

# 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 world objects where every object has its own property, characteristics defined by a blueprint. As this programming model is inspired by real-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.

```
1 // Base class
2 class Animal;
3     function void sound();
4         $display("Animal makes a sound");
5     endfunction
6 endclass
7
8 // Derived class
9 class Dog extends Animal;
10     function void bark();
11         $display("Dog barks");
12     endfunction
13 endclass
14
15 // Test module
16 module inheritance_example;
17     initial begin
18         Dog d = new();
19         d.sound(); // Inherited from Animal
20         d.bark();  // Defined in Dog
21     end
22 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.

```
1 class Dog;
2     // Encapsulated/private data
3     protected string name;
4     protected int age;
5
6     // Constructor
7     function new(string n, int a);
8     |   name = n;
9     |   age = a;
10    endfunction
11
12    // Public method to access internal info
13    function void introduce();
14    |   $display("Woof! I'm %s and I'm %0d years old.", name, age);
15    endfunction
16
17    // Method to change age (internal logic is hidden)
18    function void birthday();
19    |   age++;
20    |   $display("%s had a birthday! Now %0d years old.", name, age);
21    endfunction
22 endclass
23
24 // Test Module
25 module test_dog_encapsulation;
26     initial begin
27         Dog d = new("Bruno", 3); // creating object (constructor hides internal logic)
28         d.introduce();           // controlled access to data
29         d.birthday();            // data modified safely using method
30         d.introduce();
31     end
32 endmodule
33
```

```
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

```
1 // Base class Animal
2 class Animal;
3     protected string name;
4     function new(string n);
5         name = n;
6     endfunction
7     function void introduce();
8         $display("I am an animal named %s", name);
9     endfunction
10 endclass
11
12 // Derived class Dog
13 class Dog extends Animal;
14     function new(string n);
15         super.new(n);
16     endfunction
17     function void introduce();
18         $display("Woof! I am a dog named %s", name);
19     endfunction
20 endclass
21
22 // Test Module
23 module test_polymorphism;
24     initial begin
25         Animal a = new("Animal");
26         Dog d = new("Bruno");
27
28         a.introduce(); // Base class method
29         d.introduce(); // Derived class method (Dog)
30     end
31 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.

```
1  // Abstract class Animal
2  abstract class Animal;
3      protected string name;
4
5      // Abstract method to be implemented by derived classes
6      virtual function void introduce();
7      endfunction
8  endclass
9
10 // Derived class Dog
11 class Dog extends Animal;
12     function new(string n);
13         name = n;
14     endfunction
15
16     // Implementing the abstract method
17     function void introduce();
18         $display("Woof! I am a dog named %s", name);
19     endfunction
20 endclass
21
22 // Test Module
23 module test_data_abstraction;
24     initial begin
25         Dog d = new("Bruno");
26         d.introduce(); // Call implemented method
27     end
28 endmodule
29
```

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

## OOPs features in System Verilog

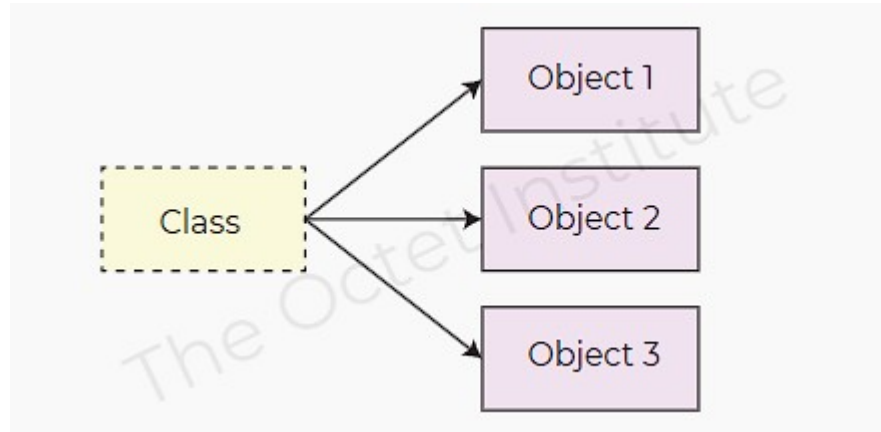
System Verilog is an HDL, i.e., hardware define language and thus all features of OOP are not needed in SV. Some of the complex OOP feature are omitted in the latest version of OOP. Some of the supported features are:

- 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 :

```
1 class Person;
2     // Class member variable
3     string name;
4     int age;
5
6     // Constructor to initialize name and age
7     function new(string name_input, int age_input);
8         name = name_input;
9         age = age_input;
10    endfunction
11
12    // Method to display person details
13    function void display_details();
14        $display("Name: %s, Age: %d", name, age);
15    endfunction
16 endclass
17
18 // Testbench to instantiate the class and call the method
19 module test;
20     Person p1;
21
22     initial begin
23         // Instantiate the class with name and age
24         p1 = new("John", 25);
25         p1.display_details(); // Should display "Name: John, Age: 25"
26     end
27 endmodule
```

## Structure :

```
1 // Define a structure with a function inside it
2 typedef struct {
3     string name;
4     int age;
5
6     // Function to display person details
7     function void display_details();
8     $display("Name: %s", name);
9     $display("Age: %d", age);
10    endfunction
11 } Person;
12
13 // Testbench to use the structure and call the function
14 module test;
15     // Declare a variable of type Person
16     Person p1;
17
18     initial begin
19         // Initialize the structure fields
20         p1.name = "Bob";
21         p1.age = 40;
22
23         // Call the function inside the structure to display the details
24         p1.display_details(); // Displays Name: Bob, Age: 40
25     end
26 endmodule
```

## References:

- <https://vlsiverify.com/systemverilog/>
- <https://www.chipverify.com/tutorials/systemverilog>
- [www.chatgpt.com](https://www.chatgpt.com)

**Thank You**