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

* wire: Represents a physical connection between hardware elements, and its value is driven by another source (e.g., another module or an output). It cannot hold values, only pass them. It is used primarily for connecting different components in a design.
* reg: Represents a variable that holds a value (i.e., it stores state). It is used when modeling sequential logic. A reg variable is assigned a value in always blocks or procedural code, and it can retain that value until explicitly changed. In SystemVerilog, reg is not used as much, as logic is preferred, but it is still commonly seen in legacy Verilog code.
* logic: This is the most modern and flexible data type in SystemVerilog. It can represent both combinational and sequential logic and is used in place of reg and wire in most cases. It can hold a 0, 1, or unknown value (x or z) and can be used in both procedural and continuous assignments. It is often preferred because it combines the functionality of reg and wire.

1. What is the difference between logic and bit?

* logic: Represents a 4-state value in SystemVerilog, meaning it can take values 0, 1, unknown (x), or high-impedance (z). It is typically used in most hardware design and simulation.
* bit: Represents a 2-state value, which can only be 0 or 1. It cannot take the values x or z like logic can. It is used when you want to model a strict binary signal without unknown or high-impedance states.

1. What are 2-state and 4-state variables? Explain with an example.

* 2-state variables: These variables can only have two possible values: 0 and 1. In SystemVerilog, bit is an example of a 2-state variable. It does not support x (unknown) or z (high-impedance).
* 4-state variables: These variables can have four possible values: 0, 1, x (unknown), and z (high-impedance). logic and wire are examples of 4-state variables. This is useful when simulating hardware behavior where signals might be in an undefined state (x) or not driven (z).

1. Difference between integer and int?

* integer: In Verilog, an integer is a 32-bit signed number. It is a default data type for signed numbers in Verilog and SystemVerilog.
* int: In SystemVerilog, the int data type is the same as integer, i.e., it is also a 32-bit signed number. However, int was introduced to align with other SystemVerilog types and improve consistency across different data types.

1. How do you write Power B in System Verilog?

To represent a power of b (where b is some value), you can use \*\* for exponentiation in SystemVerilog. For example, if you want to represent b raised to the power of 2: b \*\* 2

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

* Pass-by-value: When you pass an argument to a function or task, the function or task works on a copy of the argument. Any changes made inside the function/task do not affect the original value.
* Pass-by-reference: When you pass an argument by reference, the function/task can modify the original value of the argument, since it is operating directly on the memory location of the variable.

Example:

module test;

int a = 5;

initial begin

pass\_by\_value(a); // a remains 5

$display(a);

pass\_by\_reference(a); // a becomes 10

$display(a);

end

task pass\_by\_value(int b);

b = 10; // Only modifies b, not a

endtask

task pass\_by\_reference(input output int b);

b = 10; // Modifies the original value of a

endtask

endmodule

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

* Fixed-size arrays: The size of the array is defined at compile time and cannot be changed during simulation.
* Dynamic arrays: These arrays can be resized during simulation. The size is not fixed and can be changed using new.
* Associative arrays: These arrays are indexed by a key, which is not necessarily a contiguous integer, but can be any value (e.g., strings, integers).
* Queue arrays: These are dynamic arrays that allow adding or removing elements at the front or back, similar to a queue in a data structure.

1. Explain the difference between packed and un-packed arrays with an example.

* Packed arrays: These arrays are stored contiguously in memory. Each element of the array is packed together, meaning they take up contiguous bits. For example, a packed array can represent a bus of bits or a multi-bit vector.

Example:

logic [7:0] packed\_array [3:0]; // 4 elements, each 8 bits wide

* Unpacked arrays: These arrays are stored as individual elements, which may not be contiguous in memory. The elements of the array can be of any size and can be indexed by different values.

Example:

logic [7:0] unpacked\_array [3]; // Array of 4 elements, each 8 bits wide

1. Explain dynamic and associative arrays with examples.

* Dynamic Arrays: The size of a dynamic array can be determined during simulation using the new keyword. It is flexible and allows resizing during runtime.

Example:

int dynamic\_array[]; // Declaring a dynamic array

dynamic\_array = new[10]; // Allocating 10 elements

dynamic\_array[0] = 5; // Assigning a value

* Associative Arrays: Associative arrays use a unique key for indexing, and the key does not need to be contiguous or ordered.

Example:

int assoc\_array[string]; // Associative array indexed by a string

assoc\_array["apple"] = 10;

assoc\_array["banana"] = 20;

1. Difference between associative arrays and dynamic arrays.

* Associative arrays can be indexed by arbitrary data types (strings, integers, etc.), whereas dynamic arrays are indexed by integers and have contiguous elements.
* Associative arrays don't require pre-allocation of data whereas, dynamic arrays need to use new[] operator to explicitly allocate memory space.
* Associative arrays are uninitialized whereas, dynamic arrays are initialized to default values like 0/null.

1. What is the procedure for assigning elements in an array in System Verilog?

int arr[5]; // Declare an array

arr[0] = 1; // Assign value to index 0

arr[1] = 2; // Assign value to index 1

For dynamic arrays

int dynamic\_array[];

dynamic\_array = new[10]; // Allocate 10 elements

dynamic\_array[0] = 5; // Assign to element 0

1. Explain Queues & their methods with an example.

* A queue is a dynamic array where elements can be added and removed from either the front or the back. It is useful for modelling FIFO (First In, First Out) behaviours.
* Queue methods include:
  + push\_back() – Adds an element at the back of the queue.
  + push\_front() – Adds an element at the front.
  + pop\_back() – Removes an element from the back.
  + pop\_front() – Removes an element from the front.

Example:

module test;

logic [7:0] queue[$]; // Declare a queue of 8-bit elements

initial begin

queue.push\_back(8'b00000001); // Add element to the back

queue.push\_front(8'b00000010); // Add element to the front

$display("Front: %b, Back: %b", queue[0], queue[$-1]);

queue.pop\_front(); // Remove from the front

$display("Front after pop: %b", queue[0]);

end

endmodule