CODE CROSS REFERENCE

Amrutha Anil (amrutha@pdx.edu)
Manjari Rajasekharan (manjari@pdx.edu)
Durganila Anandhan (anandhan@pdx.edu)

Directory Structure

File	Description	Relative Path
axi_lite_master.sv	Drives address and controls to the slave via interface.	.DUT/axi_lite_master.sv
axi_lite_slave.sv	Writes data received from master to a memory. Sends data from the requested address to the master.	.DUT/axi_lite_slave.sv
axi_lite_interconnect.sv	Connects multiple slaves and masters.	.DUT/axi_lite.interconnect.sv

Table 1: DUV Files

File	Description	Relative Path
transaction.sv	Contains all the random variables and other variables required to generate a transaction. Some of the variables are constrained.	.TB/transaction.sv
top.sv	Top module where the DUVs are instantiated. Environment class is executed in this class.	.TB/top.sv
testFactory.sv	Allows user to select the testbench during run time.	.TB/testFactory.sv
scoreboard.sv	Compares the BFM's address lines and the testbench local memory.	.TB/scoreboard.sv
monitor.sv	Monitors the virtual interface and send the data to scoreboards class via mailbox.	.TB/monitor.sv
generator.sv	Parent class for test classes. Contains pure virtual execute task to be defined in all the child classes.	.TB/generator.sv
driver.sv	The data send by test classes are received by the driver class using mailbox and	.TB/driver.sv

	drives stimulus to DUV via virtual interface.	
axi_lite_coverage.sv	Functional coverage groups and bins.	.TB/axi_lite_coverage.sv
axi_env.sv	Objects of generator, driver, monitor, scoreboard, testfactory and coverage are created. Execute task is called to run the handles concurrently.	.TB/axi_env.sv
fully_random_test.sv	Contains fully random test stimulus. Uses mailbox to send the transactions to driver.	.TB/TestTypes/fully_random_test.sv
deterministic_tests.sv	Contains different interesting sequence for DUV inputs, edge cases and error conditions	.TB/TestTypes/deterministic_test.sv

Table 2: Testbench Files

File	Description	Relative Path
axi_lite_pkg.sv	Contains shared parameters, structures and enums used in DUV and testbench	./axi_lite_pkg
axi_lite_if.sv	Encapsulates all the signals required by the DUV.	./axi_lite_if.sv

Table 3: Shared files

testFactory.sv

Purpose:
Allows user to select the testbench during run time.

Class Instantiations:
None

Other functions called by the class:
None

Variables:
None

Tasks & Functions:

Task	Purpose

`static function generator Get_TestType (string testType, mailbox mb_generator2driver, logic debugMode, int numTransactions);`
 	
`fully_random_test fully_random_test_h;`
 	
`case (testType)`
`"full_random" : begin`
`fully_random_test_h = new(mb_generator2driver, debugMode, numTransactions);`
`return fully_random_test_h;`
`end`
	To get the handler of test selected by the user

transaction.sv

Purpose:
Used to generate transactions of various kinds. Contains random variables and fixed variables required to generate a transaction.

Class Instantiations:
None

Other functions called by the class:
None

Variables and constraints:

	Purpose

`logic reset_n;`
 	
`rand addr_t addr;`
`rand data_t data;`
`bit start_read;`
`bit start_write;`
`data_t rd_data;`
	To generate random resets to drive into DUV
	
	Generate read and write address
	Generate write data
	To start a read
	To start a write
	To get read data

`constraint addr_range { addr inside{[0 : 4098]}; }`
 	
`constraint dist_data_cn { data dist { 0 := 40, [1: data_range] := 60}; }`
	To constraint the address within a range
	
	To constraint data within a range

Tasks & Functions:
None

fully_random_test.sv

Purpose:
Contains fully random test stimulus. It uses mailbox to send the transactions to driver.

Class Instantiations:
None

Other functions called by the class:
None

Variables and handlers:

Variable	Purpose
`transaction txn;` `transaction generate_pkt;`	transaction object to send the transaction to mailbox transaction object to generate random transactions

Tasks & Functions:

```
function new(mailbox mb_generator2driver, logic debugMode, int numTransactions);
    super.new(mb_generator2driver, debugMode, numTransactions);
```

Constructor that calls the super class new function.

```
task execute();
      repeat(numTransactions) begin
```

Task to generate the random transactions

```
task driver_send(
                input
                logic reset_n,
                addr_t addr,
                data_t data,
                logic start_read,
                logic start_write
);
    txn                 = new();
    txn.reset_n         = reset_n;
    txn.addr            = addr;
    txn.data            = data;
    txn.start_read      = start_read;
    txn.start_write     = start_write;

    if(debugMode)
        $display($time," fully_random_test.driver_send: reset_n %b, addr %h, data %h read %b, write %b",
                    reset_n, addr, data, start_read, start_write);
```

Task to send the generated transaction to mailbox which will be used by driver to drive the stimulus to interface.

generator.sv

Purpose:
Parent class for test classes. Contains pure virtual execute task to be defined in all the child classes.

Class Instantiations:
None

Other functions called by the class:
None

Variables and handlers:

Variable	Purpose

`mailbox mb_generator2driver`
`logic debugMode`
`int numTransactions`
	Mailbox for driving transactions from generator to driver
	To get the debug mode
	To get the number of transactions to run

Tasks & Functions:

```
pure virtual task execute();
```

Ensures all the child classes define execute definition

```
function new(mailbox mb_generator2driver, logic debugMode, int numTransactions);
```

Constructor for generator class

driver.sv

Purpose:
The data send by test classes are received by the driver class using mailbox and drives stimulus to DUV via virtual interface.

Class Instantiations:
None

Other functions called by the class:
None

Variables and handlers:

Variable	Purpose

`mailbox mb_generator2driver`
`logic debugMode`
`int numTransactions`
	Mailbox for driving transactions from generator to driver
	To get the debug mode
	To get the number of transactions to run

Tasks & Functions:

```
task execute();
    txn = new();

    forever begin
        mb_generator2driver.get(txn);
        drive_master(txn);
    end
endtask
```

Execute task to get the transaction from transaction mailbox generated by test classes.

```
task drive_master(transaction txn);
    // set writes
    bfm0.addr           = txn.addr;
    bfm0.data           = txn.data;
    bfm0.start_write    = txn.start_write;
    bfm0.start_read     = txn.start_read;
```

Task to drive the transaction to interface. It follows the DUV pin level protocols.

```
function new(mailbox mb_generator2driver, virtual axi_lite_if bfm0, logic debugMode);
```

Constructor to get the interface, user inputs and mailbox from generator.

scoreboard.sv

Purpose:
Compares the BFM's address lines and the testbench local memory and outputs the score.

Class Instantiations:

```
transaction         txn;
```

Other functions called by the class:

```
this.mb_monitor2scoreboard  = mb_monitor2scoreboard;
this.bfm                    = bfm;
txn                         = new();

mb_monitor2scoreboard.get(txn);
```

Variables and handlers:

Variable	Purpose
`mailbox mb_monitor2scoreboard;`	Mailbox for driving transactions from generator to driver.
`virtual axi_lite_if bfm;`	Virtual interface to get data from interface.
`transaction txn;`	Transaction object to send data to scoreboard
`int score = 0;`	To store score.
`logic [DATA_WIDTH-1:0] local_mem[BUFFER_SIZE];`	Store values for comparison.
`int i;`	Temporary variable for a for loop.

Tasks & Functions:

```
function new (mailbox mb_monitor2scoreboard,  virtual axi_lite_if bfm);
```

Function to create instances of mailbox, interface and transaction and to initialise the local memory locations to 0.

```
protected task save_val();
```

Function to read the write data line and store the value to local memory.

```
protected task check_val();
```

Function to read the read data line and check the local memory to confirm.

```
task execute();
```

Function to check read line and write line and compute the score for the scoreboard.

monitor.sv

Purpose:
Monitors the virtual interface and send the data to scoreboards class via mailbox.

Class Instantiations:

```
//Instantiate the BFM:
axi_lite_if bfm0(.aclk(aclk), .areset_n(areset_n));

//Instantiate the DUT master and slave:

axi_lite_master #(addr0) master0 (
        .m_axi_lite(bfm0.master)
);

axi_lite_slave slave0 (
  .s_axi_lite(bfm0.slave)
);
```

Other functions called by the class:

```
env_h = new(bfm0, test_type, debugMode, numTransactions);
env_h.execute();
```

Variables and handlers:

Variable	Purpose
`mailbox mb_monitor2scoreboard;`	Mailbox to send data to scoreboard
`virtual axi_lite_if bfm0;`	Virtual interface to get data from interface
`transaction txn;`	Transaction object to send data to scoreboard
`logic debugMode;`	Object for debug mode

Tasks & Functions:

```
task execute();
    forever begin
        sampleData();
    end
endtask
```

Task which runs forever and calls sample data task.

```
task sampleData();
    @(posedge bfm0.aclk);
```

Task to sample data from virtual interface and send to the scoreboard via mailbox at every positive edge of the clock

axi_lite_coverage.sv

Purpose:
To create the coverage groups and bins to obtain functional coverage.

Class Instantiations:
None

Other functions called by the class:
None

Variables and handlers:

Variable	Purpose
`virtual axi_lite_if bfm;`	Virtual interface to get data from interface

Tasks & Functions:

```
function new (virtual axi_lite_if b);
    cg_Read_Address = new();
    cg_Read_Data = new();
    cg_Write_Address = new();
    cg_Write_Data = new();
    cg_Write_Response = new();
    cg_Reset_Signal = new();
    cg_Master_FSM = new();
    cg_Slave_FSM = new();

    this.bfm = b;

endfunction : new
```

Function to create instances of the covergroups

```
task execute();
    forever begin : sampling_block
        @(posedge bfm.aclk);
        cg_Read_Address.sample();
        cg_Read_Data.sample();
        cg_Write_Address.sample();
        cg_Write_Data.sample();
        cg_Write_Response.sample();
        cg_Reset_Signal.sample();
        cg_Master_FSM.sample();
        cg_Slave_FSM.sample();
    end : sampling_block

endtask : execute
```

Function to sample the covergroups at the posedge of clock.

axi_env.sv

Purpose:

To create the environment with the objects of generator, driver, monitor, scoreboard, testfactory and coverage. The *execute* task is called to run the handles concurrently.

Class Instantiations:

```
generator           generator_h;
driver              driver_h;
monitor             monitor_h;
scoreboard          scoreboard_h;
axi_lite_coverage   coverage_h;
testFactory         testFactory_h;
```

Other functions called by the class:

```
generator_h     = testFactory_h.Get_TestType(testType, mb_generator2driver, debugMode, numTransactions);
driver_h        = new(mb_generator2driver, bfm0, debugMode);
monitor_h       = new(mb_monitor2scoreboard, bfm0, debugMode);
scoreboard_h    = new(mb_monitor2scoreboard, bfm0);
coverage_h      = new(bfm0);

monitor_h.execute();
scoreboard_h.execute();
generator_h.execute();
driver_h.execute();
coverage_h.execute();
```

Variables and handlers:

Variable	Purpose
`virtual axi_lite_if bfm0;`	Virtual interface to get data from interface
`string testType;`	Variable for testType
`logic debugMode;`	Variable for debug mode
`int numTransactions;`	Variable for number of transactions

Tasks & Functions:

```
function new (virtual axi_lite_if bfm0, string testType, logic debugMode, int numTransactions );
```

Function to create instance of interface and the variables.

```
task execute();
```

Function to create new objects of generator, driver, monitor, scoreboard, testfactory and coverage and run the *execute* function of each of these objects.

deterministic_tests.sv

Purpose:
Contains different interesting sequences for DUV inputs, edge cases and error conditions.

Class Instantiations:
None

Other functions called by the class:
None

Variables and handlers:

Variable	Purpose
transaction txn	transaction object to send the transaction to mailbox

Tasks & Functions:

```
function new(mailbox mb_generator2driver, logic debugMode, int numTransactions);
    super.new(mb_generator2driver, debugMode, numTransactions);
```

Constructor that calls the super class new function.

```
task execute();
    int i,j;

     //write and read 1 byte data
    driver_send(1'b1, 12'h4, 8'hEF, '0, '1);
    #10;
    driver_send(1'b1, 12'h4, '0, '1, '0);
    #10;

    //write and read 2 bytes of data
    driver_send(1'b1, 12'h4, 16'hBEEF, '0, '1);
    #10;
    driver_send(1'b1, 12'h4, '0, '1, '0);
    #10;

    //write and read 3 bytes data
```

Task to generate the directed tests

```
task driver_send(
                input
                logic reset_n,
                addr_t addr,
                data_t data,
                logic start_read,
                logic start_write
);
    txn                 = new();
    txn.reset_n         = reset_n;
    txn.addr            = addr;
    txn.data            = data;
    txn.start_read      = start_read;
    txn.start_write     = start_write;

    if(debugMode)
        $display($time," fully_random_test.driver_send: reset_n %b, addr %h, data %h read %b, write %b",
                    reset_n, addr, data, start_read, start_write);
```

Task to send the generated transaction to the mailbox which will be used by the driver to drive the stimulus to the interface.

top.sv

Purpose:
Top module where the DUVs are instantiated. Environment class is executed in this class.

Class Instantiations:

```
//Instantiate the BFM:
axi_lite_if bfm0(.aclk(aclk), .areset_n(areset_n));

//Instantiate the DUT master and slave:

axi_lite_master #(addr0) master0 (
        .m_axi_lite(bfm0.master)
);

axi_lite_slave slave0 (
  .s_axi_lite(bfm0.slave)
);
```

Other functions called by the class:

```
env_h = new(bfm0, test_type, debugMode, numTransactions);
env_h.execute();
```

Variables and handlers:

Variable	Purpose
`logic aclk;` `logic areset_n;`	Clock Active low reset

`string test_type`
`logic debugMode`
`int numTransactions`
	Get which test type to run from user during runtime
	Get user input if the run is in debug mode
	Get the number of transactions/runs to be performed

`environment env_h;`	Environment object

Tasks & Functions:

```
task InitialReset();
  areset_n = 0;
  repeat(10) @(posedge aclk);
  areset_n = 1;
```

To perform the initial reset