4 OOP Encapsulation

Learning Objectives

After completing this lab, you should be able to:

- Encapsulate packet information into a **`Packet`** class
- Utilize **randomization** in **`Packet`** class to randomly generate source address, destination address and payload
- Create two **`Packet`** objects, one for the input into the DUT, the other for reconstructing the output of the DUT
- Use the `compare()` method embedded in the **`Packet`** objects to verify the correctness of DUT operation

Lab Duration:
60 minutes

Getting Started

In Lab 3, you added the monitor and self-check. In this lab, you will encapsulate the packet information into a class structure. You will create random `Packet` objects in the generator then send, receive and check the correctness of the DUT using these `Packet` objects.

The architecture is shown below:

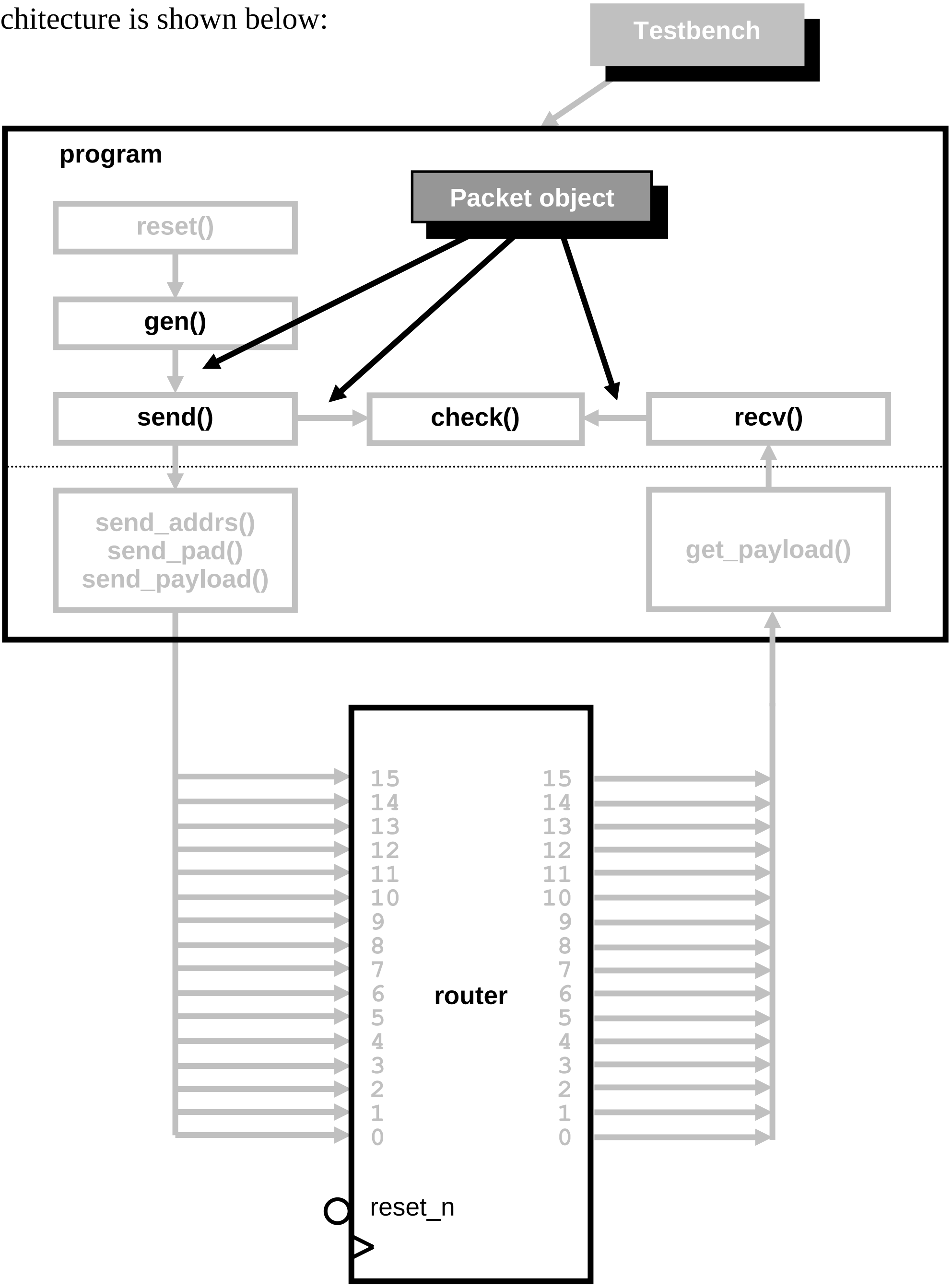


Figure 1. Lab 4 testbench architecture

Lab Overview

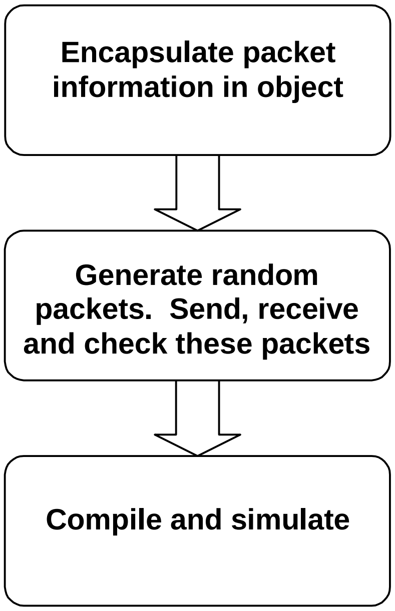

Encapsulate packet information in object
Generate random packets. Send, receive and check these packets
Compile and simulate

Figure 2. Diagram of Lab Exercise

Note: You will find Answers for all questions and solutions in the Answers / Solutions at the end of this lab.

Working with Objects

Task 1. Copy Files from Lab 3's Solutions directory

1. Go into the lab4 directory.

```
> cd ../lab4
```

2. Copy the source files in the **`solutions/lab3`** directory into the current directory with **`make`** script.

```
> make copy
```

(If you chose to use your own lab files from lab3, type "**`make mycopy`**".)

Task 2. Create a Packet Class File

Create a Packet class to encapsulate the packet information.

1. Open the **`Packet.sv`** file in an editor.

2. Declare a class definition for Packet as follows:

```
class Packet;

endclass: Packet
```

3. Use macros as a guard against multiple compilation before and after the class statements.

```
`ifndef INC_PACKET_SV
`define INC_PACKET_SV
class Packet;

endclass: Packet
`endif
```

4. In the body of the **Packet** class, create the following properties:

- **`rand bit[3:0] sa, da;`** // random port selection
- **`rand logic[7:0] payload[$];`** // random payload array
- **`string name;`** // (see description below)

Individual test **`Packet`** objects will be created by the **`gen()`** routine. For these **`Packet`** objects, you will want to tag each one uniquely to identify the object.

The string property **name** is the unique identifier. Displaying the **name** property as part of the printed message will be very helpful during the debugging process.

Task 3. Define Packet Property Constraints

1. Immediately after the property declarations, add a constraints block to limit **sa** and **da** to **0:15**, and the **payload.size()** to **2:4**.

Task 4. Define Packet Class Method Prototypes

1. Add the following method prototype declarations in the body of Packet class:

```
class Packet;
  ...
  extern function new(string name = “Packet”);
  extern function bit compare(Packet pkt2cmp, ref string message);
  extern function void display(string prefix = “NOTE”);
  extern function Packet copy();
endclass: Packet
```

Task 5. Define Packet Class `new()` Constructor

The class constructor **new()** is used to initialize object properties. For **Packet** objects, most properties will be set with a call to **randomize()**. The one property that needs to be initialized in the constructor is **name**.

1. Outside of the class body, create the constructor **new()** method. Make sure to reference the method back to the class via the **Packet::** notation.
2. Inside the constructor body, assign the class property **name** with the string passed in via the argument:

```
function Packet::new(string name);
  this.name = name;
endfunction: new
```

This mechanism makes the string passed in through the constructor argument accessible to all other methods of the **Packet** class.

Task 6. Define Packet `compare()` Method

For self checking, comparing contents of two data objects is a common need. It is a good idea to build the compare method within the data object.

1. Cut and paste **`compare()`** from **`test.sv`** into the **`Packet`** class at the end of the file.

Note: An alternative to cut and paste is to concatenate (cat) content of **`test.sv`** into **`Packet.sv`** then keep only **`compare()`** in **`Packet.sv`**.

2. Reference the method to **`Packet`** class with **`::`** notation.

3. Modify the **argument list** to contain a **`Packet`** handle:

```
function bit Packet::compare(Packet pkt2cmp, ref string message);
```

4. Inside the **`compare()`** method, change **`pkt2cmp_payload`** to reference the class property **`pkt2cmp.payload`**.

Task 7. Define Packet `display()` Method

It is helpful during the debugging process to print out the contents of a packet. To ease this effort, a display method should be defined for the Packet class to print the content of the Packet object to the console.

1. Outside of the class body, create the **`display()`** method:

```
function void Packet::display(string prefix = “NOTE”);
```

2. Inside the method body, print a formated content of the object to terminal.

In the printed message, you should also include the string passed via the argument list. This string can be set by user to differentiate between different types of message: ERROR, WARNING, DEBUG, etc.

Or, you can just use the display routine saved in the **`.display`** file in solutions directory (**`` `endif ``** must be after the inserted code):

```
> cat ../../solutions/lab4/.display >> Packet.sv
```

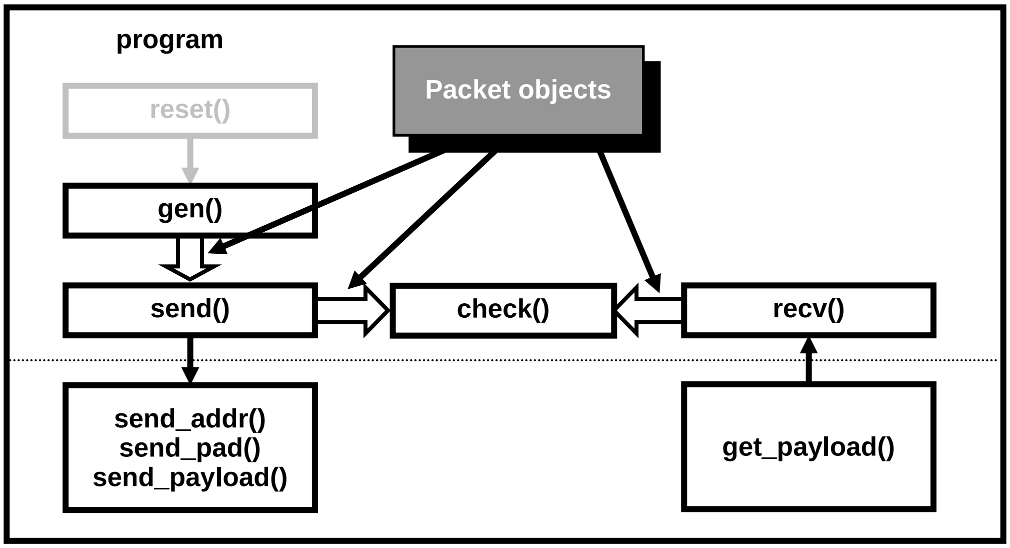
3. The **`copy()`** method is already written for you.

4. Save and close the file.

Task 8. Modify `test.sv` to use Packet class

The **`Packet`** class now encapsulates the router packet information.

You will now generate random **`Packet`** objects, then send and receive packets through the router based on these random **`Packet`** objects.



The following steps take you through this process.

1. Open **`test.sv`** in an editor.
2. Inside the program block, add an include statement for the **`Packet`** class file.
3. **Create** and **construct** two program global **`Packet`** objects **`pkt2send`** and **`pkt2cmp`**.

Task 9. Modify `gen()` Task To Generate Packet objects

You will now use the `Packet` objects just created to generate random stimulus.

1. In the **`gen()`** task, delete all existing code.
2. Declare a static int **`pkts_generated`** variable (initialize to 0) to keep track of how many packets were generated by the generator
3. Set the **`name`** property of **`pkt2send`** to a unique string (use the **`pkts_generated`** variable value as part of the string)
4. Randomize the `Packet` object **`pkt2send`** (print an error message to terminal and end simulation if randomization fails)
5. Update all program global variables, **`sa`**, **`da`** and **`payload`** with values from the randomized **`pkt2send`** object.

This makes the content of **pkt2send** object visible to all components of the program. These program global variables are only an interim solution to simplify development of subroutines in the program block. In the next lab, these variables will all be deleted and migrated inside individual testbench component objects.

When done, your code might look something like the following:

```
task gen();
  static int pkts_generated = 0;
  pkt2send.name = $sformatf("Packet[%0d]", pkts_generated++);
  if (!pkt2send.randomize()) begin
    $display("\n%m\n[ERROR]%t Randomization Failed!", $realtime);
    $finish;
  end
  sa = pkt2send.sa;
  da = pkt2send.da;
  payload = pkt2send.payload;
endtask: gen
```

Task 10. Modify recv() task

In the **recv()** task, the payload sampled from the output of the router needs to be assembled into a Packet object (**pkt2cmp**). This Packet object will then be used in the checking process against the **pkt2send** object.

In **recv()** task:

Before calling the get_payload() _task do the following:_

1. Create a static int variable **pkt_cnt** to track the number of packets received (should be initialized to 0).

After the get_payload() _task do the following:_

2. Assign **pkt2cmp.da** with the value of program global variable **da**.
3. Assign **pkt2cmp.payload** with the values from **pkt2cmp_payload** array.
4. Set a unique **name** for the **pkt2cmp** object. (use **pkt_cnt** value as part of the string).

When finished, the **`recv()`** task should look like:

```
task recv();
  static int pkt_cnt = 0;
  get_payload();
  pkt2cmp.da = da;
  pkt2cmp.payload = pkt2cmp_payload;
  pkt2cmp.name = $sformatf("rcvdPkt[%0d]", pkt_cnt++);
endtask: recv
```

Task 11. Modify the `check()` function

In the **`check()`** function, you will use the **`compare()`** method built into the **`Packet`** object to verify the content of the **`Packet`** object sent and received. You should use the **`display()`** method in the **`Packet`** object to assist in debugging errors.

1. Replace the **`compare()`** call with a call to the **`compare()`** method within the **`Packet`** object. (The two objects you want to compare are the program global objects **`pkt2send`** and **`pkt2cmp`**).

2. Make use of the **`display()`** method when an error is detected.

When finished, the **`check()`** routine should look like:

```
function void check();
  string message;
  static int pkts_checked = 0;
  if (!pkt2send.compare(pkt2cmp, message)) begin
    $display("\n%m\n[ERROR]%t Packet#%0d %s\n", $realtime, pkts_checked,
message);
    pkt2send.display("ERROR");
    pkt2cmp.display("ERROR");
    $finish;
  end
  $display("[NOTE]%t Packet#%0d %s", $realtime, pkts_checked++, message);
endfunction: check
```

Task 12. Check and Save file

1. Make sure you have deleted the `compare()` routine in `test.sv`.
2. Save and close the file.

Task 13. Compile and Run

1. Use **`make`** script to compile and run your program.

   ```
   > make
   ```

   Debug any error you find.

2. If the testbench runs successfully, execute the following script which runs the testbench on a bad RTL code.

   ```
   > make bad
   ```

   If the simulation finds an error, you are done.

   If the simulation does not find an error, you have a problem with your testbench. You must debug the error in your testbench.

Congratulations, you completed Lab 4!

Answers / Solutions

`test.sv` Solution:

```
program automatic test(router_io.TB rtr_io);
  // The following program variables will be seen by the included files without
extern
  int run_for_n_packets;    // number of packets to test
  `include "Packet.sv"

  // The following program variables can be seen by the included files with
extern
  bit[3:0] sa;           // source address
  bit[3:0] da;              // destination address
  logic[7:0] payload[$];        // expected packet data array
  logic[7:0] pkt2cmp_payload[$];       // actual packet data array
  Packet   pkt2send = new();    // expected Packet object
  Packet   pkt2cmp = new();     // actual Packet object

  initial begin
    $vcdpluson;
    run_for_n_packets = 2000;
    reset();
    repeat(run_for_n_packets) begin
      gen();
      fork
        send();
        recv();
      join
      check();
    end
    repeat(10) @rtr_io.cb;
  end

  task reset();
    rtr_io.reset_n <= 1'b0;
    rtr_io.cb.frame_n <= '1;
    rtr_io.cb.valid_n <= '1;
    repeat(2) @rtr_io.cb;
    rtr_io.cb.reset_n <= 1'b1;
    repeat(15) @rtr_io.cb;
  endtask: reset

  task gen();
    static int pkts_generated = 0;
    pkt2send.name = $sformatf("Packet[%0d]", pkts_generated++);
    if (!pkt2send.randomize()) begin
      $display("\n%m\n[ERROR]%t gen(): Randomization Failed!", $realtime);
      $finish;
    end
    sa = pkt2send.sa;
    da = pkt2send.da;
    payload = pkt2send.payload;
  endtask: gen
```

Continued…

…Continued from previous page

```
task send();
  send_addrs();
  send_pad();
  send_payload();
endtask: send


task send_addrs();
  rtr_io.cb.frame_n[sa] <= 1'b0;
  for(int i=0; i<4; i++) begin
    rtr_io.cb.din[sa] <= da[i];
    @rtr_io.cb;
  end
endtask: send_addrs

task send_pad();
  rtr_io.cb.frame_n[sa] <= 1'b0;
  rtr_io.cb.valid_n[sa] <= 1'b1;
  rtr_io.cb.din[sa] <= 1'b1;
  repeat(5) @rtr_io.cb;
endtask: send_pad

task send_payload();
  foreach(payload[index]) begin
    for(int i=0; i<8; i++) begin
      rtr_io.cb.din[sa] <= payload[index][i];
      rtr_io.cb.valid_n[sa] <= 1'b0;
      rtr_io.cb.frame_n[sa] <= (index == (payload.size() - 1)) && (i == 7);
      @rtr_io.cb;
    end
  end
  rtr_io.cb.valid_n[sa] <= 1'b1;
endtask: send_payload

task recv();
  static int pkt_cnt = 0;
  get_payload();
  pkt2cmp.da = da;
  pkt2cmp.payload = pkt2cmp_payload;
  pkt2cmp.name = $sformatf("rcvdPkt[%0d]", pkt_cnt++);
endtask: recv
```

Continued…

```
  …Continued from previous page
  task get_payload();
    pkt2cmp_payload.delete();
    fork
      begin: wd_timer_fork
      fork: frameo_wd_timer
        begin //see class notes to understand this block
          wait(rtr_io.cb.frameo_n[da] != 0);
          @(rtr_io.cb iff(rtr_io.cb.frameo_n[da] == 0 ));
        end
        begin
          repeat(1000) @rtr_io.cb;
          $display("\n%m\n[ERROR]%t Frame signal timed out!\n", $realtime);
          $finish;
        end
      join_any: frameo_wd_timer
      disable fork;
      end: wd_timer_fork
    join
    forever begin
      logic[7:0] datum;
      for (int i=0; i<8; ) begin
        if (!rtr_io.cb.valido_n[da])
          datum[i++] = rtr_io.cb.dout[da];
        if (rtr_io.cb.frameo_n[da])
          if (i == 8) begin
            pkt2cmp_payload.push_back(datum);
            return;
          end
          else begin
            $display("\n%m\n[ERROR]%t Payload not byte aligned!\n", $realtime);
            $finish;
          end
        @rtr_io.cb;
      end
      pkt2cmp_payload.push_back(datum);
    end
  endtask: get_payload

  function void check();
    string message;
    static int pkts_checked = 0;
    if (!pkt2send.compare(pkt2cmp, message)) begin
      $display("\n%m\n[ERROR]%t Packet #%0d %s\n", $realtime, pkts_checked,
message);
      pkt2send.display("ERROR");
      pkt2cmp.display("ERROR");
      $finish;
    end
    $display("[NOTE]%t Packet #%0d %s", $realtime, pkts_checked++, message);
  endfunction: check

endprogram: test
```

`Packet.sv` Solution:

```
`ifndef INC_PACKET_SV
`define INC_PACKET_SV
class Packet;
  rand bit[3:0] sa, da;          //random port selection
  rand logic[7:0] payload[$];    //random payload array
       string   name;            //unique identifier
  constraint Limit {
    sa inside {[0:15]};
    da inside {[0:15]};
    payload.size() inside {[2:4]};
  }

  extern function new(string name = "Packet");
  extern function bit compare(Packet pkt2cmp, ref string message);
  extern function void display(string prefix = "NOTE");
  extern function Packet copy();
endclass: Packet

function Packet::new(string name);
  this.name = name;
endfunction: new

function bit Packet::compare(Packet pkt2cmp, ref string message);
  if (payload.size() != pkt2cmp.payload.size()) begin
    message = "Payload Size Mismatch:\n";
    message = { message, $sformatf("payload.size() = %0d,
pkt2cmp.payload.size() = %0d\n", payload.size(), pkt2cmp.payload.size()) };
    return(0);
  end
  if (payload == pkt2cmp.payload) ;
  else begin
    message = "Payload Content Mismatch:\n";
    message = { message, $sformatf("Packet Sent:  %p\nPkt Received: %p",
payload, pkt2cmp.payload) };
    return(0);
  end
  message = "Successfully Compared";
  return(1);
endfunction: compare

function void Packet::display(string prefix);
  $display("[%s]%t %s sa = %0d, da = %0d", prefix, $realtime, name, sa, da);
  foreach(payload[i])
    $display("[%s]%t %s payload[%0d] = %0d", prefix, $realtime, name, i,
payload[i]);
endfunction: display

function Packet Packet::copy();
  Packet pkt_copy = new();
  pkt_copy.sa = this.sa;
  pkt_copy.da = this.da;
  pkt_copy.payload = this.payload;
  return(pkt_copy);
endfunction

`endif
```