**1. assign Statement (Combinational Logic)**

* The assign statement is used for **continuous assignment**.
* It is used to drive **wire** type signals.
* It is typically used for **pure combinational circuits**.

🔹 **Why no reg?**

* assign works continuously and does not require a clock or sensitivity list.
* It directly drives a **wire** without the need for procedural assignments.

**2. always Block (Procedural Logic)**

* The always block is used for **procedural assignments**.
* It requires **reg** type variables for storing values.
* It is useful for **combinational logic (always @(\*))** and **sequential logic (always @(posedge clk))**.

🔹 **Why use reg?**

* In **procedural assignments** inside an always block, you must use reg because:
  + The value of y is **updated based on conditions**.
  + The value is stored and **not continuously driven** (unlike assign).
  + The assignment occurs **inside a procedural block** (always).

**Summary of assign vs. always**

| **Feature** | **assign (Continuous)** | **always Block (Procedural)** |
| --- | --- | --- |
| **Used for** | Combinational logic | Combinational or Sequential |
| **Drives** | wire | reg |
| **When it executes** | Always (continuous assignment) | When triggered by sensitivity list |
| **Example** | assign y = a & b; | always @(\*) y = a & b; |

**Key Takeaways**

✅ Use assign for simple **continuous assignments** (combinational logic).  
✅ Use always with reg when dealing with **procedural logic** (such as case or if-else).  
✅ **reg does NOT mean it's a register!** It just means the variable is assigned inside an always block.