Lab 3b
SystemVerilog Class Polymorphism and Virtual Methods

Module ID : CX-301

Design Verification

Instructor : Dr. Abid Rafique

Version 1.1

Document History

The changes and versions of the document are outlined below:

Version	State / Changes	Date	Author
1.0	Initial Draft	Jan, 2024	Qamar Moavia
1.1	Modified with new exercises	feb, 2025	Qamar Moavia

Table of Contents

Objectives

By the end of this Lab3 part B, students will be able:

- Use SystemVerilog OOP features to implement real-world designs like counters and timers.

Tools

- SystemVerilog
- Synopsys VCS

Instructions for Lab Tasks

This is part B of Lab3, you will start from the solution of Lab3 part A. The submission must follow the hierarchy below, and the file names exactly as listed below.

```
./Lab3b/
        ├── Task1/
        │   ├── simple_class.sv
        │   ├── constructor.sv
        │   ├── derived_class.sv
        │   ├── counter_limits.sv
        │   ├── roll_over_under.sv
        │   ├── static_members.sv
        │   ├── aggregate.sv
        │   ├── count_polymorphism.sv
        │   ├── // screenshots of outputs
```

Along with that you also need to upload your solution on the github as well, and share the link.

Task 1: Using Class Polymorphism and Virtual Methods

To use SystemVerilog Polymorphism, Virtual Classes and Virtual Methods.

Copy the solution of Lab3a into the Lab3b directory. Working in the Lab3b/Task1 directory perform the following:

1. Modify the counter class to declare it as virtual.
2. Add a next method to the counter class to match the next methods in upcounter and downcounter, so that counter next is overridden by the next methods of the subclasses. Inside the counter next method, simply display a message reporting that you are in the counter class.
3. Comment out your existing verification code and add new code as follows:
   a. Declare a counter class handle, but do not construct an instance. As the counter class is now virtual, trying to create an instance will generate compiler errors.
   b. Create an instance of upcounter and assign this to the counter handle.
   c. Call next from the counter handle.
4. Simulate and debug as needed. The next call from the counter handle should call the counter next implementation, even though the handle contains a subclass instance.
5. Modify your verification code as follows:
   a. Declare another upcounter handle and use $cast to copy the upcounter instance from the counter handle to this new upcounter handle.
   a. Add a next call from the new upcounter handle.
6. Simulate and debug as needed. The next call from the new upcounter handle should call the upcounter next implementation.
7. Modify your code to declare the next method of the counter class as virtual.

8. Simulate and debug as needed. Since next is now virtual, you should see that calling next from **both** the counter handle (containing an upcounter instance) and from an upcounter handle are directed to the upcounter next implementation.

Good Luck 🙂