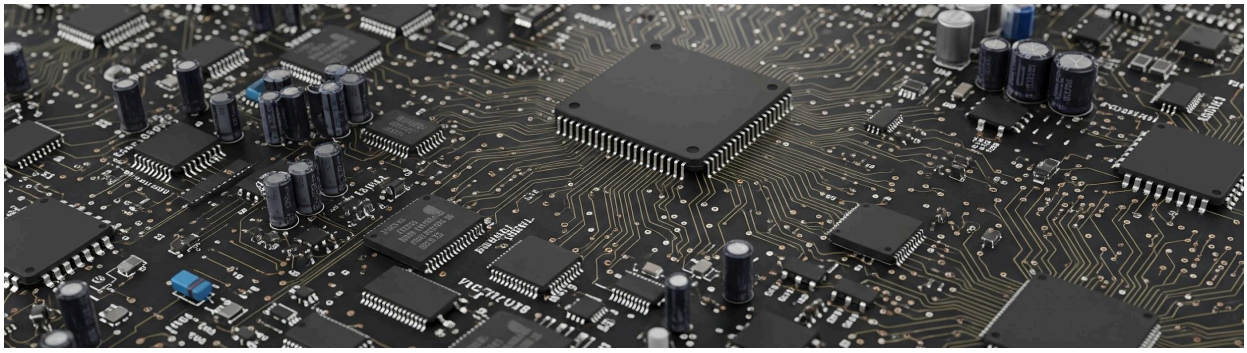

SystemVerilog Classes: Object-Oriented Programming for Verification

Welcome to the world of SystemVerilog Classes!

This guide explores the power of Object-Oriented Programming (OOP) in SystemVerilog for building robust and flexible verification environments.

The Core: What's a Class Anyway?

At its heart, a class is a blueprint. It's a user-defined type that bundles data (properties or members) and the operations (methods – tasks and functions) that act on that data.

- **Defining the Blueprint**: You define a class using the class keyword and close it with endclass.class Transaction;

  // Properties (data members) will go here

  // Methods (tasks/functions) will go here

  endclass

- **Infinite Reusability**: Once you define a class, you can create countless instances (objects) of that blueprint. Each object is independent and holds its own data.

- **Dynamic Nature**: Class objects are dynamic, created and destroyed during simulation runtime, offering flexibility compared to static Verilog modules.

Getting Hands-On: Creating and Managing Objects

Building something from the blueprint involves class handles and the new() operator.

- **The "Handle" Concept**: A class handle is a variable that refers to an object in memory.Transaction tr_h; // Declares a handle named 'tr_h' for a Transaction object.

    // At this point, 'tr_h' points to nothing (its value is 'null').

- **Allocating Memory: The new() Operator**: This operator allocates memory for a new class object on the heap, initializes the object's properties, and returns a handle to this object.tr_h = new(); // A new Transaction object is created in memory,

    // and 'tr_h' now points to it.

- **Accessing with the Dot Operator (.)**: Use the dot operator to access an object's properties or call its methods.tr_h.address = 'h100; // Assign a value to the 'address' property

tr_h.send(); // Call the 'send' method

- **Releasing Memory: null**: Set the handle to null (tr_h = null;) to break the connection. SystemVerilog's garbage collection reclaims the memory.

Methods: Tasks vs. Functions within a Class

Classes are about data and behavior, implemented through tasks and functions as methods.

- **task (The Time Traveler)**:
    - Timing Control: Use for operations involving simulation time (#delay, @event wait, wait).

    - Flexible Arguments: Can have multiple arguments of any type (input, output, inout).
    - No Return Value: Tasks don't return a value directly.
- **function (The Instant Calculator)**:
    - Zero Time: Execute instantly, consuming no simulation time.
    - Returns a Value: Must return a single value.
    - Input Only: Primarily work with input arguments.
    - new() is a Function: The new() constructor is a special type of function that executes instantly and returns the class handle.

Deep Dive: Advanced Class Concepts

Let's explore advanced concepts for elevating your OOP game in SystemVerilog.

1. Customizing Object Creation: Explicit Constructors (function new())

Defining your own function new() inside a class allows you to customize how an object is initialized.

- **Signature**: Your custom new() can take arguments, allowing you to pass initial values.
- **this Keyword**: Refers to the current object instance.

```
class Packet;

  rand int id;

  rand int length;

  function new(int initial_id = 0, int initial_length = 8);

    this.id = initial_id;

    this.length = initial_length;

    $display("--- Packet %0d created with length %0d ---", id, length);

  endfunction
```

endclass

2. Visibility: Class Scope vs. Local Scope

Understanding scope is crucial for robust code.

- **Class Scope**: Properties and methods declared directly within class ... endclass.
- **Local Scope**: Variables declared inside a specific task or function.

3. Building Hierarchies: Inheritance (extends)

Inheritance allows you to create a new class (subclass/child class) that gets all properties and methods from an existing class (superclass/parent class).

- **The extends Keyword**: Declares inheritance.
- **Benefits**: Code reuse, clear hierarchy, maintainable testbench.
- **super Keyword**: Calls the parent class's constructor (super.new()) or parent's version of an overridden method (super.method_name()).

```
class Animal;

  string name;

  virtual function void speak();

    $display("%s makes a generic sound.", name);

  endfunction

endclass

class Dog extends Animal;

  int age;

  virtual function void speak();

    $display("%s barks loudly!", name);
```

```
  endfunction

endclass
```

4. Dynamic Behavior: Polymorphism (virtual methods)

Polymorphism allows a single class handle to refer to objects of different (related) types, and for a method call to execute the correct, type-specific behavior at runtime.

- **The Key: virtual**: To enable polymorphism, declare the method as virtual in the base class.

```
module tb_polymorphism_demo;

  initial begin

    Animal a_handle;

    a_handle = new("Max", 5);

    a_handle.speak(); // Dog's method is called!

    a_handle = new("Fluffy");

    a_handle.speak(); // Animal's method is called!

  end

endmodule
```

5. Protecting Your Data: Encapsulation (Access Specifiers)

Encapsulation controls access to data.

- **Access Specifiers**:
  - **public**: Accessible from anywhere.
  - **protected**: Accessible within the class and its derived classes.
  - **local**: Accessible only within the class where it's declared.

```
class BankAccount;
  protected int balance;
  local string account_number;
  public function void deposit(int amount);
    // ...
  endfunction
  public function int get_balance();
    return this.balance;
  endfunction
endclass
```

By using these concepts, you can create powerful, reusable, and maintainable SystemVerilog verification environments.