DAY 68 - 111 DAYS VERIFICATION CHALLENGE

Topic : OOPs

Skill: System Verilog

DAY 68 CHALLENGE:

1. What is virtual function?

In SystemVerilog, a virtual function is a method declared in a base class that can be overridden by derived classes. It allows dynamic binding, meaning the function that is invoked depends on the runtime type of the object, not the compile-time type. Virtual functions enable polymorphism, allowing flexibility in how different derived classes can implement the same interface.

```
class Base;
    virtual function void display();
        $display("Base display");
    endfunction
endclass

class Derived extends Base;
    function void display();  // Override virtual function
        $display("Derived display");
    endfunction
endclasse
```

2. What is the use of scope resolution operator?

The scope resolution operator :: in SystemVerilog is used to access static members (variables, functions, tasks) of a class or to refer to a type within a class without needing an object. It's also used to disambiguate between global and class-local variables or methods.

```
class MyClass;
    static int value = 42;
endclass

initial begin
    $display("%0d", MyClass::value);   // Access static
member
end
```

3. Difference between virtual and pure virtual function

- **Virtual function**: A function that is defined in the base class and can be overridden in derived classes. It may or may not be overridden.
- **Pure virtual function**: A function that is declared but not defined in the base class, and **must be overridden** in any derived class. In SystemVerilog, a pure virtual function is declared by assigning it = 0 in the base class.

```
class Base;

    virtual function void display(); // Virtual function, can be
overridden

        $display("Base display");

    endfunction

    pure virtual function void pure_display(); // Pure virtual
function, must be overridden

endclass

class Derived extends Base;

    function void display();

        $display("Derived display");

    endfunction

    function void pure_display(); // Must implement pure virtual
function

        $display("Derived pure display");

    endfunction

endclass
```

4. What is virtual class?

A virtual class in SystemVerilog is similar to an abstract class in other OOP languages. It cannot be instantiated directly and is designed to be a base class for other derived classes. Virtual classes often contain pure virtual methods and are used to define common interfaces for derived classes to implement.

```
virtual class BaseClass;   // Virtual class, cannot be
instantiated
    pure virtual function void show();
endclass
```

5. What are parameterized classes?

Parameterized classes in SystemVerilog are similar to templates in C++. They allow classes to be written in a generic way where the data type or certain values can be passed as parameters. This enhances code reusability and flexibility.

```
class MyClass #(type T = int, int SIZE = 10);  // 
Parameterized class

    T data[SIZE];

    function void display();

        foreach (data[i])

            $display("data[%0d] = %0d", i, data[i]);

    endfunction

endclass

MyClass#(bit, 8) myObject;  // Instantiating with bit type and 
size
```

6. Difference between static and dynamic casting

- **Static casting**: Performed at **compile-time**. It is used when the type of the object is known at compile-time and is safe. SystemVerilog provides $cast() for static casting when the types are compatible.
- **Dynamic casting**: Performed at **runtime** and is used when the type of the object might not be known until runtime. In SystemVerilog, dynamic casting is also done using $cast(), but it returns a **status** (1 if successful, 0 otherwise). It is useful when casting to or from **derived classes** and the runtime type of the object is uncertain.

```
class Base;

endclass


class Derived extends Base;

endclass

Derived d;

Base b = new;

initial begin

    // Static cast (assumes Base can be statically cast to 
Derived)
```

```
    $cast(d, b);  // Will fail at runtime if b is not of type
Derived
    if ($cast(d, b)) begin
        $display("Dynamic cast successful");
    end else begin
        $display("Dynamic cast failed");
    end
end
```

7. I don't want to see class properties of base class in child. How do you do it?

To hide the base class properties from the child class, you can use the local or protected access qualifiers in SystemVerilog. If you declare the base class properties as local, they will not be accessible from the derived (child) class. If declared protected, they will be accessible only within the base class and its derived classes but not outside.

- **Local:** Variables or functions with local access will not be visible in derived classes.
- **Protected:** Variables or functions with protected access are visible in derived classes but cannot be accessed outside the class hierarchy.

```
class Base;
    local int id_local;       // Hidden from child
    protected int id_protected; // Accessible in child, but not
outside
    public int id_public;      // Visible everywhere

    function new();
        id_local = 1;
        id_protected = 2;
        id_public = 3;
    endfunction
endclass

class Derived extends Base;
    function void display();
        //  $display("id_local  =  %0d",  id_local);    //  Error:
id_local not accessible
```

```
        $display("id_protected   =   %0d",   id_protected);      //
Accessible
        $display("id_public = %0d", id_public);  // Accessible
    endfunction
endclass
```

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

In SystemVerilog, the default values of variables depend on their data type:

- Integer types (int, bit, logic, etc.): Default value is 0.
- Class handles: Default value is null.
- Strings: Default value is an empty string "".
- Real numbers: Default value is 0.0.

When an object is constructed without assigning explicit values, these default values are automatically assigned.

```
class MyClass;
    int id;         // Default is 0
    string name;    // Default is ""
    real value;     // Default is 0.0

    function new();
        // No need to initialize as they already have default values
    endfunction
endclass
```

9. What are local and protected access qualifiers?

- **Local**: A local access qualifier restricts visibility to **within the class** where it is declared. It cannot be accessed from derived classes or outside the class.
- **Protected**: A protected access qualifier allows access **within the class and its derived classes**. It restricts access from outside the class hierarchy, so the variable or method is visible only to the base class and its derived classes.

```
class Base;
    local int local_var;         // Accessible only within Base
```

```
    protected int protected_var; // Accessible within Base and its
children

    public int public_var;       // Accessible everywhere

    function new();

        local_var = 10;

        protected_var = 20;

        public_var = 30;

    endfunction

endclass

class Child extends Base;

    function void display();

        // $display("local_var = %0d", local_var);   // Error: Not
accessible

        $display("protected_var   =   %0d",   protected_var);      //
Accessible

        $display("public_var = %0d", public_var);  // Accessible

    endfunction

endclass
```

10. Explain with example how to create arrays of class handles.

We can create arrays of class handles in SystemVerilog by defining an array of the desired class type. You need to allocate memory for each element in the array, as class handles are initialized to null by default.

```
class MyClass;
    int id;


    function new(int id_val);
        id = id_val;
    endfunction
endclass


program main;
```

```
    MyClass my_array[5];  // Array of class handles (default is
null)
    int i;

    initial begin
        // Allocate memory for each class handle in the array
        for (i = 0; i < 5; i++) begin
            my_array[i] = new(i);
        end
        // Display the id values
        for (i = 0; i < 5; i++) begin
            $display("my_array[%0d].id = %0d", i, my_array[i].id);
        end
    end
endprogram
```

11. What is the output?

```
Class sample_class;
   Int id ;
Ectern function void display ;
End class

Function void display ;
   $display (“id = %0d”, id);
End function
```

OutPut : - id = 10

- The id variable is set to 10.
- When obj.display() is called, it prints the value of id, which is 10.