


EGC 455  
SOC Design & Verification


***Packages, Includes, and  
Macros***



**Baback Izadi**  
Division of Engineering Programs  
bai@engr.newpaltz.edu

1

**Packages**



2

### NameSpaces


```
21 module top;
22   initial $timeformat(-12, 0, " ps");
23   mod1 m1();
24   mod2 m2();
25 endmodule // top
```

```
1 module mod1;
2   integer a;
3
4   initial begin
5     #5ns;
6     a = 1;
7     $display("%0t: a in %m:%0d", $time, a);
8   end
9 endmodule // mod1
```

```
11 module mod2;
12  integer a;
13
14  initial begin
15    a = 10;
16    #6ns;
17    $display ("%0t: a in %m:%0d", $time, a);
18  end
19 endmodule // mod2
```

```
# Loading work.mod2
# 5000 ps: a in top.m1:1
# 6000 ps: a in top.m2:10
VSIM 2>
```

Why are these different?



3


### Sharing Variables Across Namespaces

```
2 module mod1;
3   integer a;
4   initial begin
5     #5ns;
6     a = 1;
7     top.m2.a = a;
8     $display("%0t: a in %m:%0d", $time, a);
9   end
10 endmodule // mod1
```

```
12 module mod2;
13  integer a;
14  initial begin
15    a = 10;
16    #6ns;
17    $display ("%0t: a in %m:%0d", $time, a);
18  end
19 endmodule // mod2
```

```
# Loading work.mod2
# 5000 ps: a in top.m1:1
# 6000 ps: a in top.m2:1
```

Why are these the same?



4

## Packages: A Common Namespace

```
1 package mypackage;  
2   integer a;  
3 endpackage // mypackage
```

```
1 module mod1;  
2   initial begin  
3     #5ns;  
4     mypackage::a = 1;  
5     $display("%0t: a in %m:%0d", $time, mypackage::a);  
6   end  
7 endmodule // mod1
```

```
1 module mod2;  
2   integer a;  
3   initial begin  
4     mypackage::a = 10;  
5     #6ns;  
6     $display ("%0t: a in %m:%0d", $time, mypackage::a);  
7   end  
8 endmodule // mod2
```

```
# Loading work.mod2  
# 5000 ps: a in top.m1:1  
# 6000 ps: a in top.m2:1
```

Why are these the same?



5

## Import Packages to Access Namespace

```
1 package mypackage;  
2   integer a;  
3 endpackage // mypackage  
4  
   mypackage.sv
```

```
1 import mypackage::*;  
2  
3 module mod1;                               mod1.sv  
4   initial begin  
5     #5ns;  
6     a = 1;  
7     $display("%0t: a in %m:%0d", $time, a);  
8   end  
9 endmodule // mod1
```

```
1 import mypackage::*;  
2  
3 module mod2;                               mod2.sv  
4   initial begin  
5     a = 10;  
6     #6ns;  
7     $display ("%0t: a in %m:%0d", $time, a);  
8   end  
9 endmodule // mod2
```

```
# Loading work.mod2  
# 5000 ps: a in top.m1:1  
# 6000 ps: a in top.m2:1
```

Why are these the same?



6

## Declaration Overrides

```
1 package vars_pkg;
2
3     integer a = 10;
4
5 endpackage : vars_pkg;
6
```

```
1 import vars_pkg::*;
2
3 module top;
4
5     integer a;
6
7     initial
8     begin
9         a = 1;
10        $display(a);
11    end
12 endmodule : top
```

```
VSIM 1> run 1
#
VSIM 2> 1
```



7

## Packages and Classes

```
1 package mypackage;
2     typedef enum {read, write} op_t;
3     typedef logic [7:0] data_t;
4     typedef logic [15:0] addr_t;
5
6 class mem_op;
7     rand op_t op;
8     rand data_t data;
9     rand addr_t addr;
10
11     function string convert2string();
12     string s;
13     $sformat(s, "data: %2h addr: %4h op: %0s", data, addr, op);
14     return s;
15 endfunction // convert2string
16 endclass // mem_op
17
18     mem_op common_op = new();
19
20 endpackage // mypackage
```

Common typedefs

Common class definition

Shared instance



8

### Accessing common and local objects


```
1 import mypackage::*;
2
3 module mod1;
4     mem_op my_op = new();
5     initial begin
6         common_op.op=write;
7         common_op.addr = 0;
8         common_op.data = 8'hFF;
9         my_op.op = write;
10        my_op.addr = 0;
11        my_op.data = 8'hFF;
12        #10;
13        $display("%0t: common_op in %m:%0s",
14                $time,common_op.convert2string());
15        $display("%0t: myop in %m:%0s",
16                $time,my_op.convert2string());
17    end
18 endmodule // mod1
```

Import package: mypackage

Declare local object

Set values in common object

Set values in local object



9

### Randomizing common and local objects


```
1 import mypackage::*;
2
3 module mod2;
4     mem_op my_op = new();
5     initial begin
6         #6ns;
7         assert(common_op.randomize());
8         assert(my_op.randomize());
9         $display ("%0t: common_op in %m:%0d",
10                $time,common_op.convert2string());
11        $display ("%0t: my_op in %m:%0d",
12                $time,my_op.convert2string());
13    end
14 endmodule // mod2
```

Import package: mypackage

Declare local object

Randomize() common object

Randomize() local object



10

## Package and Class Example

```
# 6000 ps: common_op in top.m2:data: d6 addr: 33b8 op: read
# 6000 ps: my_op in top.m2:data: d6 addr: 33b8 op: read
# 10000 ps: common_op in top.m1:data: d6 addr: 33b8 op: read
# 10000 ps: myop in top.m1:data: ff addr: 0000 op: write
```

The common object is the same in both modules

	Time 0	Time 6 ns
common_op	0, FF, write	random
my_op m1	0, FF, write	
my_op m3		random



11

## The UVM is delivered as a package

```
1  import uvm_pkg::*;
2
3  module top;
4      initial begin
5          #10ns;
6          uvm_top.uvm_report_info($psprintf("%m"),
7                                  "THIS IS AN INFO MESSAGE");
8      end
9  endmodule // top
```

UVM\_top is declared and constructed in the UVM\_pkg package.

```
27  #
28  # UVM-1.1b
29  # (C) 2007-2012 Mentor Graphics Corporation
30  # (C) 2007-2012 Cadence Design Systems, Inc.
31  # (C) 2006-2012 Synopsys, Inc.
32  # (C) 2011-2012 Cypress Semiconductor Corp.
33  #
34  #
35  # UVM_INFO @ 10: reporter [top] THIS IS AN INFO MESSAGE
36
```

12

## Packages

- **Provide a common name space**
- **Can be imported**
- **Can contain common objects**

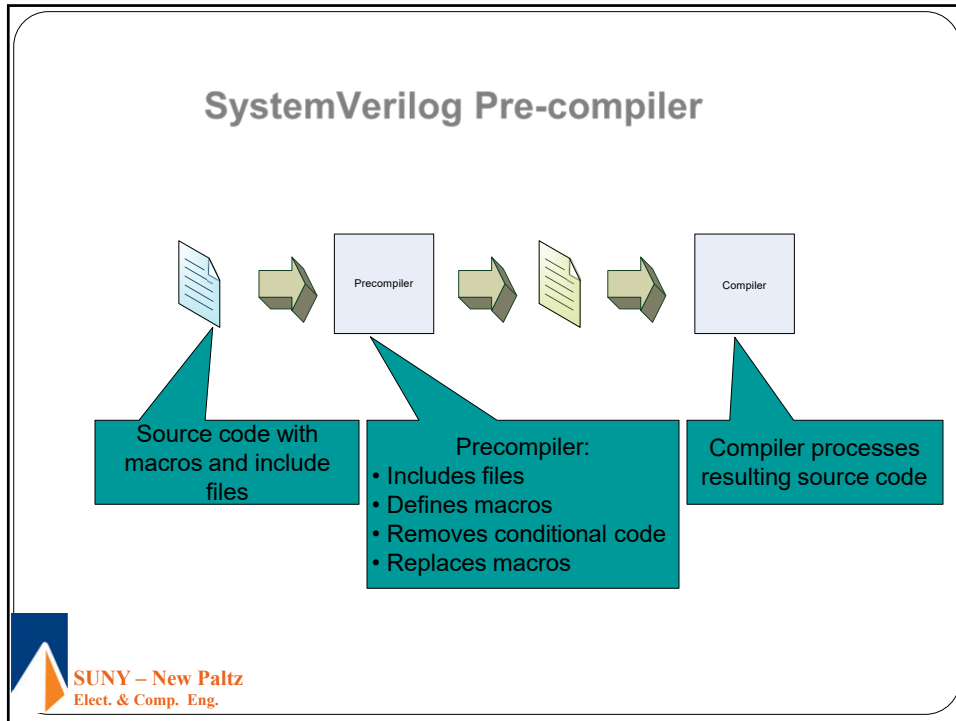


13

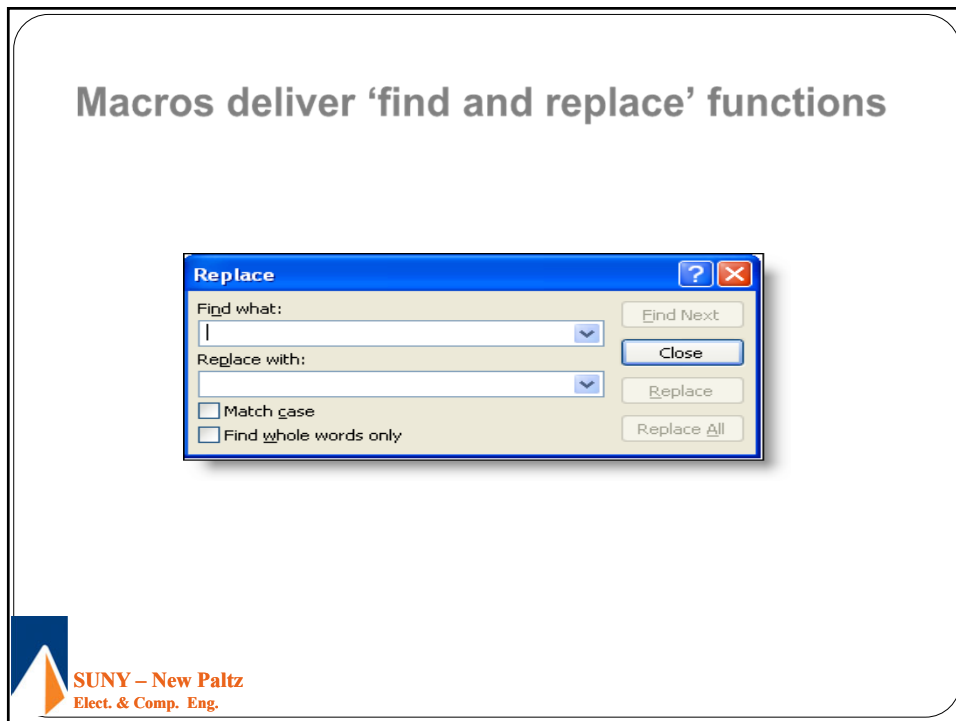
## Macros



14



15



16



## Macro Example

```
1  `define info(msg)      uvm_top.uvm_report_info($sprintf("%m"), msg);
2  `define warning(msg)  uvm_top.uvm_report_warning($sprintf("%m"), msg);
3  `define error(msg) \
4      uvm_top.uvm_report_error($sprintf("%m"), msg);
5  `define fatal(msg)   uvm_top.uvm_report_fatal($sprintf("%m"), msg);
6
7  import uvm_pkg::*;
8
9  module top;
10     initial begin
11         `info ("My INFO message");
12         `warning("My WARNING message");
13         `error("My ERROR message");
14         `fatal("It's FATAL!");
15     end
16 endmodule // top
17
```

Macros are not function calls. They are textual replacements

```
    `info ("My INFO message");
        becomes
uvm_top.uvm_report_info($sprintf("%m"), "My INFO message");;
```



17

## Running Report Macros

```
27  # -----
28  # UVM-1.1b
29  # (C) 2007-2012 Mentor Graphics Corporation
30  # (C) 2007-2012 Cadence Design Systems, Inc.
31  # (C) 2006-2012 Synopsys, Inc.
32  # (C) 2011-2012 Cypress Semiconductor Corp.
33  # -----
34  #
35  # UVM_INFO @ 0: reporter [top] My INFO message
36  # UVM_WARNING @ 0: reporter [top] My WARNING message
37  # UVM_ERROR @ 0: reporter [top] My ERROR message
38  # UVM_FATAL @ 0: reporter [top] It's FATAL!
39  #
40  # --- UVM Report Summary ---
41  #
42  # ** Report counts by severity
43  # UVM_INFO : 3
44  # UVM_WARNING : 1
45  # UVM_ERROR : 1
46  # UVM_FATAL : 1
47  # ** Report counts by id
48  # [Questa UVM] 2
49  # [top] 4
50  # ** Note: $finish : /tools/mentor/questa/10.1c_1/questasim/linux
51  # Time: 0 ns Iteration: 0 Instance: /top
```



18

## Conditional Compilation

```
1 import uvm_pkg::*;
2
3 `define INFO(msg)      uvm_top.uvm_report_info($sprintf("%m"), msg);
4
5 `ifdef FIRST
6 module top;
7     initial `INFO("FIRST Defined");
8 endmodule // top
9
10 `else
11
12 module top;
13     initial `INFO("FIRST Undefined");
14 endmodule // top
15
16 `endif // !`ifdef FIRST
17
```

Compiled if FIRST macro  
is defined



19

## Conditional with Undefined Macro

```
1 import uvm_pkg::*;
2
3 `define INFO(msg)      uvm_top.uvm_report_info($sprintf("%m"), msg);
4
5 `ifdef FIRST
6 module top;
7     initial `INFO("FIRST Defined");
8 endmodule // top
9
10 `else
11
12 module top;
13     initial `INFO("FIRST Undefined");
14 endmodule // top
15
16 `endif // !`ifdef FIRST
17
```


Compiled if FIRST macro  
is NOT defined



20


### Define macro when you compile

<pre>1  if [file exists work] {vdel -all} 2  vlib work 3  vlog top.sv +define+FIRST 4  vsim -novopt top 5  run -all</pre> <p style="text-align: center;">run_first.do</p>	<pre>1  if [file exists work] {vdel -all} 2  vlib work 3  vlog top.sv 4  vsim -novopt top 5  run -all</pre> <p style="text-align: center;">run_nofirst.do</p>
<pre>28 # 29 # UVM-1.1b 30 # (C) 2007-2012 Mentor Graphics Corporation 31 # (C) 2007-2012 Cadence Design Systems, Inc. 32 # (C) 2006-2012 Synopsys, Inc. 33 # (C) 2011-2012 Cypress Semiconductor Corp. 34 # 35 # 36 # UVM_INFO @ 0: reporter [top] FIRST Defined 37 #</pre>	<pre>68 # 69 # UVM-1.1b 70 # (C) 2007-2012 Mentor Graphics Corporation 71 # (C) 2007-2012 Cadence Design Systems, Inc. 72 # (C) 2006-2012 Synopsys, Inc. 73 # (C) 2011-2012 Cypress Semiconductor Corp. 74 # 75 # 76 # UVM_INFO @ 0: reporter [top] FIRST Undefined 77 #</pre>



21

## Includes



22

### Include same as paste

```

1 `include "tester.svh"
2
3 module top;
4
5     memory_if mi();
6     memory dut (mi.mem_mp);
7     tester tst;
8
9     initial begin
10        tst = new(mi);
11        fork
12            tst.run;
13        join_none
14        end
15    endmodule // top
16
17
        
```

top.sv


```

1 class tester;
2
3     logic [3:0] tiny_addr;
4     virtual interface memory_if t;
5
6     function new(virtual interface memory_if it);
7         t = it;
8     endfunction // new
9
10    task run;
11        t.wr = 1'b1;
12        for (int i=0; i< 'h10; i++) begin
13            @(negedge t.clk);
14            t.wr_data_reg = i;
15            t.addr = i;
16        end
17
18        repeat (50) begin
19            @(negedge t.clk);
20            t.wr = $random;
21            t.rd = ~t.wr;
22            tiny_addr = $random;
23        end
24    end
        
```

tester.svh

tester.svh replaces include statement

.svh extension is a convention for headers




23

### Replace Common Code

```

1 `define info(msg)      uvm_top.uvm_report_info($sprintf("%m"), msg);
2 `define warning(msg)  uvm_top.uvm_report_warning($sprintf("%m"), msg);
3 `define error(msg)    uvm_top.uvm_report_error($sprintf("%m"), msg);
4 `define fatal(msg)    uvm_top.uvm_report_fatal($sprintf("%m"), msg);
5
6 import uvm_pkg::*;
7
8 module top;
9     initial begin
10        `info ("My INFO message");
11        `warning("My WARNING message");
12        `error("My ERROR message");
13        `fatal("It's FATAL!");
14    end
15 endmodule // top
16
17
        
```

These macros could be put into an include file across all files



24

## Sharing Declarations



25

## How do I get declarations into many files?

```
1 include "monitor.svh"
2 class monitor;
3 virtual interface memory_if mi;
4 function new (virtual interface memory_if mi);
5   mi = mi;
6 endfunction
7 task run;
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

monitor.svh

```
1 include "monitor.svh"
2 class scoreboard;
3 virtual interface memory_if mi;
4 logic [15:0] testmem [2**16-1:0];
5 function new(virtual interface memory_if mi);
6   mi = mi;
7 endfunction // new
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

scoreboard.svh

```
1 class tester;
2
3 logic [1:0] tiny_addr;
4 virtual interface memory_if t;
5
6 function new(virtual interface memory_if it);
7   t = it;
8 endfunction // new
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

tester.svh

Need these declarations in our namespace for code to work.

```
2
3 module top;
4
5   memory_if mi ();
6   memory dut (mi.mem_mp);
7   tester tst;
8   scoreboard sb;
9   monitor m;
10
11   initial begin
12     tst = new(mi);
13     sb = new(mi);
14     m = new(mi);
15
16   end
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



26

## One Solution: Include Files

```
1 import uvm_pkg::*;
2
3 `include "tester.svh"
4 `include "monitor.svh"
5 `include "scoreboard.svh"
6
7 module top;
8
9     memory_if mi();
10    memory dut (mi.mem_mp);
11    tester tst;
12    scoreboard sb;
13    monitor m;
14
15    initial begin
16        tst = new(mi);
```

Pro: Easy to Understand

Con: Long List of Files

Con: Need to change filenames in many places



27

## Another Solution: Packages

```
1 package memory_pkg;
2     import uvm_pkg::*;
3
4     class tester;
5
6         logic [3:0] tiny_addr;
7
8     endclass
9
10    class tester;
11
12        logic [3:0] tiny_addr;
13        virtual interface memory_if t;
14
15        function new(virtual interface memory_if it);
16            t = it;
17        endfunction // new
18
19    endclass
20
21    class monitor;
22        virtual interface memory_if mi;
23
24        function new (virtual interface memory_if imi);
25            mi = imi;
26        endfunction
27
28        task run;
29            forever begin
30
31            end
32        endtask
33
34    endclass
35
36    class scoreboard;
37        virtual interface memory_if mi;
38        logic [15:0] testmem [2*16-1:0];
39
40        function new(virtual interface memory_if imi);
41
42        end
43
44        if (mi.wr)
45            testmem[mi.addr] = mi.data;
46        end
47        endtask // run
48    endclass // scoreboard
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

Pros:

- Single file
- One compile

Con:

- One very big file



28

### Suggested Solution Combine Includes and Packages

Import UVM package into your package

```
1 package memory_pkg;
2   import uvm_pkg::*;
3   `include "tester.svh"
4   `include "monitor.svh"
5   `include "scoreboard.svh"
6 endpackage // memory_pkg
7
```

Notice that imports do not chain. You must import all namespaces explicitly

```
1 import uvm_pkg::*;
2 import memory_pkg::*;
3
4 module top;
5
6   memory_if mi();
7   memory dut (mi.mem_mp);
8   tester tst;
9   scoreboard sb;
```


We must import UVM\_pkg into source file

Pros:

- One file per class
- Only compiled once
- All changes in one place

Con:

- Package must be compiled before it is imported.



29


### Compiling

```
1 if [file exists work] {vdel -all}
2 vlib work
3 vlog -f compile.f
4 vsim -voptargs="+acc" top;
5 run -all
6
```

Compile and Run Script

```
1 memory_pkg.sv
2 memory_if.sv
3 memory.sv
4 top.sv
5
```

compile.f – A list of files to compile



30

## Summary

- **Create .svh files to hold each class**
- **Create a package to deliver all class declarations**
- **Include the .svh files in the package file.**
- **Use compile.f to control compiles**



31

## Polymorphism

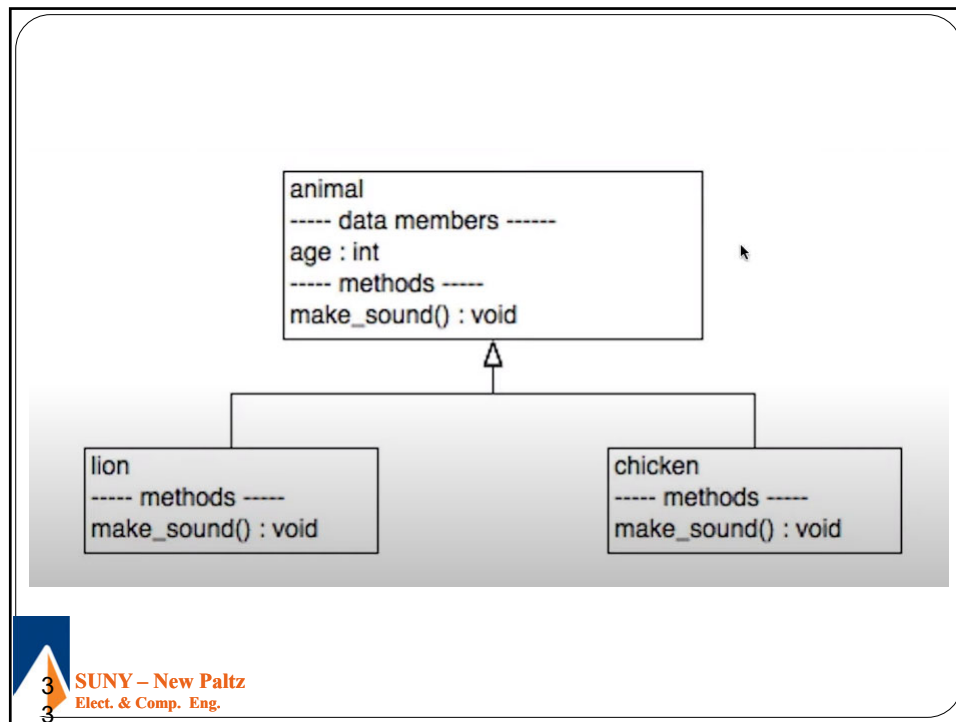
---

Polymorphism is the method in an object-oriented programming language that **performs different things as per the object's class**, which calls it. With Polymorphism, a message is sent to multiple class objects, and every object responds appropriately according to the properties of the class



32





33

```

/* Not Virtual */
class animal;
int age=-1;
function new(int a);
    age = a;
endfunction : new
function void make_sound();
    $fatal(1, "Generic animals don't have a sound.");
endfunction : make_sound
endclass : animal

class lion extends animal;
function new(int age);
    super.new(age);
endfunction : new
function void make_sound();
    $display ("The Lion says Roar");
endfunction : make_sound
endclass : lion

class chicken extends animal;
function new(int age);
    super.new(age);
endfunction : new
function void make_sound();
    $display ("The Chicken says BECAWW");
endfunction : make_sound
endclass : chicken
    
```

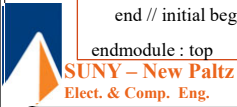
The code block contains two class definitions. The first is 'class animal' with an integer member 'age' initialized to -1, a constructor 'new(int a)' that sets 'age = a', and a 'make\_sound()' method that triggers a fatal error. The second is 'class lion extends animal' which inherits from 'animal', has its own constructor 'new(int age)' that calls 'super.new(age)', and a 'make\_sound()' method that displays "The Lion says Roar". The third is 'class chicken extends animal' which inherits from 'animal', has its own constructor 'new(int age)' that calls 'super.new(age)', and a 'make\_sound()' method that displays "The Chicken says BECAWW". In the bottom left corner, there is a logo for SUNY - New Paltz Elect. & Comp. Eng. with the number 3.

34

```

module top;
  initial begin
    lion lion_h;
    chicken chicken_h;
    animal animal_h;
    lion_h = new(15);
    lion_h.make_sound();
    $display("The Lion is %0d years old", lion_h.age);
    chicken_h = new(1);
    chicken_h.make_sound();
    $display("The Chicken is %0d years old", chicken_h.age);
    animal_h = lion_h;
    animal_h.make_sound();
    $display("The animal is %0d years old", animal_h.age);
    animal_h = chicken_h;
    animal_h.make_sound();
    $display("The animal is %0d years old", animal_h.age);
  end // initial begin
endmodule : top

```



35

```

    $display("The Lion is %0d years old", lion_h.age);

    chicken_h = new(1);
    chicken_h.make_sound();
    $display("The Chicken is %0d years old", chicken_h.age);

    animal_h = lion_h;
    animal_h.make_sound();
    $display("The animal is %0d years old", animal_h.age);

    animal_h = chicken_h;
    animal_h.make_sound();


```

---

```

not_virtual.sv 75% L77  Git-master (Verilog)-----
# // WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# // LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading sv_std.std
# Loading work.not_virtual_sv_unit(fast)
# Loading work.top(fast)
# The Lion says Roar
# The Lion is 15 years old
# The Chicken says BECAWW
# The Chicken is 1 years old
# ** Fatal: Generic animals don't have a sound.
# Time: 0 ns Scope: not_virtual_sv_unit.animal.make_sound File: not_virtual.sv Line: 24
# ** Note: $finish : not_virtual.sv(24)

```




36

```
/* Virtual */
class animal;
  int age=-1;
  function new(int a);
    age = a;
  endfunction : new
  virtual function void make_sound();
    $fatal(1, "Generic animals don't have a sound.");
  endfunction : make_sound
endclass : animal

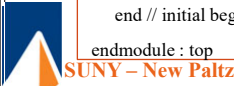
class lion extends animal;
  function new(int age);
    super.new(age);
  endfunction : new
  function void make_sound();
    $display ("The Lion says Roar");
  endfunction : make_sound
endclass : lion

class chicken extends animal;
  function new(int age);
    super.new(age);
  endfunction : new
  function void make_sound();
    $display ("The Chicken says BECAWW");
  endfunction : make_sound
endclass : chicken
```



37

```
module top;
  initial begin
    lion lion_h;
    chicken chicken_h;
    animal animal_h;
    lion_h = new(15);
    lion_h.make_sound();
    $display("The Lion is %0d years old", lion_h.age);
    chicken_h = new(1);
    chicken_h.make_sound();
    $display("The Chicken is %0d years old", chicken_h.age);
    animal_h = lion_h;
    animal_h.make_sound();
    $display("The animal is %0d years old", animal_h.age);
    animal_h = chicken_h;
    animal_h.make_sound();
    $display("The animal is %0d years old", animal_h.age);
  end // initial begin
endmodule : top
```



38

```
# // Version 10.2c linux Jul 18 2013
# //
# // Copyright 1991-2013 Mentor Graphics Corporation
# // All Rights Reserved.
# //
# // THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
# // WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# // LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading sv_std.std
# Loading work.virtual_sv_unit(fast)
# Loading work.top(fast)
# The Lion says Roar
# The Lion is 15 years old
# The Chicken says BECAWW
# The Chicken is 1 years old
# The Lion says Roar
# The animal is 15 years old
# The Chicken says BECAWW
# The animal is 1 years old
```



39

```
/* Pure Virtual */
virtual class animal;
  int age=-1;
  function new(int a);
    age = a;
  endfunction : new
  pure virtual function void make_sound();
endfunction : make_sound
endclass : animal
```

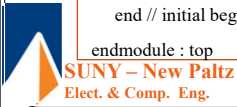
```
class lion extends animal;
  function new(int age);
    super.new(age);
  endfunction : new
  function void make_sound();
    $display ("The Lion says Roar");
  endfunction : make_sound
endclass : lion

class chicken extends animal;
  function new(int age);
    super.new(age);
  endfunction : new
  function void make_sound();
    $display ("The Chicken says BECAWW");
  endfunction : make_sound
endclass : chicken
```




40

```
module top;
  initial begin
    lion lion_h;
    chicken chicken_h;
    animal animal_h;    animal_h = new(3); // will cause a fatal error since we try to call an abstract class.
    lion_h = new(15);
    lion_h.make_sound();
    $display("The Lion is %0d years old", lion_h.age);
    chicken_h = new(1);
    chicken_h.make_sound();
    $display("The Chicken is %0d years old", chicken_h.age);
    animal_h = lion_h;
    animal_h.make_sound();
    $display("The animal is %0d years old", animal_h.age);
    animal_h = chicken_h;
    animal_h.make_sound();
    $display("The animal is %0d years old", animal_h.age);
  end // initial begin
endmodule : top
```



41

```
# do run.do
# QuestaSim vlog 10.2c Compiler 2013.07 Jul 18 2013
# -- Compiling package pure_virtual_sv_unit
# -- Compiling module top
#
# Top level modules:
#   top
# vsim -c -voptargs="+acc\" top
# ** Note: (vsim-3812) Design is being optimized...
# // Questa Sim
# // Version 10.2c linux Jul 18 2013
# //
# // Copyright 1991-2013 Mentor Graphics Corporation
# // All Rights Reserved.
# //
# // THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
# // WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# // LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading sv_std.std
# Loading work.pure_virtual_sv_unit(fast)
# Loading work.top(fast)
# ** Fatal: (vsim-8250) Class allocator method 'new' called on Abstract Class.
#   Time: 0 ns Iteration: 0 Process: /top/#INITIAL#57(#ublk#31584#59) File: pure_virtual.sv
# Fatal error in Module top at pure_virtual.sv line 63
#
```



42

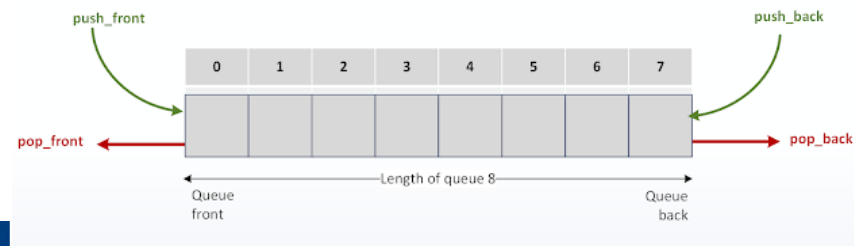
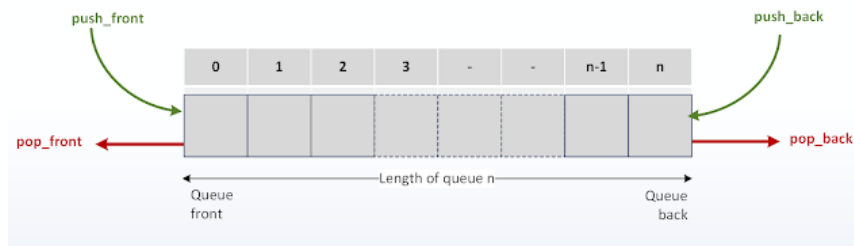
## Static Methods and Variables

Object Oriented programming provides global storage within class definitions using static variables and static methods.



43

## Queue in System Verilog



44

```
// queue declaration
Int qu[$];
```

The diagram shows a sequence of operations on a queue implemented as an array of size 12 (indices 0-11). The elements are 6, 4, 8, 11, 6, 4. The operations are: push\_front(6), push\_front(4), push\_back(8), push\_front(11), pop\_back(), and pop\_front().

**4** SUNY – New Paltz  
**5** Elect. & Comp. Eng.

45

```
virtual class animal;
protected int age=-1;
function new(int age);
set_age(age);
endfunction : new

function void set_age(int a);
age = a;
endfunction : set_age

function int get_age();
if (age == -1)
    $fatal(1, "You didn't set the age.");
else
    return age;
endfunction : get_age

pure virtual function void make_sound();
endclass : animal

class lion extends animal;
protected string name;

function new(int age, string n);
super.new(age);
name = n;
endfunction : new

function void make_sound();
$display ("%s says Roar", get_name());
endfunction : make_sound

function string get_name();
return name;
endfunction : get_name

endclass : lion

class lion_cage;
static lion cage[$];
endclass : lion_cage
```

**4** SUNY – New Paltz  
**5** Elect. & Comp. Eng.

46

```
module top;

  initial begin
    lion lion_h;
    lion_h = new(2, "Kimba");
    lion_cage::cage.push_back(lion_h);
    lion_h = new(3, "Simba");
    lion_cage::cage.push_back(lion_h);
    lion_h = new(15, "Mustafa");
    lion_cage::cage.push_back(lion_h);
    $display("Lions in cage");
    foreach (lion_cage::cage[i])

    $display(lion_cage::cage[i].get_name());
  end
endmodule : top
```



47

### Static Method hides the implementation and make it easier to use our objects

```
virtual class animal;
  protected int age=-1;
  function new(int age);
    set_age(age);
  endfunction : new

  function void set_age(int a);
    age = a;
  endfunction : set_age

  function int get_age();
    if (age == -1)
      $fatal(1, "You didn't set the age.");
    else
      return age;
    endfunction : get_age

  pure virtual function void make_sound();
endclass : animal
```

```
class lion extends animal;

  protected string name;

  function new(int age, string n);
    super.new(age);
    name = n;
  endfunction : new

  function void make_sound();
    $display ("%s says Roar", get_name());
  endfunction : make_sound

  function string get_name();
    return name;
  endfunction : get_name
endclass : lion
```



48



We can create Static function if we don't want to deal with queue.

```
class lion_cage;  
  
    protected static lion cage[$];  
  
    static function void cage_lion(lion l);  
        cage.push_back(l);  
    endfunction : cage_lion  
  
    static function void list_lions();  
        $display("Lions in cage");  
        foreach (cage[i])  
            $display(cage[i].get_name());  
    endfunction : list_lions  
  
endclass : lion_cage
```

```
module top;  
  
    initial begin  
        lion lion_h;  
        lion_h = new(2, "Kimba");  
        lion_cage::cage_lion(lion_h);  
        lion_h = new(3, "Simba");  
        lion_cage::cage_lion(lion_h);  
        lion_h = new(15, "Mustafa");  
        lion_cage::cage_lion(lion_h);  
        lion_cage::list_lions();  
    end  
  
endmodule : top
```

