SystemVerilog Abstract Class

SystemVerilog prohibits a class declared as `virtual` to be directly instantiated and is called an *abstract* class.

Syntax

```
virtual class <class_name>
    // class definition
endclass
```

However, this class can be extended to form other sub-classes which can then be instantiated. This is useful to enforce testcase developers to always extend a base class to form another class for their needs. So *base* classes are usually declared as `virtual` although it is not mandatory.

Normal Class Example

```
class BaseClass;
    int data;

    function new();
        data = 32'hc0de_c0de;
    endfunction
endclass

module tb;
    BaseClass base;
    initial begin
        base = new();
        $display ("data=0x%0h", base.data);
    end
endmodule
```

- **OUTPUT**

```
ncsim> run
 data=0xc0dec0de
ncsim: *W,RNQUIE: Simulation is complete.
```

Abstract Class Example

Let us declare the class `BaseClass` as `virtual` to make it an *abstract* class and see what happens.

```
1  virtual class BaseClass;
2      int data;
3
4      function new();
5          data = 32'hc0de_c0de;
6      endfunction
7  endclass
8
9  module tb;
10     BaseClass base;
11     initial begin
12         base = new();
13         $display ("data=0x%0h", base.data);
14     end
15 endmodule
```

A compilation error is reported by the simulator as shown below since abstract classes are not allowed to be instantiated.

- OUTPUT

```
        base = new();
           |
ncvlog: *E,CNIABC (testbench.sv,12|5): An
abstract (virtual) class cannot be instantiated.
```

Click to try this example in a simulator! (https://www.edaplayground.com/x/5_mR)

Extending Abstract Classes

Abstract classes can be extended just like any other SystemVerilog class using the `extends` keyword like shown below.

```
virtual class BaseClass;
    int data;

    function new();
        data = 32'hc0de_c0de;
    endfunction
endclass

class ChildClass extends BaseClass;
    function new();
        data = 32'hfade_fade;
    endfunction
endclass

module tb;
    ChildClass child;
    initial begin
        child = new();
        $display ("data=0x%0h", child.data);
    end
endmodule
```

It can be seen from the simulation output below that it is perfectly valid to extend abstract classes to form other classes that can be instantiated using `new()` method.

- **OUTPUT**

```
ncsim> run
 data=0xfadefade
ncsim: *W,RNQUIE: Simulation is complete.
```

> Click to try this example in a simulator! (https://www.edaplayground.com/x/3B9K)

Pure Virtual Methods

A `virtual` method inside an abstract class can be declared with the keyword `pure` and is called a *pure virtual* method. Such methods only require a prototype to be specified within the abstract class and the implementation is left to defined within the sub-classes.

Pure Method Example

```
virtual class BaseClass;
    int data;

    pure virtual function int getData();
endclass

class ChildClass extends BaseClass;
    virtual function int getData();
        data = 32'hcafe_cafe;
        return data;
    endfunction
endclass

module tb;
    ChildClass child;
    initial begin
        child = new();
        $display ("data = 0x%0h", child.getData(
    end
endmodule
```

The pure virtual method prototype and its implementation should have the same arguments and return type.

OUTPUT

```
ncsim> run
 data = 0xcafecafe
ncsim: *W,RNQUIE: Simulation is complete.
```

Click to try this example in a simulator! (https://www.edaplayground.com/x/3SWh)

INTERVIEW QUESTIONS

- SystemVerilog Interview Set 1 (/systemverilog/systemverilog-interview-questions-set-1)
- SystemVerilog Interview Set 2 (/systemverilog/systemverilog-interview-questions-set-2)
- SystemVerilog Interview Set 3 (/systemverilog/systemverilog-interview-questions-set-3)
- SystemVerilog Interview Set 4 (/systemverilog/systemverilog-interview-questions-set-4)
- SystemVerilog Interview Set 5 (/systemverilog/systemverilog-interview-questions-set-5)

Verification Guide

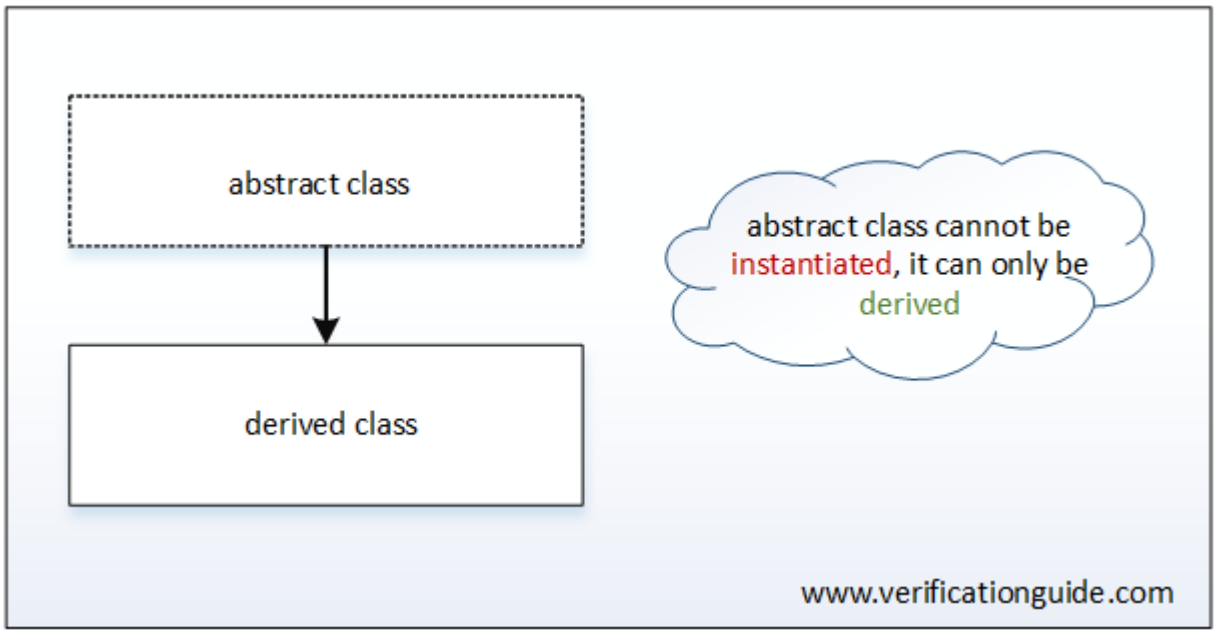
Abstract Class in SystemVerilog

Abstract Class

Table of Contents

SystemVerilog class declared with the keyword virtual is referred to as an *abstract* class.

- An abstract class sets out the prototype for the sub-classes.
- An abstract class cannot be instantiated, it can only be derived.
- An abstract class can contain methods for which there are only a prototype and no implementation, just a method declaration.



Abstract Class SystemVerilog

Abstract class Syntax

```
virtual class abc;
  //Class defination
endclass
```

Abstract Class Examples

Instantiating virtual class

In the below example, Creating an object of a virtual class. An abstract class can only be derived, creating an object of a virtual class leads to a compilation error.

```
//abstract class
virtual class packet;
  bit [31:0] addr;
endclass
module virtual_class;
  initial begin
    packet p;
    p = new();
  end
endmodule
```

Simulator Output

```
virtual_class, "p = new();"
Instantiation of the object 'p' can not be done because its type
'packet' is
an abstract base class.
Perhaps there is a derived class that should be used.
```

Click to execute on EDA playground

Deriving virtual class

In the below example, An abstract class is derived and written extend the class and creating it.

```
//abstract class
virtual class packet;
  bit [31:0] addr;
endclass

class extended_packet extends packet;
  function void display;
    $display("Value of addr is %0d", addr);
  endfunction
endclass

module virtual_class;
  initial begin
    extended_packet p;
    p = new();
    p.addr = 10;
    p.display();
  end
endmodule
```

Simulator Output

```
Value of addr is 10
```

Click to execute on EDA playground

‹ Previous Next ›

Verification Guide

SystemVerilog Virtual Method

Virtual Methods in SystemVerilog

Table of Contents

SystemVerilog Methods declared with the keyword *virtual* are referred to as virtual methods.

Virtual Methods,

- Virtual Functions
- Virtual Tasks

Virtual Functions

A function declared with a virtual keyword before the function keyword is referred to as virtual Function

Virtual Task

Task declared with a virtual keyword before the task keyword is referred to as virtual task

About Virtual Method

In a virtual method,
If the base_class handle is referring to the extended class, then the extended class method handle will get assigned to the base class handle.

In the below explanation, extended_class is an extended class of base_class.

```
base_class    b_c;
```

```
extended_class e_c;
```

Considering both the class's has the method display().

assigning e_c to b_c,

```
b_c = e_c;
```

On calling b_c.display()

- if display() method in base_class is virtual, then extended class display method will get called
- if display() method in base_class is non-virtual, then base class display method will get called

Virtual function syntax

```
virtual function function_name;

  //Function definition
endfunction
```

Virtual task syntax

```
virtual task task_name;
  //task definition
endtask
```

Virtual Method Examples

Method without virtual keyword

In the below example,
the method inside the base class is declared without a virtual keyword, on calling method of the base class which is pointing to the extended class will call the base class method.

```
class base_class;

  function void display;
    $display("Inside base_class");
  endfunction

endclass

class extended_class extends base_class;

  function void display;
    $display("Inside extended class");
  endfunction
```

```
endclass

module virtual_class;
  initial begin
    base_class     b_c;
    extended_class e_c;

    e_c = new();
    b_c = e_c;

    b_c.display();
  end
endmodule
```

Simulator Output

```
Inside base_class
```

Click to execute on EDA playground

A method with virtual keyword

In the below example,
the method inside the base class is declared with a virtual keyword, on calling method of the base class which is pointing to an extended class will call the extended class method.

```
class base_class;

  virtual function void display;
    $display("Inside base_class");
  endfunction

endclass

class extended_class extends base_class;
```

```
  function void display;
    $display("Inside extended_class");
  endfunction

endclass

module virtual_class;
  initial begin
    base_class     b_c;
    extended_class e_c;

    e_c = new();
    b_c = e_c;

    b_c.display();
  end
endmodule
```

Simulator Output

```
Inside extended_class
```

Click to execute on EDA playground

‹ Previous

Next ›