CSCE 616 - Hardware Design Verification

Lab 3 lecture: HTAX Packet Class

Recap

- Create the testbench for simple-mem and cache-mem design
- Understand how the testbench is different that design
- Define tasks/functions for ease in creating stimulus
- Nuance of interface in TB .. Reduce code side and make it more readable
- Debug statements in TB
- Most Importantly – We verify the design to catch hold and fix the design BUGS

Objective for Lab-3

- OOP aspect of System-Verilog
- Understand the term Packet/Transaction
- Create packet class for our HTAX design (UVM compliant)
- Rand and Randc modifiers for class properties
- Randomize and Constraint the class properties
- Create class object with new() and create() functions

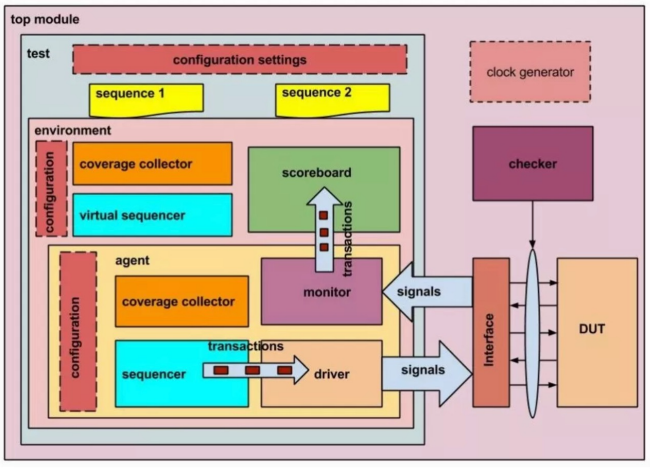
What is UVM?

The Universal Verification Methodology (UVM) is a standardized methodology for verifying integrated circuit designs.

Reusability

Scalability

Interoperability



OOP in System-Verilog

1. Encapsulation
   Creating class with members and functions
2. Inheritance
   Creating a sub-class from parent class
3. Abstraction
   Data hiding with local and protected
4. Polymorphism
   Same function with different behavior based on object

```
class shape;   //Parent Class
      local int area;
      int perimeter;
      function new (input int area=0, int peri=0) ;
            this.area = area;
            perimeter =peri;
      endfunction
      virtual function setArea(input int area)
            this.area = area;
      endfunction
endclass
class circle extends shape; //Child-Subclass
      int radius; --- followed by constructor
      function setArea ( input int rad) // Polym of setArea
            area = (3.14)*rad*rad;
      endfunction
endclass
```

Packet/Transaction – UVM sequence item

- Packet – Object from Packet class
  - To verify any DUT, packet forms the basis of any stimulus
  - Data & Address properties in Lab1&2 when clubbed into a single class object, that is our write packet

```
class my_pkt extends uvm_sequence_item;
   `uvm_object_utils(my_pkt)  // Factory registration

  randc logic [4:0] addr;
  rand logic [8:0] wdata;
//constructor
function new (string name = "my_txn");
  super.new(name);
endfunction: new

//constraints
endclass
```

Rand and Randc modifiers for class properties

Rand : Class properties with “rand” keyword are random variables with a uniform probability distribution over all the possible values.

Randc : Class properties with “randc” keyword are random-cyclic variables which loops through all the possible values in random permutation.

Eg. Values of 2 bit rand variable can be – 0 0 3 2 3 1 1 3 2 0 ….

Values of 2 bit randc variable can be – 2 3 0 1 1 3 2 0 ….

Randomize()

- Packet Handle :

  my_pkt trans1; //creating a handle name trans1 for object

- Creating a new packet – generates with constructor

  trans1 = new(); // normal object creation method

  trans1 = my_pkt::get_type_id::create() // factory registered object creation

Every class has a virtual predefined function randomize(), which is provided for generating a new value. Randomization function returns 1 if the solver finds a valid solution.

```
if(trans1.randomize())
   $display(" Randomization successful : address= %0d , wdata = %0f",trans1.addr, trans1.wdata);
```

Constraint class properties

Hard Constraint vs Soft Constraint

Equality -- Range – Distribution

Syntax –

```
class A;
  int x, y, z;
  //constraint for x between 10 and 100
  constraint <name> { soft/hard x inside [10:100];}

  //constraint for y in 0 to 50 with 80% probability and between 51 -100 with 20% probability
  constraint <name> {soft/hard y dist {[0:50]:/80, [51-100]:/20};}

  //constraint z is equal to 25
  constraint <name> {z == 25;}

endclass
```

Instantiation of Class Object

In system-verilog

<class object handle> = new (<arguments to constructor>);

E.g. pkt1 = new();

Concept of Factory in UVM (beyond the scope of this course)

<class object handle> = <class type>::type_id::create(<argu to constructor>)

Factory Registration and print()

`uvm_object_utils_begin(htax_packet_c)

…

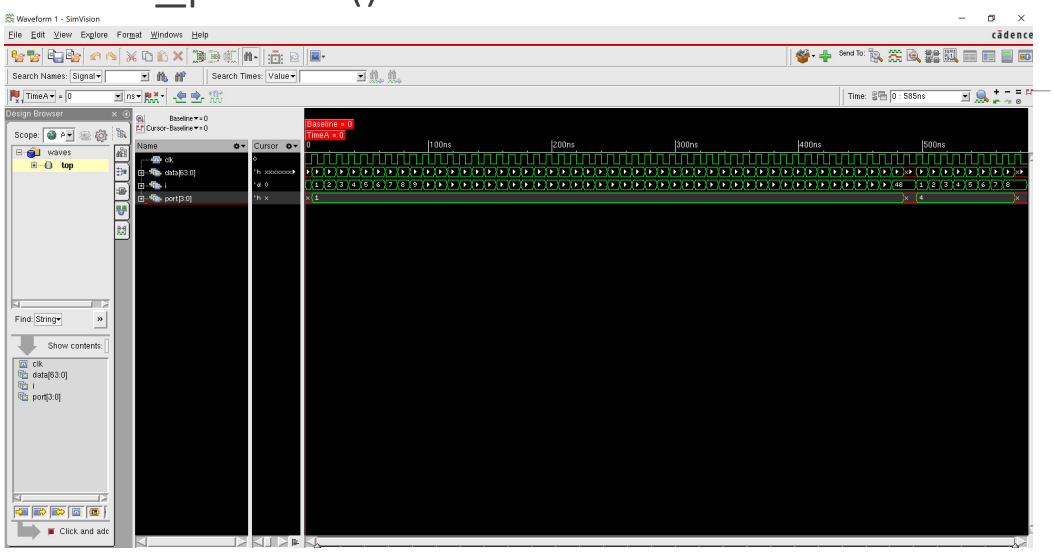
`uvm_object_utils_end

This part of code registers the htax_packet_c. The "`uvm_field_....." allows us to use and inbuilt function print() to print any object of this class.

For e.g. pkt is the instance of htax_packet_c then

- pkt.print() will print the contents of pkt.

drive_packet() task in lab-3



Thank You