KACPER
designing future

Object Oriented Programming

SystemVerilog.in

Procedural Approach

KACPER
designing future

- ✓Focus is on procedures
- ✓All data is shared: no protection
- ✓More difficult to modify
- ✓Hard to manage complexity

Introduction to OOP

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of one or more hierarchy of classes united via inheritance relationships

Basic OOP

- ✓ OOP is object oriented programming
- ✓ Classes form the base of OOP programming
- ✓ Encapsulation - OOP binds data & function together
- ✓ Inheritance –extend the functionality of existing objects
- ✓ Polymorphism – wait until runtime to bind data with functions
- ✓ OOPs breaks a testbench into blocks that work together to accomplish the verification goal

OOP cont..

KACPER designing future

✓Why OOP?

- Highly abstract system level modelling
- Classes are intended for verification
- Classes are easily reused and extended
- Data security
- Classes are dynamic in nature
- Easy debugging, one class at a time
- Redundant code can be avoided by using inheritance
- Inheritance reduces time and cost
- Easy upgrading of system is possible using object oriented system

Class

- ✓ A class is a user-defined data type.
- ✓ Classes consist of data (called *properties*) and tasks and functions to access the data (called *methods*).
- ✓ Classes are used in object-oriented programming.
- ✓ In SystemVerilog, classes support the following aspects of object-orientation – encapsulation, data hiding, inheritance and polymorphism

Class Declaration

```
class register;
        local bit[31:0] contents;
        function void write(bit[31:0] d)
          contents = d;
        endfunction

        function bit[31:0] read();
          return contents;
        endfunction
endclass
```

Objects

- ✓ Organized into groups with similar characteristics
  - They are said to be related by a common characteristic
- ✓ Object-Oriented programming seeks to provide mechanisms for modelling these relationships

Objects as instances of Classes

KACPER designing future

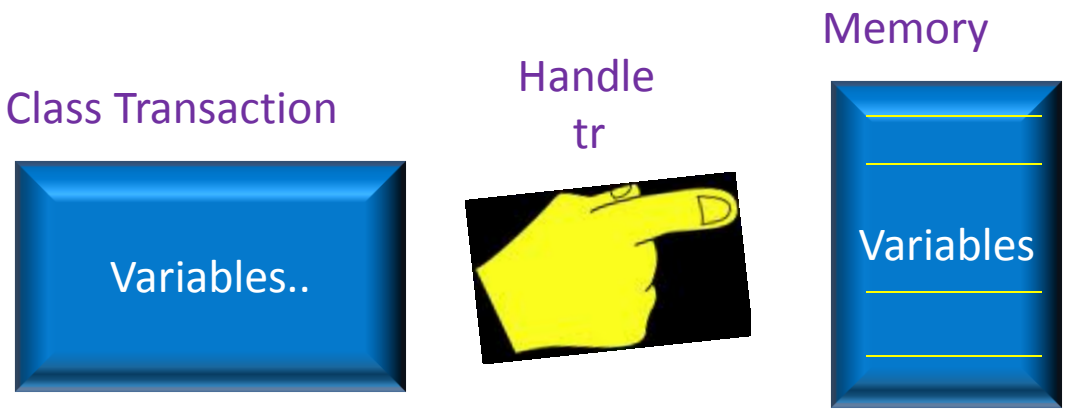
- ✓ The world conceptually consists of objects
- ✓ Many objects can be said to be of the same type or class
  - My bank account, your bank account, Bill Gates' bank account …
- ✓ We call the object type a class
- ✓ Instantiation
  - An Object is instantiated from a Class

```
BankAccount myAccount;  //creating a handle for
                        //the class BankAccount

myAccount = new ();//creating object
```

What is Handle?

- ✓ It is a pointer to an object.
- ✓ An oop handle is like the address of the object but is stored in a pointer that only refer to one type.

```
Ex:
  class Transaction;
     logic addr,data;
  endclass

  initial begin
    Transaction tr;
    tr=new();
  end
```



Class Example:

KACPER
designing future

```
class register;
        local bit[31:0] contents;
        function void write(bit[31:0] d)
          contents = d;
        endfunction

        function bit[31:0] read();
          return contents;
        endfunction
endclass
```

```
module top;
    register r;
    bit[31:0] d;
    initial begin
        r = new();
    r.write(32'h00ff72a8);
        d = r.read();
     end
endmodule
```

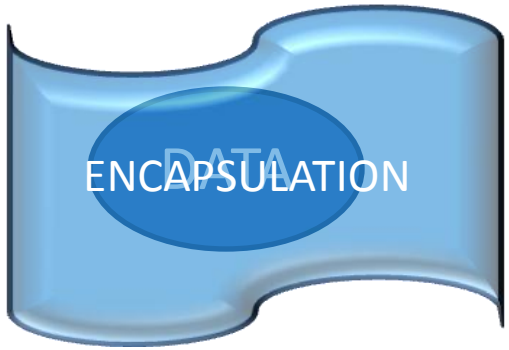
Class Encapsulation

- ✓ A class encapsulation is a process of hiding all internal details of an object from out side world.

- ✓ Encapsulation is the ability to hide data and methods from out side world and only expose data and methods that are required.

- ✓ Advantage of encapsulation

  - Protection
  - Consistency
  - Allows change



Encapsulation example:

KACPER designing future

```
class base;
    local int i;
endclass

program main;
    initial
    begin
        base b = new();
        b.i = 123;
    end
endprogram
```

Result:
Local variable error

Polymorphism

- ✓ Polymorphism refers to the ability to assume different forms. In OOP, it indicates a language's ability to handle objects differently based on their runtime type.
- ✓ When objects communicate with one another, we say that they send and receive messages.
- ✓ The sending object only needs to be aware that the receiving object can perform a particular behaviour.

Example:

KACPER designing future

```
class A ;
    virtual task disp ();
        $display(" This is class A ");
    endtask
endclass

class EA extends A ;
     task disp ();
        $display(" This is Extended class A ");
     endtask
endclass

program main ;
    EA my_ea;
    A my_a;

    initial
    begin
        my_a = new();
        my_a.disp();

        my_ea = new();
        my_a = my_ea;
        my_a.disp();
    end
endprogram
```

RESULTS
This is class A
This is Extended class A

INHERITANCE

- ✓ Definition (Inheritance) Inheritance is the mechanism which allows a class B to inherit properties of a class A.
- ✓ We say “ B inherits from A”.
- ✓ Objects of class B thus have access to attributes and methods of class A without the need to redefine them.
- ✓ The following definition defines two terms with which we are able to refer to participating classes when they use inheritance.

INHERITANCE (Cont...)

- ✓ The idea of inheritance is simple but powerful:
- ✓ When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class.
- ✓ In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

Example:

KACPER
designing future

```
class parent;
      task printf();
         $display(" THIS IS PARENT CLASS ");
      endtask
  endclass

  class subclass extends parent;
     task printf();
         $display(" THIS IS SUBCLASS ");
     endtask
  endclass

  program main;

      initial
      begin
         parent p;
         subclass s;
         p = new();
         s = new();
         p.printf();
         s.printf();
      end
  endprogram
```

RESULTS

THIS IS PARENT CLASS A
THIS IS SUBCLASS

Why not C++?

Why SystemVerilog?

Why Not C++?

Comparison

C++

- ✓ No relation to verilog
- ✓ Interface is required to interact with Verilog

System Verilog

- ✓ Superset of Verilog
- ✓ RTL/Verification language
- ✓ Assertion language
- ✓ Constraint language
- ✓ Code coverage language