Virtual Interfaces

- A pointer to an actual *interface*
- Allow classes to access the signals in the interface
- Provides a mechanism for separating abstract models and test programs from the actual signals that make up the design.
- Allows the same subprogram to operate on different portions of a design and to dynamically control the set of signals associated with the subprogram.
- Instead of referring to the actual set of signals directly, users are able to manipulate a set of virtual signals.
- Changes to the underlying design do not require the code using virtual interfaces to be rewritten.
- By abstracting the connectivity and functionality of a set of blocks, virtual interfaces promote code reuse.

Virtual Interfaces

- The interface is used to simplify the connection between DUT and Testbench.

- But the interface can't be instantiated inside a class or program block, we need a virtual interface to point the physical interface.

- So, the virtual interface is a pointer to the actual interface and using virtual interface, a class can point to different physical interfaces, dynamically (at run time).

Virtual Interfaces - Example

```
module adder(input  [3:0] a, input  [3:0] b,
             output [7:0] c);

  assign c = a + b;

endmodule
```

```
interface intf();

  //declaring the signals
  logic [3:0] a;
  logic [3:0] b;
  logic [7:0] c;

endinterface
```

```
class environment;

  //virtual interface
  virtual intf vif;

  //constructor
  function new(virtual intf vif);
    //get the interface from test
    this.vif = vif;
  endfunction

  //run task
  task run;
    vif.a = 6;
    vif.b = 4;

    $display("Value of a = %0d, b = %0d",vif.a,vif.b);
    #5;
    $display("Sum of a and b, c = %0d",vif.c);
    $finish;
  endtask

endclass
```

Virtual Interfaces - Example

```
`include "environment.sv"

  program test(intf i_intf);

  //declaring environment instance
  environment env;

  initial begin
    //creating environment
    env = new(i_intf);

    //calling run of env
    env.run();
  end

endprogram
```

```
`include "test.sv"
`include "interface.sv"

module tbench_top;

  //creatinng instance of interface
  intf i_intf();

  //Testcase instance
  test t1(i_intf);

  //DUT instance, interface signals
  //are connected to the DUT ports
  adder DUT (
    .a(i_intf.a),
    .b(i_intf.b),
    .c(i_intf.c)
   );

endmodule
```

Randomization

- Use : to generate random values for the inputs in class
- SV provides randomize() method
- rand - rand bit [7:0] data; - the value will be any value between 0 to 255
- randc – random-cyclic - variable values don't repeat a random value until every possible value has been assigned.
- Call the method randomize() method for variables randomization
- Following types can be declared as rand and randc,
  - singular variables of any integral type
  - arrays
  - arrays size
  - object handle's

Randomization - Example

```
module random_example;
  class random_class;
    randc bit [2:0] address;
          bit [2:0] data;
  endclass

  random_class r1 = new;

  initial begin
    repeat(10) begin
      r1.randomize();
      $display($time, " address = %0d", r1.address);
      #1;
    end
  end

endmodule
```

```
0 address = 3
1 address = 7
2 address = 1
3 address = 4
4 address = 6
5 address = 0
6 address = 2
7 address = 5
8 address = 6
9 address = 3
```

```
  class random_class;
    rand  bit [2:0] address;
          bit [2:0] data;
  endclass
```

```
0 address = 7
1 address = 0
2 address = 5
3 address = 6
4 address = 7
5 address = 1
6 address = 2
7 address = 7
8 address = 3
9 address = 3
```

Randomization - Example

```
class bus;
  rand bit [7:0]  address;
  rand bit [15:0] data;
  rand bit        wr_rd;
  constraint align {address[7:6] == 2'b00;}
endclass

module random_constrained;

  bus b1 = new;

  initial begin
    repeat(10) begin
      if(b1.randomize())
        $display("address=%b, data=%0d, wr_rd=%b", b1.address,b1.data,b1.wr_rd);
      else
        $display("Randomization Failed");
    end
  end

endmodule
```

```
address=00110001, data=16932, wr_rd=0
address=00101000, data=42689, wr_rd=1
address=00000101, data=32919, wr_rd=1
address=00000011, data=14821, wr_rd=1
address=00101001, data=56236, wr_rd=1
address=00111001, data=34884, wr_rd=1
address=00100101, data=27844, wr_rd=1
address=00100110, data=63100, wr_rd=0
address=00010001, data=5187, wr_rd=1
address=00100111, data=19599, wr_rd=1
```

- randomize() – is a virtual function, called to generate new random values

Randomization - Methods

- Every class contains
  - pre_randomize() – settings prior to randomization
  - post_randomize() – settings after to randomization
- If randomize() fails, post_randomize() will not be called
- rand_mode() – enable or disable ransomization of any variable
- constraint_mode() – enable or disable any constraint block
- srandom() – s for seed - create same set of random numbers each time the randomization is called
- Constraint blocks – to set constraints on the random number generator (say, a>b)
  - Set membership – inclusive, exclusive
  - Distribution – weight can be applied to the random numbers
  - Implication – to provide conditional relations
  - Iterative – foreach construct can be used to iterate over constrained block

Randomization - Methods

- Variable ordering
  - solve x before y
- In-line constraints
- Random number – functions
- urandom(int seed_value) – 32 bit unsigned integer
- urandom_range(max, min) - unsigned integer within a range

Interprocess synchronization

- Non-blocking event trigger (->>)
- Mailbox
- Semaphore

Mailbox

- A communication channel, allows messages to be exchanged between processes
- Built in methods are
  - new()
  - num()
  - put()
  - try_put()
  - get()
  - try_get()
  - peek()
  - try_peek()

Mailbox - Example

```
class my_frame #(a_width=8, d_width=16);

  rand bit [a_width - 1 : 0] address;
  rand bit [d_width - 1 : 0] write_data;
  rand bit [d_width - 1 : 0] read_data;
  rand bit t_type;

endclass
```

```
class my_generator;
  my_frame gen_f1;
  mailbox gen_mailbox;

  function new(mailbox gen_mailbox);
    this.gen_mailbox = gen_mailbox;
  endfunction

 //generator task
 task run(int count);
   begin
     repeat(count) begin
       gen_f1 = new();
       assert(gen_f1.randomize);
       gen_mailbox.put(gen_f1);
       $display("Generator: Entries in mailbox=%0d,address=%0h,write_data=%0h,read_data=%0h,transfer_type=%b",
                 gen_mailbox.num(),gen_f1.address,gen_f1.write_data, gen_f1.read_data,gen_f1.t_type);
       end
     end
   endtask

 endclass:my_generator
```

Mailbox - Example

```
`include "my_frame.sv"
`include "my_generator.sv"
`include "my_driver.sv"

module mailbox_top;
  mailbox top_mailbox = new;
  my_generator g1 = new(top_mailbox);
  my_driver d1 = new(top_mailbox);

  initial begin
    g1.run(2);
    d1.run(2);
  end

endmodule
```

```
class my_driver;
  my_frame drv_f1;
  mailbox drv_mailbox;

  function new(mailbox drv_mailbox);
    this.drv_mailbox = drv_mailbox;
  endfunction

//driver task
task run(int count);
  begin
    drv_f1 = new();
    repeat(count) begin
      drv_mailbox.get(drv_f1);
      $display("Driver: Entries in mailbox=%0d,address=%0h,write_data=%0h,read_data=%0h,transfer_type=%b",
                drv_mailbox.num(),drv_f1.address,drv_f1.write_data, drv_f1.read_data,drv_f1.t_type);
    end
  end
endtask

endclass:my_driver
```

Mailbox - Example

Generator: Entries in mailbox=1,address=b8,write_data=5f08,read_data=9379,transfer_type=0

Generator: Entries in mailbox=2,address=56,write_data=60ea,read_data=7f85,transfer_type=0

Driver: Entries in mailbox=1,address=b8,write_data=5f08,read_data=9379,transfer_type=0

Driver: Entries in mailbox=0,address=56,write_data=60ea,read_data=7f85,transfer_type=0

Semaphores

- Semaphore is a SystemVerilog built-in class, used for access control to shared resources, and for basic synchronization.
- Methods

- new(); Create a semaphore with a specified number of keys
- get(); Obtain one or more keys from the bucket
- put(); Return one or more keys into the bucket
- try_get(); Try to obtain one or more keys without blocking

Semaphores - Example

```
module my_semaphore;
  semaphore my_sphore=new(1);
    initial begin
      repeat(2) begin
        fork begin
          $display($time,"ns Process (1) waiting");
          my_sphore.get(1);
          $display($time,"ns Process (1) has key");
          #10;
          my_sphore.put(1);
          $display($time,"ns Process (1) completed, returning key");
        end

        begin
          $display($time,"ns Process (2) waiting");
          my_sphore.get(1);
          $display($time,"ns Process (2) has key");
          #10;
          my_sphore.put(1);
          $display($time,"ns Process (1) completed, returning key");
        end

      join
    end
   #100;
   $display($time, "ns End of the Process");
 end
endmodule:my_semaphore
```

Semaphores - Example

0ns Process (1) waiting
0ns Process (1) has key
0ns Process (2) waiting
10ns Process (1) completed, returning key
10ns Process (2) has key
20ns Process (1) completed, returning key
20ns Process (1) waiting
20ns Process (1) has key
20ns Process (2) waiting
30ns Process (1) completed, returning key
30ns Process (2) has key
40ns Process (1) completed, returning key
140ns End of the Process

Assignments

Write a SystemVerilog program and simulate

1. To define an event and detect an event in other process

2. Store your full name in a string and find the length

3. Declare enumeration type for storing states, IDLE, SETUP, ACCESS and print the values

4. Using Struct, write a program to store 8-bit address, 16-bit data_in, 32-bit data_out, read_write. Assign respective values to the struct variable and print?

5. Demonstrate with an example – fork..join, join_any, join_none

6. Write a task to take inputs 8-bit address, 16-bit data_in, 32-bit data_out, read_write, and and output 32-bit dataout, and print the values

7. Write a function to take inputs 8-bit address, 16-bit data_in, 32-bit data_out, read_write, and and output 32-bit dataout, and print the values

8. In a module, declare a packed array of 16-bit data and array size 128.

9. In a module, declare an unpacked array of 8-bit data and array size 64

10.Demonstrate with an example for each constructs – foreach, iff, packages, program block, final block, clocking block