1. What is a virtual function?

In SystemVerilog, a virtual function is a method in a class that can be overridden in a derived class. It allows polymorphism, meaning the method to be called is determined at runtime based on the type of the object, not the type of the reference used to call it.

A virtual function is defined with the virtual keyword, and derived classes can override it using the virtual keyword again (or redef if you want to explicitly redefine it).

Example:

class Base;

virtual function void display();

$display("Base display");

endfunction

endclass

class Derived extends Base;

// Overriding the virtual function

virtual function void display();

$display("Derived display");

endfunction

endclass

module test;

Base b;

Derived d;

initial begin

b = new;

d = new;

b.display(); // Output: Base display

b = d; // Polymorphism

b.display(); // Output: Derived display

end

endmodule

1. What is the use of a scope resolution operator?

Tthe scope resolution operator (::) is used to access members (variables, functions) from a specific scope or to refer to a class or method from a specific module or class. It is commonly used to access global variables, methods, or members within a specific class or namespace.

Example:

module top;

int x = 10;

initial begin

// Accessing global x using scope resolution

$display("Global x = %0d", top::x); // Output: Global x = 10

end

endmodule

1. Difference between virtual and pure virtual function.

* Virtual Function: In SystemVerilog, a virtual function is one that is defined in a base class and can be overridden in derived classes. It provides a default implementation but can be redefined in derived classes.
* Pure Virtual Function: A pure virtual function is declared with the pure keyword and does not have a body in the base class. The derived class must provide an implementation for this function. A class with pure virtual functions is an abstract class and cannot be instantiated directly.

Example:

class Base;

virtual function void display(); // Virtual function

$display("Base display");

endfunction

endclass

class Derived extends Base;

virtual function void display(); // Overriding virtual function

$display("Derived display");

endfunction

endclass

class AbstractClass;

pure virtual function void show(); // Pure virtual function

endclass

1. What is a virtual class?

A virtual class is essentially a base class that may contain virtual functions. The virtual class itself cannot be instantiated directly; it is meant to be subclassed, and its virtual methods are overridden by derived classes. You cannot instantiate a class with pure virtual methods, and classes with virtual methods can be used as base classes for inheritance.

virtual class Animal;

virtual function void speak(); // Pure virtual function

endfunction

endclass

class Dog extends Animal;

function void speak(); // Overriding pure virtual function

$display("Woof!");

endfunction

endclass

Example:

class Box #(type T = int); // Default parameter is of type int

T value;

function new(T val);

value = val;

endfunction

endclass

module test;

Box#(int) intBox;

Box#(string) stringBox;

initial begin

intBox = new(42);

stringBox = new("Hello");

$display("IntBox value: %0d", intBox.value); // Output: 42

$display("StringBox value: %s", stringBox.value); // Output: Hello

end

endmodule

1. What are parameterized classes?

Parameterized classes in SystemVerilog allow the creation of generic classes. These are classes that can operate on different types by specifying the type when the class is instantiated.

1. Difference between static and dynamic casting.

* Static Casting in SystemVerilog is performed at compile-time, and it's used when you know the type at compile time (like casting between types that are known to be compatible).
* Dynamic Casting is used when you need to cast between types that might be related in an inheritance hierarchy. It is done at runtime and checks the type of the object to ensure the cast is safe. In SystemVerilog, this can be done using $cast.

Example:

class Base;

virtual function void display();

$display("Base display");

endfunction

endclass

class Derived extends Base;

function void display();

$display("Derived display");

endfunction

endclass

module test;

Base b;

Derived d;

initial begin

b = new;

d = new;

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

b.display(); // Output: Derived display

end

endmodule

1. I don’t want to see the class properties of the base in a child. How do you do it?

In SystemVerilog, you can use protected or local access modifiers to control visibility and prevent access to base class members in derived classes.

* Protected members can be accessed only within the class and its derived classes.
* Local members are not accessible outside the class where they are defined.

Example:

class Base;

protected int x; // Protected member, accessible only in this class and its subclasses

function new();

x = 10;

endfunction

endclass

class Derived extends Base;

function display();

// x is accessible here as it's protected

$display("x = %0d", x);

endfunction

endclass

1. What are the default values of variables in a System Verilog constructor?

* For non-object types (like int, real, etc.), they are initialized to 0 (zero).
* For object handles (like class types), they are initialized to null (i.e., they don’t point to any object initially).
* For arrays and structs, the default value is set to a zero-initialized state for all elements.

Example:

class Test;

int x;

string str;

Test();

$display("x = %0d, str = %s", x, str); // Output: x = 0, str = ""

endfunction

endclass

module test;

Test t = new;

initial begin

// The constructor initializes the variables

end

endmodule

1. What are local and protected access qualifiers?

* Local: The local qualifier is used to declare variables or methods that are only accessible within the scope of the module, class, or program.
* Protected: The protected qualifier restricts access to members to only the class itself and its derived classes.

Example:

class Base;

local int x; // Only accessible within Base

protected int y; // Accessible in Base and derived classes

endclass

1. Explain with an example how to create arrays of class handles.

class Item;

int value;

function new(int val);

value = val;

endfunction

endclass

module test;

Item item\_arr[3]; // Array of 3 class handles

initial begin

item\_arr[0] = new(10);

item\_arr[1] = new(20);

item\_arr[2] = new(30);

foreach(item\_arr[i]) begin

$display("Item %0d value = %0d", i, item\_arr[i].value);

end

end

endmodule

1. What is the output?

class Sample\_Class;

int id;

extern function void display;

endclass

function void display;

$display(“id=%0d”,id);

endfunction

Output

id=0