DAY 61 - 111 DAYS VERIFICATION CHALLENGE

Topic: SV Data types

Skill: System Verilog

DAY 61 CHALLENGE:

1. Explain the difference between data types: logic and reg and wire.

wire:

- A wire represents a physical connection between components in Verilog. It is used to model combinational logic.
- wire cannot store values; it only drives values from one place to another.
- You can assign values to wire using continuous assignments (e.g., assign statement).

```
wire a, b, c;
assign c = a & b;
```

reg:

- A reg (short for register) is used to model storage elements, like flip-flops, and can hold values.
- reg can be used in procedural blocks (always, initial) to store values across clock cycles.
- It is typically used for sequential logic.

```
reg q;
always @(posedge clk) begin
    q <= d;
end
```

logic:

- logic is a data type introduced in SystemVerilog, and it can be used in place of both reg and wire.
- It is a 4-state variable (0, 1, X, Z), like reg, and can be driven by continuous assignments or used in procedural blocks.

- logic removes some ambiguity present in Verilog by allowing the same signal to be driven by both continuous and procedural assignments, though it's better to avoid such practices for clarity.

```
logic q;

always @(posedge clk) begin

    q <= d;

end
```

2. What is the difference b/w logic & bit.

- **logic:**
  - logic is a 4-state variable (0, 1, X, Z), allowing it to represent unknown (X) or high-impedance (Z) states.
  - It is typically used for signals that may experience these additional states.
- **bit:**
  - bit is a 2-state variable (only 0 or 1), which is more efficient in simulation because it doesn't need to represent X or Z.
  - It is used when only binary states are needed, such as in pure combinational logic.

```
logic [3:0] data4state; // Can be 0, 1, X, Z

bit [3:0] data2state;   // Can only be 0, 1
```

3. What are 2 state and 4 state variables? Explain with examples.

- **2-State Variables:**
  - 2-state variables can only represent 0 and 1.
  - Examples include bit and int in SystemVerilog.
  - These are used when there's no need to represent unknown (X) or high-impedance (Z) states, often in purely combinational logic.
- **4-State Variables:**
  - 4-state variables can represent 0, 1, X (unknown), and Z (high-impedance).
  - Examples include logic, reg, and wire in Verilog/SystemVerilog.
  - These are used in situations where the signal might be uninitialized (X) or not driven by any source (Z).

```
logic [3:0] four_state_var; // Can take values like 4'b0001, 4'bZZZZ, 4'bXXXX, etc.
```

```
bit [3:0] two_state_var;    // Can only take values like 4'b0001, 4'b1111
```

4. Difference between integer and int?

- **integer:**
  - integer is a 4-state variable (can hold 0, 1, X, Z).
  - It is 32 bits wide.
  - Mostly used in traditional Verilog for loop counters or simple integer arithmetic.
- **int:**
  - int is a 2-state variable (can only hold 0 and 1).
  - It is 32 bits wide by default but can be explicitly defined with a different width.
  - It's faster and more efficient in simulation compared to integer.

```
integer i = 32'hXXXX;  // 4-state integer
int j = 32'h7FFF_FFFF; // 2-state integer
```

5. How do you write a power b in SV code?

In SystemVerilog, We can compute the power of b using the ** operator.

```
int base = 2;
int exponent = 3;
int result = base ** exponent; // result = 8 (2^3)
```

6. What are pass-by-value and pass-by-reference methods?

Pass-by-Value:

- When a variable is passed by value, a copy of the variable's value is made, and the function or task works with that copy.
- Changes to the variable within the function do not affect the original variable.

```
task pass_by_value(int a);
    a = a + 1;
endtask
int x = 5;
pass_by_value(x);
// x is still 5 after the task execution
```

Pass-by-Reference:

- When a variable is passed by reference, the function or task works with the actual variable rather than a copy.
- Changes made to the variable within the function will affect the original variable.

```
task pass_by_reference(ref int a);
    a = a + 1;
endtask
int x = 5;
pass_by_reference(x);
// x is now 6 after the task execution
```

7. What are the types of arrays in System Verilog?

SystemVerilog supports various types of arrays, including:

1. **Packed Arrays:**
   - Used for representing vectors or groups of bits.
   - All elements are packed tightly, and bit-select and part-select operations can be performed.
   - Example: `logic [7:0] my_array;`
2. **Unpacked Arrays:**
   - Elements are not packed together; instead, each element is treated as an individual variable.
   - Can have multiple dimensions.
   - Example: `logic [7:0] my_unpacked_array [0:3];`
3. **Dynamic Arrays:**
   - Arrays that can be resized during runtime.
   - Size is determined dynamically, allowing flexibility.
   - Example: `int dynamic_array[];`
4. **Associative Arrays:**
   - Arrays indexed by a key, which can be an integer or string.
   - Useful when the index is sparse or not sequential.
   - Example: `int associative_array[string];`

5. **Queues:**
   - Ordered collections that allow insertion and removal of elements, similar to a FIFO structure.
   - Elements are stored sequentially, and their size can change dynamically.
   - Example: `int queue_array[$];`
6. **Fixed-size Arrays:**
   - Standard arrays with a fixed size defined at compile time.
   - Example: `int fixed_array[10];`

8. Explain the difference between Packed & Unpacked array with an example.

Packed Arrays:

- Packed arrays are tightly packed bit-wise, and each element of the array is aligned next to the other without gaps.
- They allow operations like bit-select and part-select across the entire array.
- Typically used when the array represents a vector or a collection of bits.

```
logic [7:0] packed_array; // 8-bit packed array (can be treated as a single 8-bit variable)
```

Unpacked Arrays:

- Unpacked arrays have individual elements that are not packed together bit-wise.
- Each element can be an entire data type, such as an integer or another array.
- They allow arrays of structures or other complex types.

```
logic [7:0] unpacked_array [0:3]; // Unpacked array of 4 elements, each 8-bits wide
```

Key Difference:

- **Packed arrays** are useful for creating bit-vectors or signals, while **unpacked arrays** are useful when you need to create more complex data structures like arrays of structs or arrays of arrays.

9. Explain Dynamic & Associative Arrays with examples.

Dynamic Arrays:

- Dynamic arrays can change size during runtime.
- They are useful when the array size cannot be determined at compile time.

```
int dynamic_array[];
```

```
initial begin
    dynamic_array = new[5]; // Allocate 5 elements
    dynamic_array[0] = 10;
    dynamic_array.delete(); // Deallocate array
end
```

Associative Arrays:

- Associative arrays use a key to index elements rather than a simple integer index.
- The key can be of any scalar data type (integer, string, etc.).

```
int associative_array[string];
initial begin
    associative_array["apple"] = 5;
    associative_array["banana"] = 10;
end
```

10.Difference between Associative and dynamic arrays?

Dynamic Arrays:

- Indexed by integers starting from 0.
- The size can be changed dynamically, but the index must be an integer.
- Useful when you know the array size may vary during runtime but you want to maintain an ordered structure.

Associative Arrays:

- Indexed by any data type (integer, string, etc.).
- Size is not fixed and elements are stored based on the key.
- Useful for sparse arrays or when the array needs to be indexed by non-integer values.

11. What is the procedure to assign elements in an array in System Verilog?

Assigning elements in SystemVerilog arrays can be done in multiple ways:

Static Assignment:

- Assign values during declaration.

```
int array[3] = '{1, 2, 3}; // Static array initialization
```

Index-based Assignment:

- Assign values to specific indexes.

```
int array[3];
initial begin
    array[0] = 1;
    array[1] = 2;
    array[2] = 3;
end
```

Assignment via Loop:

- Assign values using a loop.

```
int array[3];
initial begin
    for (int i = 0; i < 3; i++) begin
        array[i] = i + 1;
    end
end
```

Dynamic Assignment:

- Assign values to dynamic arrays.

```
int dynamic_array[];
initial begin
    dynamic_array = new[3];
    dynamic_array = '{1, 2, 3};
end
```

12.Explain Queues & it's methods with an example

Queues:

- A queue is a variable-size, ordered collection of homogeneous elements.
- Queues support several methods to manipulate the elements.

Queue Methods:

- $push_front(): Adds an element to the front of the queue.
- $push_back(): Adds an element to the back of the queue.
- $pop_front(): Removes and returns the front element.
- $pop_back(): Removes and returns the back element.

- $size(): Returns the number of elements in the queue.
- $delete(): Deletes the entire queue or a specified element.

```
int queue[$]; // Declaring a queue

initial begin

    queue.push_back(10);  // Adds 10 to the back of the queue

    queue.push_front(20); // Adds 20 to the front of the queue

    int front = queue.pop_front(); // Removes and returns 20

    int back = queue.pop_back();   // Removes and returns 10

    int size = queue.size();        // Returns the size of the queue
(should be 0)

end
```