System Verilog
&
Concept of OOP

Introduction

- System Verilog introduces an object-oriented class data type.
- Classes allow objects to be dynamically created, deleted, assigned, and accessed via object handles.
- A class is a type that includes data and subroutines (functions and tasks) that operate on that data.
- class's data is referred to as class properties, and its subroutines are called methods.

Example1

```
class rectangle;
int lenght, width;              //class properties

function int area();            //class method
return lenght * width;
endfunction

function int perimeter();       //class method
return  2*(lenght + width);
endfunction
endclass
```

Example2

```
class person;
string name, address;                    //class properties
int number;

function void set_name(string user);     //class method
name=user;
endfunction
endclass
```

Example3

```
class packet;
bit [7:0] data;              //class property

task randomize();  //class method
data=$random;
endtask

task display();
$display("data is %d", data);
endtask
endclass
```

Objects

- A class defines a data type. An object is an instance of that class.

- An object is created by defining a handler of class type and then calling new function which allocates memory to the object.

```
packet p;          //p is handler to class packet
p=new();           //Object is constructed
```

- If objects are not created then handler points to null.

Default Constructor

- new() is a default constructor which allocates memory and initializes class variables for an object.

```
rectangle rec;

initial begin
rec=new;                //memory allocated to length and width
int a, p;
rec.set_size(3, 5);
a=rec.area;
p=rec.perimeter; end
```

Constructor

- User can define customized constructers by writing there own new function inside a class.
- The new method is defined as a function with no return type.
- It is also possible to pass arguments to the constructor, which allows run-time customization of an object.

```
function new (int x=0, y=0);
length=x;
width=y;
endfunction
```

Example

```
class rectangle;
int lenght, width;

function new(int
   x=1,y=1);.....
function int area(); .............
function int perimeter();
   ..........

endclass
```

Result: a1=15 p1=10

```
rectangle r1, r2, r3;
int a1, a3, p1;

initial begin
r1=new(3, 5);
r2=new(4);
a1=r1.area;
p1=r2.perimeter;
a3=r3.area;    //error r3 is null
end
```

Parameterized Class

```
class packet #(number=10, type dtype= bit);
dtype data [number];

function void randomize();
foreach(data[i]) data[i]=$random;
endfunction

function void display();
foreach(data[i]) $display("data[%0d"]=%0d", i, data[i]);
endfunction
endclass
```

Parameterized Class

```
packet p1;                          //number=10, dtype=bit
packet #(20) p2;                    //number=20, dtype=bit
packet #( , int) p3;                //number=10, dtype=int
packet #(30, bit [3:0]) p4;         //number=30, dtype=bit [3:0]

initial begin
p1=new();   p2=new();   p3=new();   p4=new();
p4.display;
p4.randomize;
p4.display;
end
```

This

- The this keyword is used to unambiguously refer to class properties or methods of the current instance.

- The this keyword shall only be used within non-static class methods, otherwise an error shall be issued.

```
class example;
int a;
function new(int a);
a=a;
endfunction
endclass
```

Now a is property of class as well as argument of function new.

SV will look in local scope to resolve reference to a, which in this case is subroutine argument.

This

- To solve this issue this keyword is used which now refers to property a in current class instance.

```
class example;
int a;
function new(int a);
this.a=a;
endfunction
endclass
```

```
example x, y;
initial begin
x=new(5);
y=new(3);
$display(x.a);
$display(y.a);
end
```

Assignment-7

- **Declare a class adder with following properties and inside module create adder using class properties only.(behavioral modelling).**
  - **2 inputs of integer type: A1, A2**
  - **1 input of logic type: en**
  - **1 output of integer type: sum**
- **In problem1 add the following methods & use these methods inside module.**
  - **To display all the properties on transcript**
  - **To perform addition**
  - **Randomize the input A1 and A2.**

- In problem1 make the following changes
  - Class should be parametrized (initially A1 , A2 ,sum are int type)
  - Create 4 instance of that class with A1,A2, sum of
    - Integer
    - Byte
    - Longint
    - String (just chk)

- Use THIS keyword in problem2 methods and use inside module.
- Reading assignments
    - Wire inside class
    - Function and task inside class
    - Destructor in SV
    - New( ) Vs new[ ]
    - Class Vs structure

Fundamental Principles of OOP

- Encapsulation
  - It's a concept that binds together the data and functions that manipulate the data.
  - Encapsulation keeps both data and function safe from outside world i.e. data hiding.

- Abstraction
  - Abstraction is the concept of moving the focus from the details and concrete implementation of things, to the types of things, the operations available thus making the programming simpler, more general.

Fundamental Principles of OOP

- Inheritance
  - New classes are created by inheriting properties and method defined in an existing class.
  - Existing class is called the base class(parent class), and the new class is referred to as the derived class(child class).

- Polymorphism
  - polymorphism means having many forms.
  - A member function will cause a different function to be executed depending on the type of object that invokes the function.

Inheritance

- Inheritance allows user to create classes which are derived from other classes.
- The derived class (child class) inherits all the properties and methods defined in base class (parent class).
- Additional properties and methods can be defined in child class.
- properties and methods defined in base class can be overridden by redefining them in child class. This phenomenon is called as overriding.

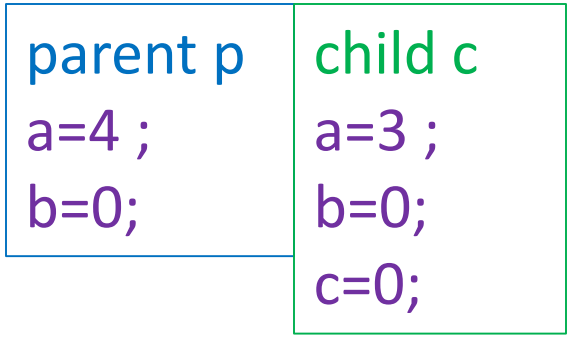
Example1

```
class parent;
int a, b;
task display();
$display("Parent Class");
endtask
endclass
```

```
class child extends parent;
int c;
task print();
$display("Child Class");
endtask
endclass
```

Example1

```
parent p;
child c;
initial begin
p=new;
c=new;
c.print;
c.display;
c.a=3;
p.a=4;
end
```



Child Class
Parent Class

Example2

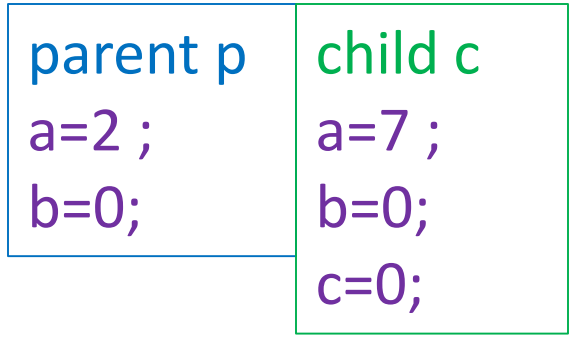
```
class parent;
int a, b;
task display();
$display("Parent Class");
endtask
endclass
```

```
class child extends parent;
int a;
task display();
$display("Child Class");
endtask
endclass
```

Display method and property can be overridden in child class

Example2

```
parent p;
child c;
initial begin
p=new;
c=new;
c.display;
p.display;
c.a=7;
p.a=2;
end
```



Child Class
Parent Class

Example3

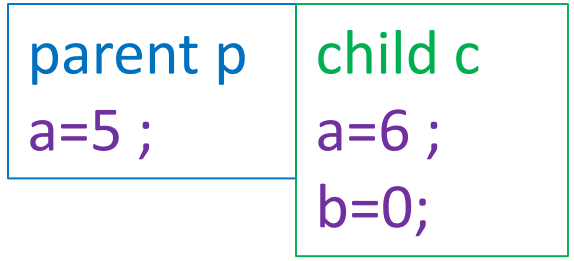
- A super keyword can be used to access properties and methods defined in parent class from a child class.

```
class parent;
int a;
task display();
$display("Parent Class");
endtask
endclass
```

```
class child extends parent;
int a, b;
task display();
$display("Child Class");
super.display;
$display(super.a);
endtask
endclass
```

Example3

```
parent p;
child c;
initial begin
p=new;
c=new;
p.a=5;
c.a=6;
p.display;
c.display;
end
```




```
Parent Class
Child Class
Parent Class
        0
```

Inheritance

- Every time when child object is created, constructer of parent (super.new) is called first implicitly.
- If a new function has been defined in parent which accepts a set of arguments and arguments don't have default values. In such a case super.new has to be explicitly specified with required arguments.
- It is because of this reason that child class is able to access properties and methods defined in parent class.

Example4

```
class parent;
function new();
$display("Parent Class");
endfunction
endclass
```

```
class child extends parent;
function new();
$display("Child Class");
endfunction
endclass
```

```
initial begin
child c;
c=new;
end
```

```
Parent Class
Child Class
```

Example5

```
class parent;
function new(string str);
$display(str);
endfunction
endclass

initial begin
child c;
c=new;
end
```

```
class child extends parent;
function new();
$display("Child Class");
endfunction
endclass
```

Error super.new is not called

Example6

```
class parent;
function new(string str=" ");
$display(str);
endfunction
endclass

initial begin
child c;
c=new;
end
```

```
class child extends parent;
function new();
$display("Child Class");
endfunction
endclass
```

Child Class

No error, parent constructor has default value

Example7

```
class parent;
function new(string str);
$display(str);
endfunction
endclass
```

```
class child extends parent;
function new();
super.new("Parent Class");
$display("Child Class");
endfunction
endclass
```

```
initial begin
child c;
c=new;
end
```

```
Parent Class
Child Class
```

Example8

```
class rectangle;
int length, width;

function new(int x, y);
this.length=x;
this.width=y;
endfunction
function int area(int x, y);....
function int perimeter(int x,
   y);...
endclass
```

```
class square extends rectangle;
int size;
function new(int size);
this.size=size;
super.new(size, size);
endfunction
endclass
```

```
square sq= new(5);
sq.area;
sq.perimeter;
```

Assignment-8

- **Create a class LOGIC_GATE with following properties and methods:**
    - **In1, in2: logic type , out: logic type**
    - **Function print with display statement LOGIC_GATE class**
  - **Now create two more classes AND_GATE , OR_GATE as child class extended from LOGIC_GATE with display methods for in1, in2, out also call print function.**
  - **In module create the objects of child classes and create functionality for AND & OR.**
- **Create a class PACKET with following properties and methods:**
    - **clk: logic type, addr :byte type, data: int type**
    - **Create new function to provide initial value as 1, 1001001,100**
    - **Create method for display and randomize.**
  - **Create a child class I2C_PACKET with same properties and methods.**
  - **check the functionality of super keyword by providing data to child class from parent class**
  - **In module create object of child class and store I2C_PACKET properties to the outputs of module at negative edge of clk.**

- **Create class A and class B without using inheritance try access properties and methods of class A with the object of class B.**

- # Practice problems:
- **How will you pass an object of a class to function?**
- **How will you return an object of class from function?**

- **How many types of inheritance supported by system Verilog?**
  - **Single**
  - **Multilevel**
  - **Hierarchical**