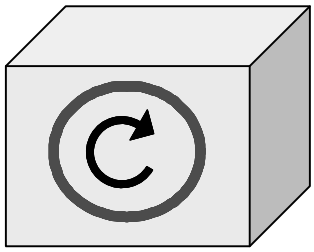
7 CUSTOM GENERATOR AND NOTIFICATIONS



In the previous chapters we addressed the use of predefined atomic transaction generators with the *`vmm_atomic_gen* macro. However, in some cases, you may want to write your own transaction generator to apply transactions to your specific requirements, such as the transactions with video data. This chapter explains the creation of a custom generator to create directed and random tests. The generator uses a factory pattern to allow the environment to select which transaction class to use for the selection of constraints. Chapter 8 has examples of the *`vmm_scenario_gen* macro that demonstrates the scenario definitions through the use of iterative constraints.

.

7.1 CUSTOM GENERATOR ARCHITECTURE

The *`vmm_atomic_gen* macro automatically creates the generator from the transaction class while the custom generator requires the user to create the code. The custom generator is a transactor that creates transactions. It puts those transactions into an output channel for use by another transactor. There are three important aspects of a transaction generator: the **transaction, the channel,** and the **notification of completion**. The transaction object is needed to create new transactions for transmission into the channel. For consistency with the VMM atomic generator, this custom generator uses a factory pattern to determine the transaction object of choice. The channel is local to the transactor, and is called *out_chan*. In the constructor, that channel is associated with the environment channel, which is passed as an argument during the build step. The command transactor (i.e., BFM) triggers a notification at the completion of a transaction to inform the generator of its status in the event the generator needs to take an action other than normal.[1] For example, a CRC error detected by the command transactor may cause the generator to send a retry of the last issued transaction. The generator triggers a notification of completion at the end of simulation to enable the environment to continue through its stepping cycles. Figure 7.1 represents the UML for the custom generator.

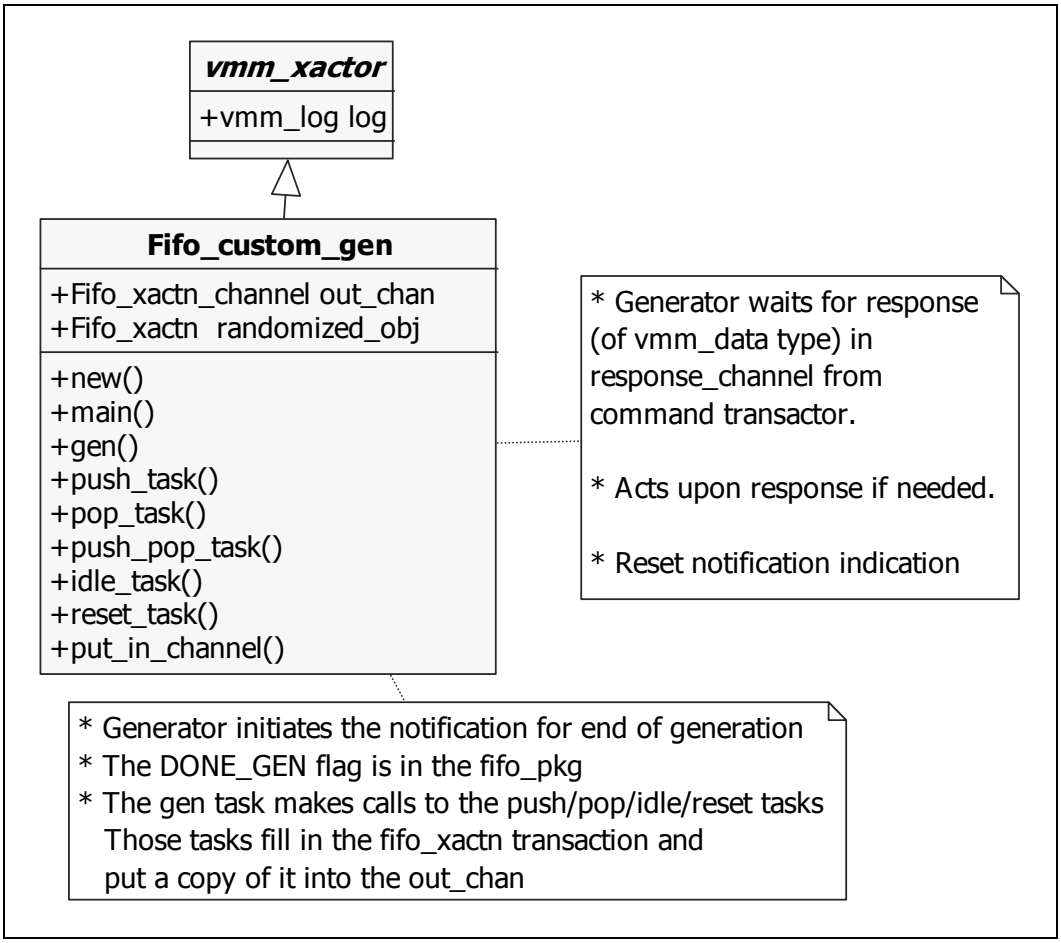


Figure 7.1 UML for Custom Generator

[1] For demonstration purposes, we limited the BFM completion notification to only the PUSH transactions when the *push()* task is called, and not all other transactions.

The goals of our model are to demonstrate the features of a custom generator; the completion notification of transaction generation; the response transaction from the command generator via a response channel or via a notification; and the use of a factory to identify a different transaction class.

7.1.1 Notification of Completion

Verilog has the event data type for process synchronization. Differences and issues related to events as compared to the *vmm_notify* methods are:

- Events are named in Verilog. A notification indication can use a notification identification of integer value. This is typically defined in an enumeration type, such as
  `typedef enum {DONE_GEN, DONE_BFM} notification_e;`

  A trigger of a notification is performed with the *vmm_notify::indicate* method and the notification ID passed as an argument. For example,
  `this.notify.indicate(fifo_pkg::DONE_GEN);`

- With *event* one cannot pass any associated status/data/information to another transactor. Such ability is useful to transfer status and data back to a transactor, such as a generator or scoreboard. That feedback information is an object derived from *vmm_data.* For example,
  `fifo_xactn_0.notify.indicate(vmm_data::ENDED,`
  `fifo_response);`

- Events are active only during the time slot in which they are triggered. This provides limited capabilities, as compared to the VMM notification that provides several synchronization modes to prevent a notification from being misused by the triggering or waiting threads. The notification synchronization modes include:[2]

  - **ON_OFF:** This is level-sensitive and requires resetting after activation. Threads waiting for a notification that is already notified will not wait. This is similar to a green light notification, and all cars waiting for the green light or arriving at the intersection when the light is green do not wait. The light remains green until reset to amber and then red.

  - **ONE_SHOT:** Only threads currently waiting for the notification to be indicated are notified. This is similar to a crossing guard who lets children currently waiting at the curb cross the street. Any child not at the curb "missed the boat" and will have to wait for the next notification.

  - **BLAST**: All threads waiting for the notification to be indicated in the same timestep as the indication are notified. This is similar to a crossing guard who allows the crossing of the street to only those children who just arrived at the curb and in same timestep he decides to allow such crossing (i.e., the indication). Thus, he "blasts" the new arrivals through the intersection.

Those issues with events create code that is less reusable and harder to maintain. To create reusable code with notification that can be configured in the environment, VMM provides the *vmm_notify* class that implements a generic, yet sophisticated, notification service. The notification mode is defined when the notification is configured, not when it is triggered or waited upon. In the FIFO custom generator, we used the *vmm_notify::ON_OFF* notification. So what is involved in notification? In any notification system, even for humans, you have the following:

---

[2] VMM *vmm_notify* Appendix A, *Table A-7. Notification Synchronization Mode Enumerated Values*

1. **A notification flag.** For example, in a raffle, the flag is a winning number. In the model, we use the enumeration constants defined in `fifo_pkg`.
   `typedef enum {DONE_GEN, DONE_BFM} notification_e;`
   DONE_GEN has the value 0, DONE_BFM has the value1, per SystemVerilog LRM.

2. **A notification configuration**. For example, in a raffle you are told that the winner needs (or does not need) to be present. For the FIFO model, the environment initialized the configuration for the *xactn_gen_0* object as shown below

```
this.xactn_gen_0.notify.configure(fifo_pkg::DONE_GEN,
                                  vmm_notify::ON_OFF);
```

3. **An indication that the notification flag is raised**. In a raffle, the organizer gets on the microphone, and calls out that number. In the model, the custom generator uses *vmm_notify.indicate* method to raise this announcement. This statement: is written as follows in the custom generator
   `this.notify.indicate(fifo_pkg::DONE_GEN);`

4. **A listener to this indication.** In a raffle where the winner needs to be present, the lucky person thankfully accepts his gift.[3] In the FIFO example, we have two users of indications. The environment needs a completion of transaction generation from the generator to know when the generator is all done. The command transactor informs the generator of its progress in processing a transaction in the event the generator needs to retry the same transaction. These statements are written as follows in the model

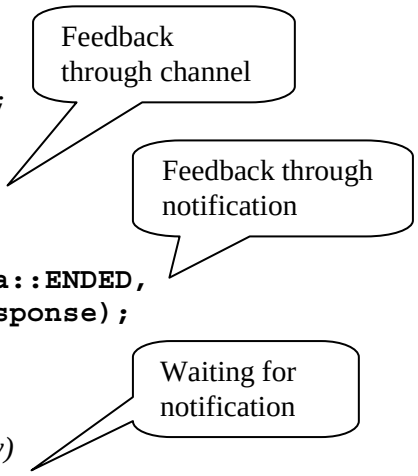
   In the environment *(file ch7_custom_generator/fifo_env.sv)*

```
task wait_for_end();
      this.xactn_gen_0.notify.wait_for(fifo_pkg::DONE_GEN);
```

   In the command transactor *(file ch7_custom_generator/fifo_cmd_xactor.sv)*. Two techniques to transfer response information back to command transactor are demonstrated: 1) through a channel, 2) through the notification itself. You do not need both.

```
begin : main_loop
  …
  fifo_response=new();
  fifo_response.kind=fifo_xactn_0.kind;
  fifo_response.status= PASSED;
  // Technique 1: Through the channel
  this.resp_chan.sneak(fifo_response);

  // Technique 2: Through notification
  fifo_xactn_0.notify.indicate(vmm_data::ENDED,
                               fifo_response);
  this.in_chan.get(fifo_xactn_0);
end : main_loop
```



   In the generator *(file ch7_custom_generator/fifo_env.sv)*

```
randomized_obj.notify.wait_for(vmm_data::ENDED);
```

[3] At DVCon'06 on his way to listen to the indication of the winning number of a raffle by Synopsys, Ben was held up by a friend, far away from the raffle. At the booth, the indication of the lucky winner of the iPod was Ben! But Ben failed to listen to this indication! Since the configuration of this notification required the winner to be present (i.e., ONE_SHOT versus ON_OFF), Ben did not win the iPod. Poor Ben!

The use of the *status* method to read the attached response is explained further down.
Figure 7.1.1 demonstrates through UML the relationships needed for the notification of end of test used in the FIFO model.

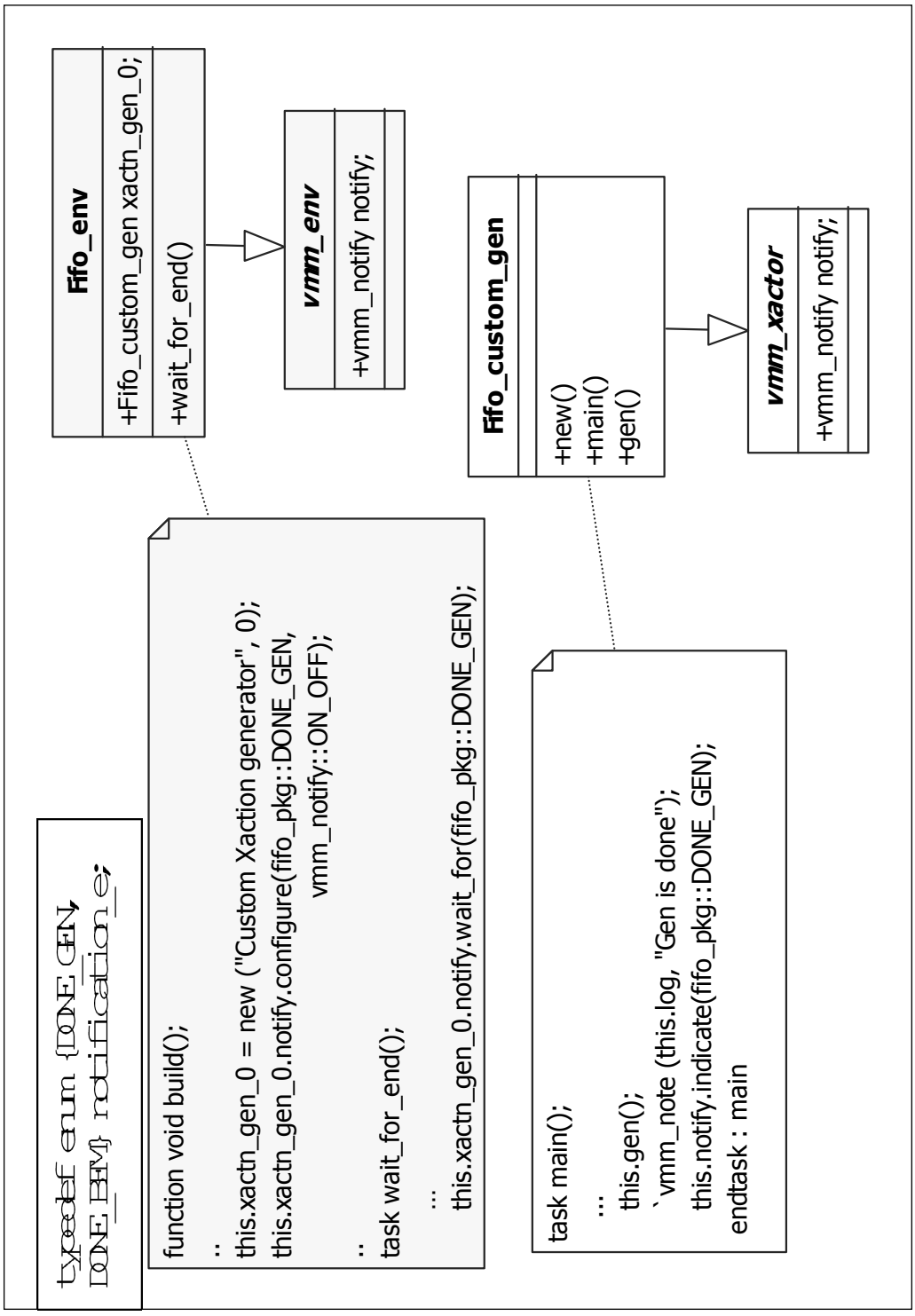


Figure 7.1.1 UML Notification of DONE_GEN for FIFO Model

7.1.2 Generator Design

Figure 7.1.2 represents the model for the custom generator.

```
class Fifo_custom_gen extends vmm_xactor;
  import fifo_pkg::*;
  Fifo_xactn_channel out_chan;
  Fifo_xactn         randomized_obj;

  Fifo_response_channel resp_chan;  // response from cmd xactor
  int unsigned       stream_id = -1;
  function new(string                inst,
               int unsigned          stream_id = -1,
               Fifo_xactn_channel    out_chan=null,
               Fifo_response_channel fifo_response_channel=null);
      super.new("Custom Fifo COMMAND Layer Xactor",
                inst, stream_id);
        if (out_chan==null)
             out_chan=new("fifo_chanel", "channel");
       else this.out_chan=out_chan;

       if (fifo_response_channel==null)
         this.resp_chan=new("fifo_response_chan", "channel");
       else this.resp_chan=fifo_response_channel;
       // setting up the configuration for end of simulation notification
       this.notify.configure(fifo_pkg::DONE_GEN,
                             vmm_notify::ON_OFF);
       this.stream_id=stream_id;
  endfunction : new

  task main();
    `vmm_trace(log, "Inside Generator");
    fork
      super.main();
   join_none
      if (randomized_obj==null) randomized_obj=new();
    this.gen();
    `vmm_note (log, "Gen is done");
    this.notify.indicate(fifo_pkg::DONE_GEN);
  endtask : main

    extern task gen(); // generation of transactions
    extern task push(word_t data);
    extern task pop();
    extern task push_pop(word_t data);
    extern task idle(int idle);
    extern task reset();
    extern task put_in_channel(Fifo_xactn obj);
endclass : Fifo_custom_gen
```

Can be updated with handle of new instance in *program* block.

Response channel transfers information back to generator

Notification after generator is done with the *gen* task

```
task Fifo_custom_gen::gen();
    word_t v_data=0;   // not used, shown for potential use.
    string msg;
       `vmm_trace(log,
     "Generator creating directed tests: Reset, 4 push, 4 pop");
    this.reset();
    // Do 4 consecutive push with directed data
    for (int i=0; i<4; i++) this.push(i);
    // do 4 pop
    repeat(4) this.pop();
    // do rand streams
    `vmm_trace(log,
           "Generator creating stream randsequence tests");
     for (int i=0; i<100; i++)
       randsequence (stream)
        stream : first second third; // := 10 | second := 20 | third :=1;
          first : s_push | s_pop;
          second : s_idle :=2 | s_push_pop := 7 | s_reset:= 1;
          third : s_idle:=8 | s_pop :=2;
          s_push          : {this.push(i*4);};
          s_pop           : {this.pop();};
          s_idle          : {this.idle(1);};
          s_push_pop      : {this.push_pop(i*2);};
          s_reset         : {this.reset();};
        endsequence
      idle(1);
      //do random tests
       `vmm_trace(log,
            "Generator creating random tests per transaction class constraints");
     repeat(100) begin : for_random_tests
       randomized_obj.randomize();
       this.put_in_channel(randomized_obj);
     end : for_random_tests
    `vmm_note(log, "done tests");
endtask : gen
```

```
// Push task with code for waiting for notification from command transactor
task Fifo_custom_gen::push(word_t data);
     Fifo_response  bfm_response;
      `vmm_note(log, "PUSH");
(1)   randomized_obj.data=data;
      randomized_obj.kind=PUSH;

(2)   this.put_in_channel(randomized_obj);


(3)   randomized_obj.notify.wait_for(vmm_data::ENDED);
      // get the BFM response from the notification
(4)  $cast(bfm_response,
           randomized_obj.notify.status(vmm_data::ENDED));

      // Check response from notification
(5)   if (bfm_response.fifo_status==fifo_pkg::PASSED)
        `vmm_trace(log, "Cmd Xactor Ended the PUSH");
      else `vmm_note(log,
             "Cmd Xactor FAILED in completing the PUSH");


    // must reset the notification
    // A vmm_notify::SOFT reset clears the specified
    // ON_OFF notification and
    // restarts the vmm_notification::indicate() and
    // vmm_notification::reset()
(6) // methods on any attached notification descriptor.
    randomized_obj.notify.reset(vmm_data::ENDED,
                                vmm_notify::SOFT);
endtask : push

task Fifo_custom_gen::pop();
  // notification not used
      `vmm_note(log, "POP");
      randomized_obj.data=32'hX;
      randomized_obj.kind=POP;
      this.put_in_channel(randomized_obj);
endtask : pop

task Fifo_custom_gen::push_pop(word_t data);
      `vmm_note(log, "PUSH_POP");
      randomized_obj.data=data;
      randomized_obj.kind=PUSH_POP;
      this.put_in_channel(randomized_obj);
endtask : push_pop
```

1. Setup the transaction
2. Put in channel
3. Wait from BFM to finish
4. Get response
5. Check for PASSED
6. Reset notification

Feedback through notification

Feedback through channel could be used here.

```
task Fifo_custom_gen::idle(int idle);
    $display("idle");
    randomized_obj.kind=IDLE;
    randomized_obj.idle_cycles=idle;
    this.put_in_channel(randomized_obj);
endtask : idle

task Fifo_custom_gen::reset();
      `vmm_note(log, "RESET");
      randomized_obj.data=32'hX;
      randomized_obj.kind=RESET;
      this.put_in_channel(randomized_obj);
endtask : reset

task Fifo_custom_gen::put_in_channel(Fifo_xactn obj);
      Fifo_xactn fifo_inst;
      obj.data_id++; // ID maintenance is custom
      $cast(fifo_inst, obj.copy(obj));
      this.out_chan.put(fifo_inst);
endtask : put_in_channel
```

Figure 7.1.2 represents the model for the custom generator.
/ch7_custom_generator/fifo_custom_gen.sv

7.1.3 Simulation of Custom Generator

The full code for the FIFO is available in file */ch7_custom_generator*. Figure 7.1.3 represents snippets of the simulation results.

```
0.00 ns test [Normal:NOTE] | Start of Test
0.00 ns Verif Env [Trace:INTERNAL] | Generating test configuration...
0.00 ns Verif Env [Trace:INTERNAL] | Building verification environment...
0.00 ns Verif Env [Trace:INTERNAL] | Reseting DUT...
1950.00 ns Verif Env [Trace:INTERNAL] | Configuring...
1950.00 ns Verif Env [Trace:INTERNAL] | Saving RNG state information...
1950.00 ns Verif Env [Trace:INTERNAL] | Starting verification environment...
1950.00 ns Verif Env [Trace:INTERNAL] | Restoring RNG state information...
1950.00 ns Verif Env [Trace:INTERNAL] | Waiting for end of test...
1950.00 ns FIFO_GEN_CUSTOM [Trace:INTERNAL] | Started
1950.00 ns FIFO_GEN_CUSTOM [Trace:DEBUG] | Inside Generator
1950.00 ns FIFO_GEN_CUSTOM [Trace:DEBUG] | Generator creating directed tests: Reset, 4 push, 4 pop
1950.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | RESET
1950.00 ns cmd_xactor [Trace:INTERNAL] | Started
1950.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.1 Fifo Xaction RESET  Cycles 0
1950.00 ns Fifo Monitor Xactor [Trace:INTERNAL] | Started
2250.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | PUSH
2250.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.2 Fifo Xaction PUSH
2350.00 ns Fifo Monitor Xactor [Trace:DEBUG] | Found a PUSH Xactn at time 2350.00 ns data 0
2350.00 ns FIFO_GEN_CUSTOM [Trace:DEBUG] | Cmd Xactor Ended the PUSH
2350.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | PUSH
```

```
2350.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.3 Fifo Xaction PUSH
2450.00 ns Fifo Monitor Xactor [Trace:DEBUG] | Found a PUSH Xactn at time 2450.00 ns data 1
2450.00 ns FIFO_GEN_CUSTOM [Trace:DEBUG] | Cmd Xactor Ended the PUSH
2450.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | PUSH
…
2950.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | POP
2950.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.9 Fifo Xaction POP
3050.00 ns FIFO_GEN_CUSTOM [Trace:DEBUG] | Generator creating stream randsequence tests
3050.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | PUSH
..
3150.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | PUSH_POP
..
3250.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.12 Fifo Xaction IDLE
Cycles 1
..
70650.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.309 Fifo Xaction
POP
idle
70750.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.310 Fifo Xaction
IDLE  Cycles 1
70850.00 ns FIFO_GEN_CUSTOM [Trace:DEBUG] | Generator creating random tests per transaction
class constraints
70850.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.311 Fifo Xaction
PUSH
70950.00 ns Fifo Monitor Xactor [Trace:DEBUG] | Found a PUSH Xactn at time 70950.00 ns data 12d
70950.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.312 Fifo Xaction
..
81550.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.409 Fifo Xaction
PUSH_POP
81650.00 ns Fifo Monitor Xactor [Trace:DEBUG] | Found a PUSH Xactn at time 81650.00 ns data 229
81650.00 ns cmd_xactor [Trace:DEBUG] | Got a new fifo xaction from in_channel  #0.0.410 Fifo Xaction
POP
81750.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | done tests
81750.00 ns FIFO_GEN_CUSTOM [Normal:NOTE] | Gen is done
81750.00 ns FIFO Env Logger [Normal:NOTE] | *** Completed transactions
81750.00 ns Verif Env [Trace:INTERNAL] | Stopping verification environment...
81750.00 ns Verif Env [Trace:INTERNAL] | Cleaning up...
Simulation PASSED on /./ (/./) at          81750.00 ns (1 warnings, 0 demoted errors & 0 demoted warnings)
81750.00 ns FIFO Env Logger [Trace:DEBUG] | This is where additional model info is displayed
81750.00 ns FIFO Env Logger [Normal:NOTE] | **** REPORT ***
81750.00 ns test [Normal:NOTE] | End of Test
$finish at simulation time          81750.00 ns
..
```

Figure 7.1.3 Simulation of Custom Generator

7.2 FILE STRUCTURE

Table 7.2 demonstrates the file Structure and the purpose of each file.

Table 7.2. File Structure and Functions

/ch6/ch6_callback and /ch5*/ch5_fct_inject_err* directories

File	Function	Used by
fifo_pkg.sv	Defines types and initialized variables	ALL
fifo_if.sv	Defines the FIFO interface	RTL and by program, testbench, transaction and transactors
fifo_csr_if.sv	Defines the FIFO configuration interface	RTL, property models, and by environment, and possibly transactors
fifo_xactn.sv	Defines the transaction class with the constraints Also used for the channel generation with: *`vmm_channel* (Fifo_xactn)	*`vmm_channel* macro for generation of channel
fifo_rtl.sv	Represents the FIFO RTL DUT	Top level
fifo_props.sv	Defines the properties for assertions	Top level for bind
fifo_log_fmt.sv	Defines formatting information for display	FIFO environment
fifo_pgm.sv	Creates the modeling for simulation and initiates the *run* in the environment	Top level
fifo_env.sv	Creates the build and start for simulation	program
fifo_mon_xactor.sv	Creates a copy of the observed transaction onto a transaction channel.	Scoreboard, top level
top_tb.sv	Represents the top level and instantiates the RTL, the bind, the monitor, etc	none
test.svh	Common include files	Program block
fifo_response.sv	Class derived from vmm_data to provide a response to a transactor (e.g., generator) through a channel	command transactor and environment
fifo_custom_gen.sv	Custom generator	Environment

Chapter 7 Questions and LAB

Q1. When is a custom generator needed?

Q2. When should notification via the *vmm_notify* be used?

Lab07

Create a custom generator for the counter. See instructions in subdirectory lab/lab07/todo/readme.txt.