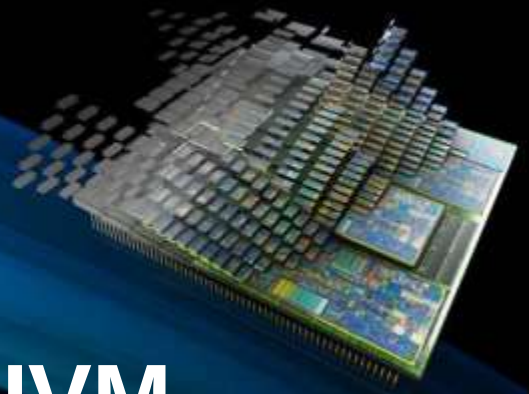
Verification Academy



Introduction to the UVM

Object Oriented Programming

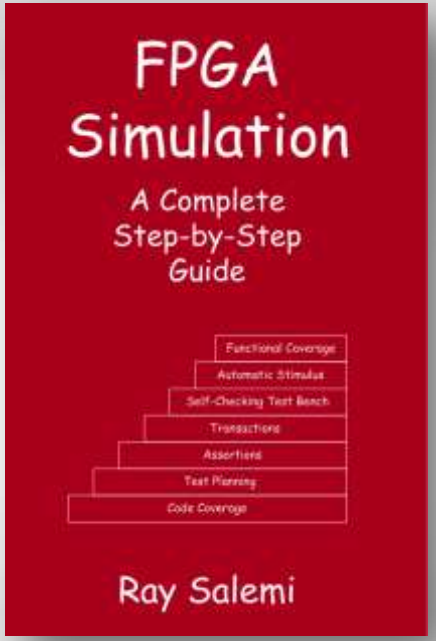
Ray Salemi

Senior Verification Consultant

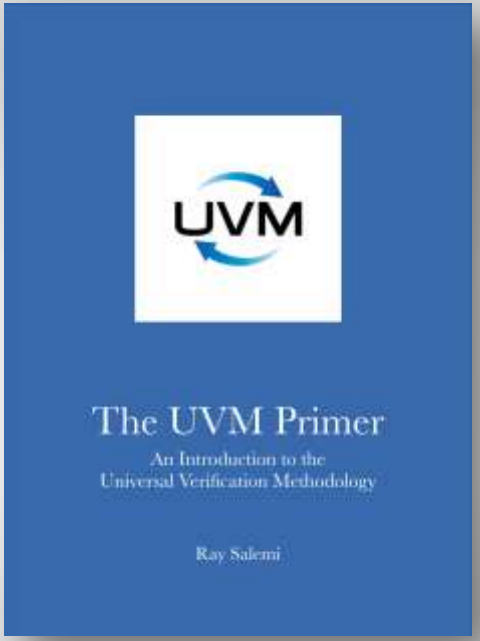
info@verificationacademy.com | www.verificationacademy.com

Mentor Graphics

Ray Salemi — Senior Verification Consultant



Introduction to Advanced Verification



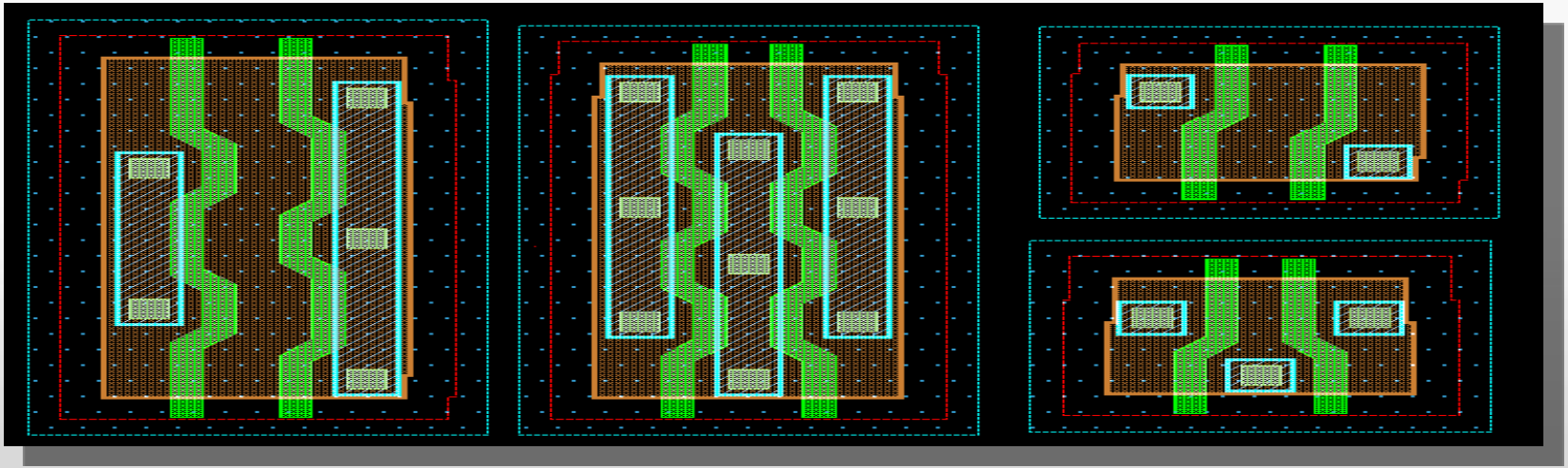
Introduction to the UVM

Agenda

1. SystemVerilog for VHDL Engineers
2. Object Oriented Programming
3. SystemVerilog Interfaces
4. Packages, Includes, and Macros
5. UVM Test Objects
6. UVM Environments
7. Connecting Objects
8. Transaction Level Testing
9. The Analysis Layer
10. UVM Reporting
11. Functional Coverage with Covergroups
12. Introduction to Sequences

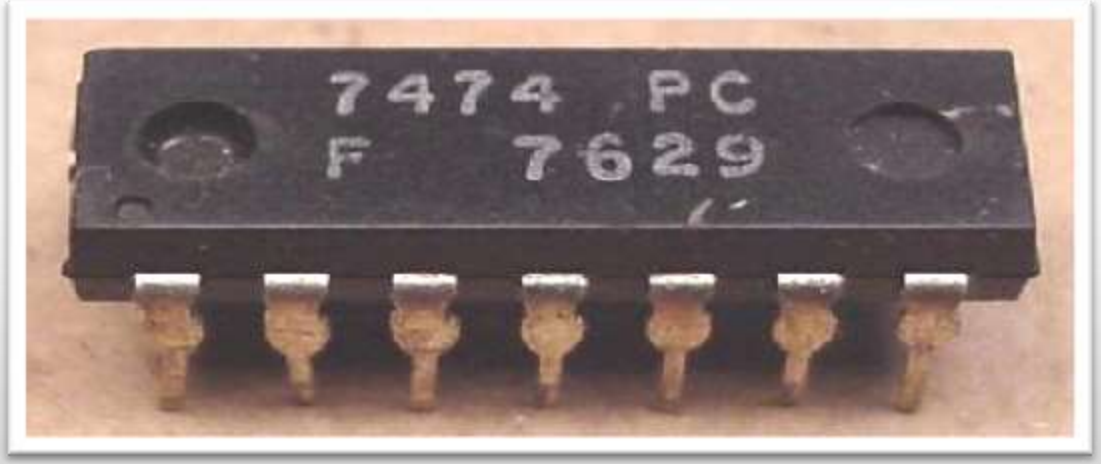
Object Oriented Programming

An Old Hardware Concept



Object Oriented Programming

An Old Hardware Concept



Object Oriented Programming

An Old Hardware Concept

Data Encapsulation in VHDL : Records

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY record_example IS
type rectangle_t is record
  length : integer;
  width  : integer;
end record;
END ENTITY record_example;

--

ARCHITECTURE example OF record_example IS
   shared variable rectangle:rectangle_t;
   shared variable area:integer;
   BEGIN
  process begin
     rectangle.length := 50;
     rectangle.width  := 20;
     area := rectangle.length * rectangle.width;
     report integer'image(area);
     wait;
  end process;


END ARCHITECTURE example;
```

```
VSIM 6> run 1
# ** Note: 1000
#    Time: 0 ns  Iteration: 0  Instance: /record_example

VSIM 7>
```

Rectangle Data Encapsulated in Record

Memory is set aside when variable is declared

Encapsulation in SystemVerilog

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY record_example IS
type rectangle_t is record
  length : integer;
  width  : integer;
end record;
END ENTITY record_example;

--

ARCHITECTURE example OF record_example IS
   shared variable rectangle:rectangle_t;
   shared variable area:integer;
   BEGIN
  process begin
    rectangle.length := 50;
    rectangle.width  := 20;
    area := rectangle.length * rectangle.width;
    report integer'image(area);
    wait;
  end process;


END ARCHITECTURE example;
```

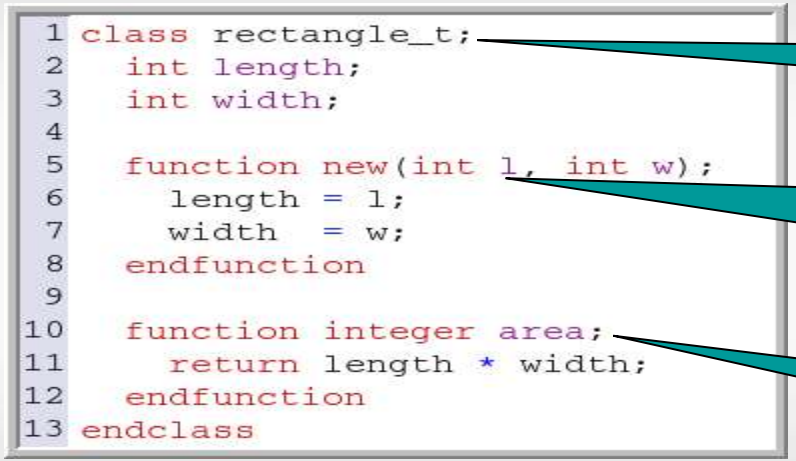
```
class rectangle_t;
  int length;
  int width;

  function new(int l, int w);
    length = l;
    width  = w;
  endfunction

  function integer area;
    return length * width;
  endfunction
endclass

module class_example ;
  rectangle_t rectangle;
  initial begin
    rectangle = new(50,20);
    $display(rectangle.area);
  end
endmodule
```

Terminology



```
1  class rectangle_t;
2    int length;
3    int width;
4
5    function new(int l, int w);
6      length = l;
7      width  = w;
8    endfunction
9
10   function integer area;
11     return length * width;
12   endfunction
13 endclass
```

Differences in SystemVerilog Classes

```
class rectangle_t;
  int length;
  int width;

  function new(int l, int w);
    length = l;
    width  = w;
  endfunction

  function integer area;
    return length * width;
  endfunction
endclass

module class_example ;
  rectangle_t rectangle;
  initial begin
    rectangle = new(50,20);
    $display(rectangle.area);
  end
endmodule
```

Declared as class

Includes functions and tasks as methods

Memory is not allocated here

Memory allocated here

Method called to determine area

Advantage of Classes: Extension

```
class rectangle_t;
  int length;
  int width;

  function new(int l, int w);
    length = l;
    width  = w;
  endfunction

  function integer area;
    return length * width;
  endfunction
endclass

class square_t extends rectangle_t;

  function new(int side);
    super.new(side,side);
  endfunction

endclass

module class_example ;
  rectangle_t rectangle;
  square_t    square;
  initial begin
    rectangle = new(50,20);
    $display("rectangle area: %0d", rectangle.area);
    square = new(50);
    $display("square area: %0d", square.area);
  end
endmodule
```

A Square is a Rectangle with the same length and width

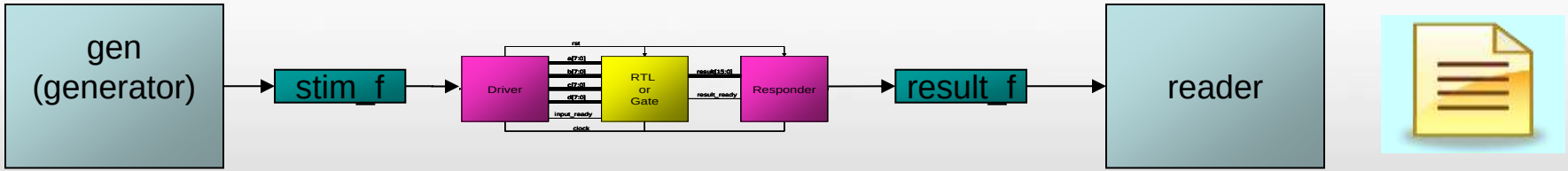
Leverage rectangle_t data and methods

In this case, constructor (new) calls parent's constructor

We can use the method from the parent

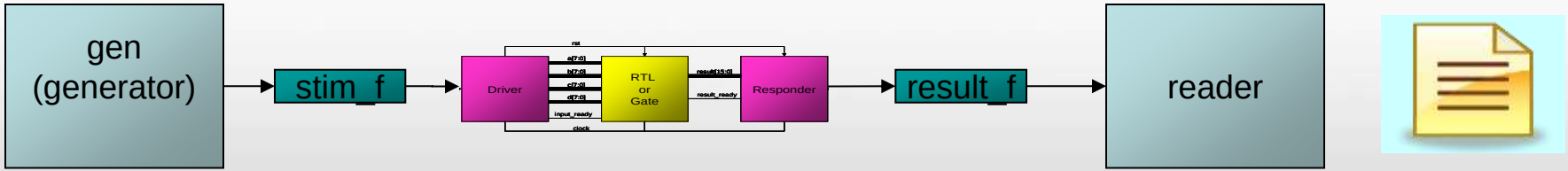
What Can I Do with Objects?


```
function void calc_beh_result (input_tran t);
  result = (t.a * t.b) + (t.c * t.d);
endfunction // void
```



What Can I Do with Objects?


```
function void calc_beh_result (input_tran t);
  result = (t.a * t.b) + (t.c * t.d);
endfunction // void
```



Randomizing Objects

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

Random Variables

Randomizing

```
28 module top;
29    mem_op op;
30
31    initial begin
32       op = new();
33       repeat (5) begin
34          assert(op.randomize());
35          $display("op ->  %s",op.convert2string);
36       end
37    end
38 endmodule // top
```

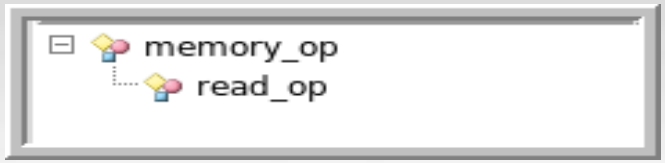
```
# Loading work.top
# op ->  addr b619  data 88   op write
# op ->  addr 3900  data 3b   op write
# op ->  addr c1f2  data 50   op write
# op ->  addr 0309  data 7d   op  read
# op ->  addr 7e69  data 87   op  read
VSIM 3>
```

Constraining Randomization

```
27    initial begin
28       op = new();
29       repeat (5) begin
30          assert(op.randomize() with {mem_op == read;});
31          $display("op ->  %s",op.convert2string);
32       end
33    end
34 endmodule // top
```

```
VSIM 1> run -all
# op ->  addr aba6  data 89   op  read
# op ->  addr 0377  data 75   op  read
# op ->  addr 85a5  data 81   op  read
# op ->  addr 31c0  data 9c   op  read
# op ->  addr 1924  data dc   op  read
```

Add Constraints by Extending Classes




```
24 class read_op extends memory_op;
25    constraint read_only_c {mem_op == read;};
26 endclass // read_op
```

```
28 module top;
29    read_op op;
30
31    initial begin
32       op = new();
33       repeat (5) begin
34          assert(op.randomize());
35          $display("op ->  %s",op.convert2string);
36       end
37    end
```

```
VSIM 1> run -all
# op ->  addr aba6  data 89   op  read
# op ->  addr 0377  data 75   op  read
# op ->  addr 85a5  data 81   op  read
# op ->  addr 31c0  data 9c   op  read
# op ->  addr 1924  data dc   op  read
```

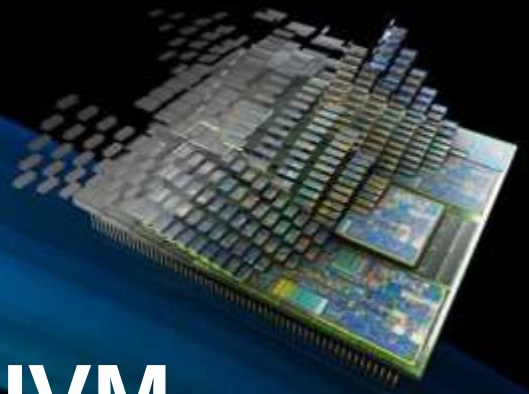
Summary

- SystemVerilog uses classes to encapsulate data and functionality
- You can create families of classes. Child classes inherit functionality from their parents.
- SystemVerilog can randomize objects. You can control the randomization with constraints.

Next Session

1. SystemVerilog for VHDL Engineers
2. Object Oriented Programming
3. SystemVerilog Interfaces
4. Packages, Includes, and Macros
5. UVM Test Objects
6. UVM Environments
7. Connecting Objects
8. Transaction Level Testing
9. The Analysis Layer
10. UVM Reporting
11. Functional Coverage with Covergroups
12. Introduction to Sequences

Verification Academy



Introduction to the UVM

Object Oriented Programming

Ray Salemi

Senior Verification Consultant

info@verificationacademy.com | www.verificationacademy.com

Mentor Graphics