Lab 3a
SystemVerilog OOP

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 lab, students will be able:

- To understand OOP in SystemVerilog.
- To understand and use Inheritance in SystemVerilog OOP.
- To understand and use the static properties and methods.
- To understand and use aggregate classes.
- Use SystemVerilog OOP features to implement real-world designs like counters and timers.

Tools

- SystemVerilog
- Synopsys VCS

Instructions for Lab Tasks

This lab doesn't require any files. You will be creating everything from scratch. The submission must follow the hierarchy below, with the folder named after the trainee (no spaces), and the file names exactly as listed below.

```
./student_name_lab3a/
        ├── Task1/
        │    ├── simple_class.sv
        │    ├── constructor.sv
        │    ├── derived_class.sv
        │    ├── counter_limits.sv
        │    ├── roll_over_under.sv
        │    ├── static_members.sv
        │    ├── aggregate.sv
        │    ├── // screenshots of outputs
```

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

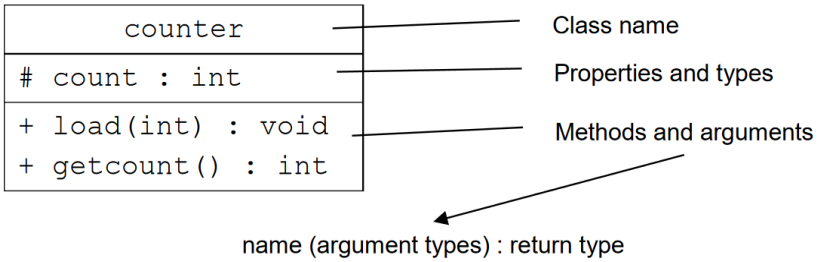
TASK 1 : Using Classes

In this task you will be creating a simple counter design using the following SystemVerilog object-oriented design features:

- Class declarations, with explicit constructors and instances
- Inheritance
- Static properties and methods
- Aggregate classes (classes with properties of other class types)

Creating a Simple Class

1. Create a file counter.sv to declare a class as described in this diagram.

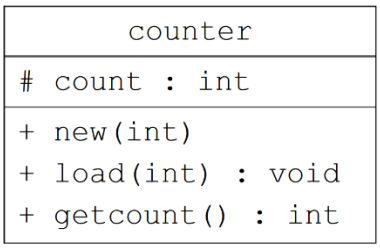


where load() sets property count and getcount () returns count.

2. Create instances of class counter and use the methods. Simulate and debug as needed.

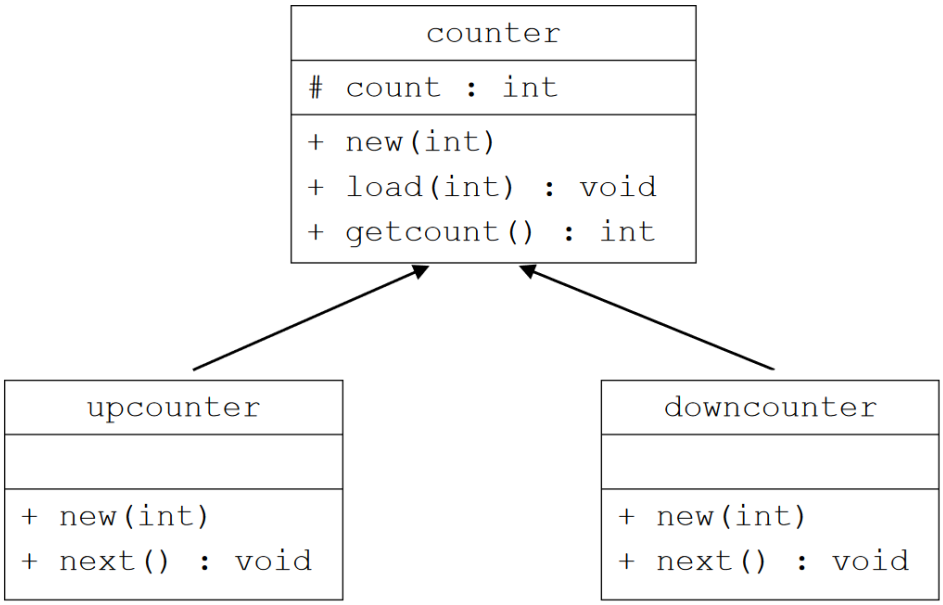
Adding a Class Constructor

3. Add a class constructor with one int argument to set the initial value of count. Give the argument a default value of 0.
4. Modify your code to test the constructor. Simulate and debug as needed.



Defining Derived Classes

5. Extend the counter class by declaring two subclasses, upcounter and downcounter, as follows:



   - upcounter and downcounter inherit the count property from counter.
   - upcounter and downcounter have explicit constructors, with a single argument, which pass the argument to the counter constructor.
   - The upcounter next method increments count and display the value for debugging.
   - The downcounter next method decrements count and displays the value for debugging.
6. Create instantiations of both subclasses and verify their methods. Simulate and debug as needed.

Setting Counter Limits

You need to control the upper and lower count limits of both counter subclasses.

7. Add max and min properties (type int) to counter for upper and lower count limits.
8. Add a check_limit method to counter with two input int arguments. The method should assign the greater of the two arguments to max and the lesser to min.
9. Add a check_set method to counter with a single input int argument count:
   - If the set argument is not within the max-min limits, assign count from min

   and display a warning message. Otherwise, assign count from set.
10. Add arguments for the max and min limits to the constructors so they can be set for each class instance. The upcounter and downcounter constructors should pass the limit arguments to the counter constructor.
11. In the counter constructor, call check_limit with the two limit arguments to ensure that max and min are consistent and check_set to ensure the initial count value is within limits.
12. In the load method, call check_set to ensure that the load value is within limits.
13. Modify the upcounter and downcounter next()methods to count between the two limits, e.g., upcounter counts up to max and then rolls over to min.
14. Modify your test code to check the new functionality. Simulate and debug as needed.

Indicating Roll-Over and Roll-Under

You need to indicate when an upcounter instance rolls over, or downcounter instance rolls under.

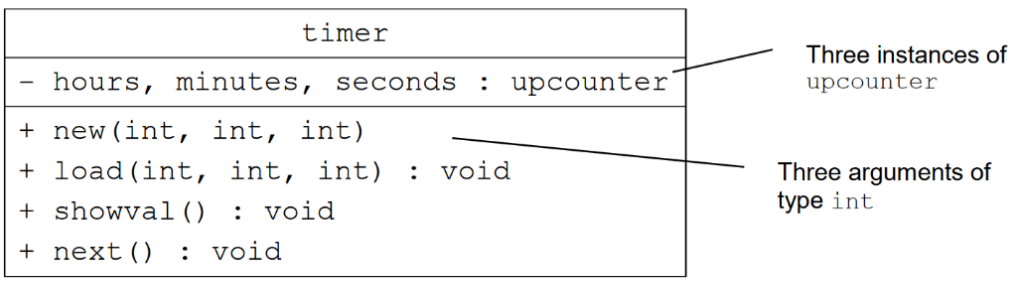
15. Modify upcounter as follows:
    a. Add a new property carry of type bit, initialized to 0 in the constructor.
    b. Modify the next method to set carry to 1 only when count rolls over from max to min; otherwise, carry should be 0.
16. Modify downcounter as follows:
    a. Add a new property borrow of type bit, initialized to 0 in the constructor.
    b. Modify the next method to set borrow to 1 only when the count rolls under from min to max, otherwise borrow should be 0.
17. Modify your test code to check the carry and borrow work. Simulate and debug as needed.

Implementing a Static Property and a Static Method

18. Add a static property to both upcounter and downcounter to count the number of instances of the class that have been created.
19. Increment the property in the class constructors.
20. Add a static method to both classes, which returns the static property.
21. Modify your test code to check the static properties work. Simulate and debug as needed.
22. For a tool to probe class values, the objects must be constructed (probably not yet at time 0).

Defining an Aggregate Class

23. Define a new aggregate class timer as follows:



where:

- hours, minutes and seconds are three instances of upcounter.
- The timer constructor has three arguments for the initial hour, minute and second counts (with default values). The constructor creates each upcounter instance with the appropriate initial value and sets max and min to operate the timer as a clock, e.g., the hours instance should have a max of 23 and a min of 0.
- The load method has three arguments to set the hour, minute and second counts.
- The showval method displays the current hour, minute and second.
- The next method increments the timer and displays the new hour, minute and second. Increment the timer by making calls to the next methods of the upcounter instance and use the carry properties to control which next method is called, e.g., the minutes next method is called only when the seconds carry is 1. Use the showval method to display the new values.

Modify your test code to check the timer works. Use load to set the timer to values such as 00:00:59 and next to check the roll-over. Simulate and debug as needed.

Good Luck 🙂