System Verilog
Object Oriented Programming Language
(OOPS)

Submitted To: Dr Rita Mahajan
Submitted By: Harshit Kaundal

OPPs

OOP stands for object-oriented programming. This type of programming gets its inspiration from real word objects where every object has its own property, characteristics defined by a blueprint. As this programming model is inspired by reads-life objects writing code becomes more easier as we can relate it to real objects.

Some key features of OOP are:

Inheritance

Polymorphism

Encapsulation

Data Abstraction

Inheritance

This is the property of OOP by virtue of which a class can inherit properties and behaviour of another class called parent or base class. The child or derived class can add more property or behaviour on the base class.

```
// Base class
class Animal;
    function void sound();
        $display("Animal makes a sound");
    endfunction
endclass

// Derived class
class Dog extends Animal;
    function void bark();
        $display("Dog barks");
    endfunction
endclass

// Test module
module inheritance_example;
    initial begin
        Dog d = new();
        d.sound(); // Inherited from Animal
        d.bark();  // Defined in Dog
    end
endmodule
```

```
Animal makes a sound
Dog barks
```

Encapsulation

This is a property by which we can bundle all the data and methods into one unit. This also helps in black boxing a unit where users can focus more on using the unit without knowing the underlying process or complexity.

```
class Dog;
    // Encapsulated/private data
    protected string name;
    protected int age;

    // Constructor
    function new(string n, int a);
        name = n;
        age = a;
    endfunction

    // Public method to access internal info
    function void introduce();
        $display("Woof! I'm %s and I'm %0d years old.", name, age);
    endfunction

    // Method to change age (internal logic is hidden)
    function void birthday();
        age++;
        $display("%s had a birthday! Now %0d years old.", name, age);
    endfunction
endclass

// Test Module
module test_dog_encapsulation;
    initial begin
        Dog d = new("Bruno", 3); // creating object (constructor hides interna
        d.introduce();           // controlled access to data
        d.birthday();            // data modified safely using method
        d.introduce();
    end
endmodule
```

```
Woof! I'm Bruno and I'm 3 years old.
Bruno had a birthday! Now 4 years old.
Woof! I'm Bruno and I'm 4 years old.
```

Polymorphism

Polymorphism is a concept in which same method can act differently in child class or when inputs are different. OOPs provide us two ways to enable polymorphism – function overloading and function overriding

```
// Base class Animal
class Animal;
    protected string name;
    function new(string n);
        name = n;
    endfunction
    function void introduce();
        $display("I am an animal named %s", name);
    endfunction
endclass

// Derived class Dog
class Dog extends Animal;
    function new(string n);
        super.new(n);
    endfunction
    function void introduce();
        $display("Woof! I am a dog named %s", name);
    endfunction
endclass

// Test Module
module test_polymorphism;
    initial begin
        Animal a = new("Animal");
        Dog d = new("Bruno");

        a.introduce();  // Base class method
        d.introduce();  // Derived class method (Dog)
    end
endmodule
```

```
I am an animal named Animal
Woof! I am a dog named Bruno
```

Data Abstraction

Data Abstraction means hiding unnecessary data and representing only what is necessary for the user basically that particular use case.

```
// Abstract class Animal
abstract class Animal;
    protected string name;

    // Abstract method to be implemented by derived classes
    virtual function void introduce();
    endfunction
endclass

// Derived class Dog
class Dog extends Animal;
    function new(string n);
        name = n;
    endfunction

    // Implementing the abstract method
    function void introduce();
        $display("Woof! I am a dog named %s", name);
    endfunction
endclass

// Test Module
module test_data_abstraction;
    initial begin
        Dog d = new("Bruno");
        d.introduce();  // Call implemented method
    end
endmodule
```

```
PS E:\react.js__proj\REDUX\File_foramt_converter> Woof! I am a dog named Bruno
```

OOPs features in System Verilog

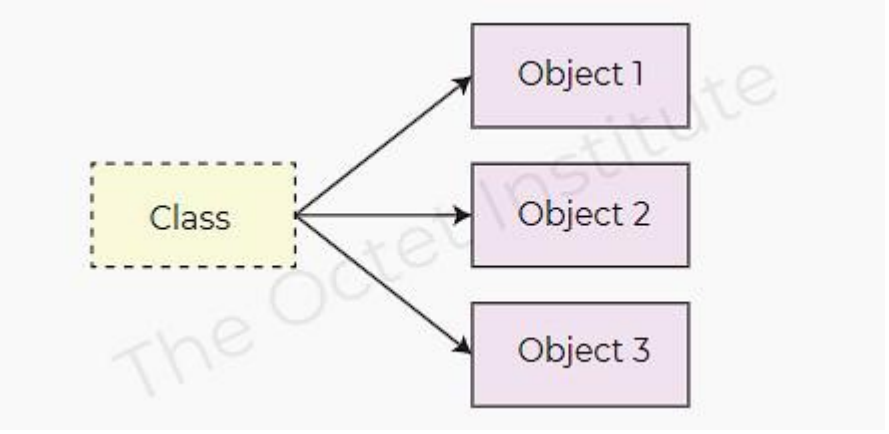
System Verilog is an HDL, i.e., hardware define language and thus all features of OOP are not needed in SV. Some of the complex OOP feature are omitted in the latest version of OOP. Some of the supported features are:

- Single & multi level inheritance
- Function overriding
- Virtual classes and pure virtual functions
- Virtual functions

Class and Objects

Class is a blueprint which defines the properties and behaviour of an object. In OOP classes are the entity which encapsulates all the data and methods.

Objects are the unique entity created from class. Objects are dynamic in nature, i.e, it is created dynamically during runtime.



Class :

```
class Person;
    // Class member variable
    string name;
    int age;

    // Constructor to initialize name and age
    function new(string name_input, int age_input);
        name = name_input;
        age = age_input;
    endfunction

    // Method to display person details
    function void display_details();
        $display("Name: %s, Age: %d", name, age);
    endfunction
endclass

// Testbench to instantiate the class and call the method
module test;
    Person p1;

    initial begin
        // Instantiate the class with name and age
        p1 = new("John", 25);
        p1.display_details();  // Should display "Name: John, Age: 25"
    end
endmodule
```

Structure :

```
// Define a structure with a function inside it
typedef struct {
    string name;
    int age;

    // Function to display person details
    function void display_details();
        $display("Name: %s", name);
        $display("Age: %d", age);
    endfunction
} Person;

// Testbench to use the structure and call the function
module test;
    // Declare a variable of type Person
    Person p1;

    initial begin
        // Initialize the structure fields
        p1.name = "Bob";
        p1.age = 40;

        // Call the function inside the structure to display the details
        p1.display_details();  // Displays Name: Bob, Age: 40
    end
endmodule
```

References:

- https://vlsiverify.com/systemverilog/
- https://www.chipverify.com/tutorials/systemverilog
- www.chatgpt.com

Thank You