1. Explain event regions in System Verilog.

Event regions in SystemVerilog define a time period during which events are triggered or detected. An event region allows specifying a period over which an event is active or waiting to be triggered. This is useful for time-based synchronization, such as delayed events or ensuring certain actions happen after a specific time duration.

Event region syntax

* @ operator is used to specify an event region.
* A region starts with @ and ends when the condition is met or when the event is triggered.

Example:

event e;

initial begin

#10ns -> e; // Trigger the event after 10ns

End

initial begin

@e; // Wait for the event `e`

$display("Event triggered after 10ns!");

end

1. Mention the purpose of dividing time slots in System Verilog.

In SystemVerilog, dividing time slots (or simulation time) is important to ensure accurate modeling and synchronization in a design or simulation. SystemVerilog uses the concept of simulation time, and dividing this time into slots helps to precisely manage when certain operations occur. This is useful for modelling sequential and parallel behaviours in digital circuits. Time slots are used to:

* Synchronize different processes or components.
* Control the execution order and ensure events occur in the correct sequence.
* Model timing behaviour in simulations (e.g., clock cycles, delays).

1. What are modports?

Modports in SystemVerilog are used in interfaces to define direction and access control for signals. Modports define the read/write access (input, output, or bidirectional) to signals when an interface is connected to a module or block. Modports allow different roles (like master or slave) to interact with the interface. They help in creating more structured and readable interface connections.

Example:

interface my\_interface;

logic clk, reset;

logic [7:0] data;

modport slave (input clk, input reset, input data);

modport master (output clk, output reset, output data);

endinterface

1. What is the use of modports?

* Modports define access permissions for signals in interfaces.
* They restrict how signals are accessed (e.g., read-only or write-only) by different components (like master or slave).
* Modports allow for clearer and more modular interface definitions, supporting multiple communication roles.

Example:

module slave\_module(input clk, input reset, input [7:0] data);

// Slave functionality

endmodule

module master\_module(output clk, output reset, output [7:0] data);

// Master functionality

endmodule

1. What is the use of $cast?

$cast in SystemVerilog is used to perform type casting. It is typically used for casting one object type to another (such as casting a base class pointer to a derived class pointer). It ensures type compatibility during object assignment or when dealing with polymorphic objects (base and derived classes).

Syntax:

class Base;

virtual function void print();

$display("Base class");

endfunction

endclass

class Derived extends Base;

function void print();

$display("Derived class");

endfunction

endclass

Base b;

Derived d;

b = d; // Allowed because of inheritance

$cast(b, d); // Performs cast from Derived to Base

1. What is a static variable?

A static variable in SystemVerilog is a variable that retains its value across function or task calls. It is initialized only once and preserves its state even after the function or task exits. Static variables are useful when you want to keep track of a value across multiple calls of the function or task.

Example:

function int counter();

static int count = 0; // Static variable

count = count + 1;

return count;

endfunction

1. What is a public declaration?

A public declaration allows variables, functions, or tasks to be accessible from outside the module, class, or package. It means that they are not private to the scope and can be referenced by other modules or classes. It is used to share common functions or variables with other modules or classes.

Example:

class MyClass;

public int x; // Public member variable

endclass

MyClass obj;

obj = new;

obj.x = 10; // Accessible because it's public

1. What is the use of local?

The local keyword is used to define variables that are only visible within the current block (such as within a begin/end block or fork/join). These variables are not accessible outside their scope. It is used to ensure that a variable is used temporarily and does not leak outside its intended scope.

Example:

initial begin

local int temp = 5; // Local to this block

$display("Temp: %0d", temp);

end

1. What is the use of package?

A package is a container in SystemVerilog for grouping related types, functions, tasks, and variables. Packages help to modularize and organize large designs. It is used to mprove code reusability, readability, and maintainability by organizing related components into logical units.

Example:

package my\_package;

int data = 10;

function void my\_func();

$display("Data: %0d", data);

endfunction

endpackage

To use the package:

import my\_package::\*;

my\_func(); // Call function from the package

1. What is the difference between [7:0] and byte?

* [7:0]: A bit vector of size 8, representing 8 individual bits. It is typically used to define arrays of bits.
* byte: A SystemVerilog predefined data type that represents an 8-bit signed value. It is an 8-bit storage unit, which can hold values from -128 to 127.

1. What is chandle in system verilog?

A chandle is a special data type used to represent handles to communication channels in SystemVerilog. It is commonly used in communication interfaces between different modules, especially for virtual interfaces. It is used to provide a reference or handle to a communication channel (like a mailbox or interface).

Example:

chandle h; // Declare a chandle

mailbox m = new();

h = m; // Assign a mailbox handle to the chandle

1. What is the difference between define and parameter?

* define: A preprocessor directive that defines a macro or constant. It is a simple substitution of text before compilation.
* define is replaced by the preprocessor and can't be changed during simulation.
* parameter: A compile-time constant used to define values that can be passed to modules or classes as parameters.
* parameter is used during module instantiation and can be overridden.

1. How automatic variables are useful in Threads?

Automatic variables are dynamically allocated on the stack each time the function or task is called. These variables are destroyed after the function or task completes. In multithreaded environments, automatic variables ensure that each thread gets its own copy of the variable, preventing race conditions.

Example:

task my\_task;

automatic int counter = 0; // New instance for each task call

counter = counter + 1;

$display("Counter: %0d", counter);

endtask

1. Write code to extract 5 elements at a time from a queue.

module extract\_elements;

queue int q;

initial begin

// Fill the queue

foreach (i in [1:10]) q.push\_back(i);

// Extract 5 elements at a time

int extracted[5];

foreach (i in [0:4]) q.dequeue(extracted[i]);

$display("Extracted elements: ");

foreach (i in [0:4]) $display(extracted[i]);

end

endmodule

1. How to check if any bit of the expression is X or Z?

logic [7:0] data = 8'b1x01z101;

if ($isunknown(data)) begin

$display("Data contains X or Z.");

end