The American University in Cairo

CENTER OF NANOELECTRONICS & DEVICES

CND212: Digital Testing and Verification

SV OOP Program Constructs

- Building SystemVerilog OOP structure vs. building a Verilog RTL structure

	RTL	OOP
Block definition	Module	Class
Block instance	Instance	Object
Block name	Instance name	Handle
Data types	Reg, wire, logic, bit...	Properties (variables)
Functionality	Tasks and functions Behavioral blocks: always/initial	Methods (subroutines): Tasks/functions
Communication between blocks	Ports or interfaces between modules	Task/ function calls, mailboxes, semaphores…

SV OOP Key Concepts

- **Encapsulation:** Creating containers of data along with the associated behaviors.
- **Inheritance:** Hiding the implementation details to reduce the complexity and raise the abstraction level.
- **Polymorphism** is the reuse of the same code to take on many different behaviors based on the type of object at hand.

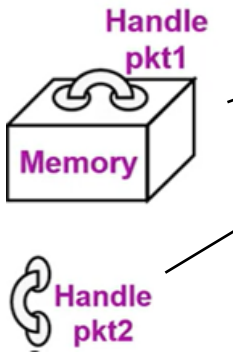
class - (OOP Encapsulation)

- A Class is a type and an OOP construct that **encapsulates**:
  - Variables (**properties**) used to model a system
  - Subroutines (**methods**) to manipulate the data
- **Properties** and **methods** are called **members** of a class
- The class **properties** and **methods**, taken together, define the contents and capabilities of some **object**

```
typedef enum [IDLE, RESET, P1, ...] cmd_t;

class Packet;

   /// Properties
   cmd_t command;
   int status;
   logic [7:0] data [0:255];

   /// Methods
   function int GetStatus();
      return(status);
   endfunction: GetStatus

   task SetCommand (input cmd_t X);
      command = X;
   endtask: SetCommand

endclass: Packet
```

class instance (object)

- OOP objects are built from class definitions
  - An object is an instance of that class (similar to module instances)
- An **object** can be used when you:
  1) Declare a variable of that class type (that holds an **object handle**)
  2) Create an **object** of that class and allocate memory for it (using the ***new()*** method)
- Uninitialized object handles are set by default to the special value null.
- Object members are accessed using the object handle
  - Accessed via the (.) notation




```
program packet_tb;
    Packet pkt1 = new(); //handle (pkt1) + memory allocation
    Packet pkt2;         //handle (pkt2)
    Packet pkt3;         //handle (pkt3)

    initial begin
        pkt3 = new(); // object memory allocation refered to by pkt3 handle
        pkt2.data = 0;  //runtime error (accessing null handle)
    end
endprogram
```

Initialization of Object Members (class constructor)

- Define constructor *new()* in class to initialize properties
  - Function with no return type
  - Executes immediately after object memory is allocated
  - Not accessible via the dot (.) notation
- function new() gives default values to the arguments so that any object created will have these default values unless overridden.
- *new()* is now being used in two very different contexts
  - Object instance
  - Object members' initialization
- Calling the class's constructor without explicitly defining its new method will call its default built-in new method

```
class packet;
  //class properties
  bit [31:0] addr;
  bit [31:0] data;
  bit  write;
  string pkt_type;
  //constructor
  function new();
    addr  = 32'h10;
    data  = 32'hFF;
    write = 1;
    pkt_type = "GOOD_PKT";
  endfunction

  //method to display class prperties
  function void display();
    $display("-------------------------");
    $display("\t addr  = %0d",addr);
    $display("\t data  = %0h",data);
    $display("\t write = %0d",write);
    $display("\t pkt_type  = %0s",pkt_type);
    $display("-------------------------");
  endfunction
endclass

module sv_constructor;
  packet pkt;
  initial begin
    pkt = new();
    pkt.display();
  end
endmodule
```

Initialization of Object Members - class constructor

- If a class does not provide an explicit user-defined *new* method, an implicit new method shall be provided automatically

```
class myPacket;
  logic [2:0] header;
  bit       encode;
  bit [2:0] mode;   // Class Variables (Properties)
  bit [7:0] data;
  logic       stop;

//   function new (bit [2:0] header = 3'h1, bit [2:0] mode = 5); // Class Method
//       this.header = header;
//     this.encode = 0;
//     this.mode   = mode;
//     this.stop   = 1;
//   endfunction

  function void display (); // Class Method
    $display ("Header = 0x%0h, Encode = %0b, Mode = 0x%0h, Stop = %0b", this.header, this.encode, this.mode,
this.stop);
  endfunction

endclass
```

"this" keyword is used to refer to the current class and used inside the class to refer to its own variables and methods.

```
module myPacket_tb;
  initial
    begin
      myPacket myPkt = new (); // Creating class handle and using it to construct and object in a
single line of code
      myPkt.display (); // Calling the class method using the class handle
    end
endmodule
```

```
Header = 0xx, Encode = 0, Mode = 0x0, Stop = x
                V C S   S i m u l a t i o n   R e p o r t
```

Initialization of Object Members - Custom Constructor

- Upon the creation of the object, we can pass values to the constructor

```
1  class myPacket;
2    logic [2:0] header;
3    bit       encode;
4    bit [2:0] mode;   // Class Variables (Properties)
5    bit [7:0] data;
6    logic       stop;
7
8  //   function new (bit [2:0] header = 3'h1, bit [2:0] mode = 5); // Class Method
9  //       this.header = header;
10 //     this.encode = 0;
11 //     this.mode   = mode;
12 //     this.stop   = 1;
13 //   endfunction
14
15   function void display (); // Class Method
16     $display ("Header = 0x%0h, Encode = %0b, Mode = 0x%0h, Stop = %0b", this.header, this.encode, this.mode,
   this.stop);
17   endfunction
18
19 endclass
```

```
1  module myPacket_tb;
2
3    myPacket myPacket0;
4
5    initial
6      begin
7        myPacket0 = new (3'h2, 2'h1);
8        myPacket0.display ();
9      end
10
11 endmodule
```

```
Header = 0x2, Encode = 0, Mode = 0x1, Stop = 1
              V C S   S i m u l a t i o n   R e p o r t
```

Creating an array of classes

- An array of classes can be created in a way similar to how you create, for example, an int array type

```
22 module tb_top ();
23   myPacket pkt [3];
24   initial
25     begin
26       for (int i =0; i < $size (pkt); i++)
27         begin
28           pkt [i] = new ();
29           pkt [i].display();
30         end
31     end
32 endmodule
```

```
Class Properties are: Header = 0, Encode = 1, Mode = 2, Data = 0, Stop = 0
Class Properties are: Header = 0, Encode = 1, Mode = 2, Data = 0, Stop = 0
Class Properties are: Header = 0, Encode = 1, Mode = 2, Data = 0, Stop = 0
                V C S   S i m u l a t i o n   R e p o r t
```

Inheritance and subclasses

- In a previous example a class called *myPacket* was created. This class can be extended to add more properties and/or methods
- In other words, a subclass is created that **inherits** the base class without re-implementing its properties and methods.
- To call the methods of the base class inside the inherited one, use the keyword "*super*"

```
21 class networkPkt extends myPacket;
22   bit parity;
23   bit [1:0] crc;
24
25   function new ();
26     super.new();
27     this.parity = 1;
28     this.crc = 3;
29   endfunction
30
31   function void display ();
32     super.display();
33     $display ("Parity = %b, CRC = 0x%0h", this.parity, this.crc);
34   endfunction
35
36 endclass
```

```
Header = 0x1, Encode = 0, Mode = 0x5, Stop = 1
Parity = 1, CRC = 0x3
              V C S   S i m u l a t i o n   R e p o r t
```

Polymorphism - Abstract/Virtual Class

- Polymorphism means having many forms.
- It is the ability to have the same code act differently for different types of objects.
- An abstract/virtual class is a class you cannot create an object from.
- It’s useful if you want to force other users to not create an object of your class and rather extend it.
- The keyword “virtual” is used to create an abstract class.

Polymorphism – Virtual Methods

- Virtual class methods allow accessing different code implementations during runtime.
- Overriding virtual methods must have the same prototype.
- Declaring a method as virtual always means it is virtual in all extended classes

```
typedef enum {IDLE, RUN, P0, P1} cmd_t;

///// Base Class Declaration
class Packet;

 /// Properties
 cmd_t cmd;
 int status;
 bit [7:0] data [0:255];

 /// Method
 virtual function void SetStatus (input int y);
 status = y;
 endfunction: SetStatus

endclass: Packet
```

```
///// Extended Class Declaration
class myPacket extends Packet;

 /// Added Properties
 bit errBit;

 /// Newly Added Method
 function bit ShowError();
   return(errBit);
 endfunction: ShowError

 /// Overriding Method
 virtual function void SetStatus (input int y);
   status = y + 2;
 endfunction: SetStatus

endclass: myPacket
```

```
module top;

 Packet pkt = new;
 myPacket m_pkt = new;

 task my_run (Packet PKT);

   PKT.SetStatus(2);
   $display("Status value is = %0d", PKT.status);
   endtask: my_run

 initial begin
   my_run(pkt);
   my_run(m_pkt);
 end

 endmodule: top
```

```
Status value is = 2
Status value is = 4
          V C S   S i m u l a t i o n   R e p o r t
```

Polymorphism

```
// base class
    class base_class;
      virtual function void display();
        $display("Inside base class");
      endfunction
    endclass

// extended class 1
    class ext_class_1 extends base_class;
      function void display();
        $display("Inside extended class 1");
      endfunction
    endclass

// extended class 2
    class ext_class_2 extends base_class;
      function void display();
        $display("Inside extended class 2");
      endfunction
    endclass

// extended class 3
    class ext_class_3 extends base_class;
      function void display();
        $display("Inside extended class 3");
      endfunction
    endclass
```

```
// module
    module class_polymorphism;

      initial begin

        //declare and create extended class
        ext_class_1 ec_1 = new();
        ext_class_2 ec_2 = new();
        ext_class_3 ec_3 = new();

        //base class handle
        base_class b_c[3];

        //assigning extended class to base class
        b_c[0] = ec_1;
        b_c[1] = ec_2;
        b_c[2] = ec_3;

        //accessing extended class methods using base class handle
        b_c[0].display();
        b_c[1].display();
        b_c[2].display();
      end

    endmodule
```

```
Inside extended class 1
Inside extended class 2
Inside extended class 3
                V C S   S i m u l a t i o n   R e p o r t
```

Shallow & Deep Copy

- SystemVerilog deep copy copies all the class members and its nested class members.
- Unlike in shallow copy, only nested class handles will be copied.
- In shallow copy, Objects will not be copied, only their handles will be copied.
- To perform a full or deep copy, a custom method needs to be added. In the custom method, a new object is created, all the class properties will be copied to a new handle, and the new handle will be returned.

Shallow Copy - Example

- Shallow copy allocates the memory, copies the variables, and returns the memory handle.
- Changes made by A2 will not be reflected on A1.
- Shallow copy allows only the first level of Properties to be copied, not the Object inside the Parent class.

```
A1.b = 5
A2.b = 9
A1.Li.a = 100
A2.Li.a = 100
          V C S   S i m u l a t i o n   R e p o r t
```

```
class Lion;
    int a;
endclass: Lion

class Animal;
    int b = 5;
    Lion Li = new;
endclass: Animal

module top;
    Animal A1, A2;

    initial begin
        A1 = new();
        A2 = new A1;
        A2.b = 9;
        A2.Li.a = 100;

        $display ("A1.b = %0d", A1.b);
        $display ("A2.b = %0d", A2.b);
        $display ("A1.Li.a = %0d", A1.Li.a);
        $display ("A2.Li.a = %0d", A2.Li.a);
    end
endmodule: top
```

Deep Copy - Example

- Deep Copy is done by creating a copy() function.

```
A1.b = 5
A2.b = 9
A1.Li.a = 200
A2.Li.a = 100
          V C S   S i m u l a t i o n   R e p o r t
```

```
class Lion;
    int a = 200;
endclass: Lion

class Animal;
    int b = 5;
    Lion Li = new;
    function void copy (Animal Ani);
        this.b = Ani.b;
        this.Li = new Ani.Li;
    endfunction: copy
endclass: Animal

module top;
    Animal A1, A2;
    initial begin
        A1 = new();
        A2 = new();
        A2.copy(A1);
        A2.b = 9;
        A2.Li.a = 100;

        $display ("A1.b = %0d", A1.b);
        $display ("A2.b = %0d", A2.b);
        $display ("A1.Li.a = %0d", A1.Li.a);
        $display ("A2.Li.a = %0d", A2.Li.a);
    end
endmodule: top
```

module vs. class

- Why use class?
- Objects are dynamic, modules are static
  - Objects are created and destroyed as needed
- Instances of modules can not be passed, copied, or compared
  - Instances of classes are objects (class instance)
  - Object Handles can be passed as arguments
  - Object memory can be copied or compared
- Classes can be inherited, modules can not
  - Classes can be modified via inheritance without impacting existing users
  - Modifications to modules will impact all existing users

Lab Task (~ 60 min)

Lab Task

- Build a transceiver system that has two instances:
  - Transmitter
  - Receiver
- The transceiver class sends data from the transmitter to the receiver.
- The Transmitter and Receiver are extended from a base class comm_component.
- The comm_component class has the following properties:
  - Data
  - Address
- The comm_component class has the following methods:
  - Initialize
  - Display