CROSS REFERENCE OF CODE

i)tbBFM.sv : This is a testbench interface having inputs which are driven into design and protocol signals to capture the transactions between master and slave. Clock is generated in this interface.

ii) duv.sv: This module has a testbench interface[tbBFM bfm] as input. This module contains a design interface instance[inf] and AXI_top_design instance[top_inst]. This duv captures the transactions between master and slave interface to testbench interface .

iii)axiprotocol_pkg.sv : This is a package which contains different classes like request, tester, coverage, scoreboard .

1. **Class request:** This class constraints the inputs as per the AMBA 3 AXI specification to provide for distributed constrained randomization. The signals constrained are AWLEN/ARLEN 0000 ->1 through 1111->16;ARSIZE/AWSIZE 000->1 through 111 -> 128;ARBURST/AWBURST 00 -> fixed 01 -> INCR 10-> WRAP;AW_Addr>32'h5ff and <=32'hfff and awsize <3'b100;AR_Addr>32'h1ff AR_Addr<=32'hfff;W_strobe{ 0001, 0010, 0100,1000,0011,0101,1001, 0110, 1010,1100,00111,1110,1011, 1101, 1111}.
2. **Class tester:** This class contains a constructor to initialize virtual interface. This has different tasks like execute, read, write , determine, rest_bus.

    a)**task read** - it sends the values needed to initiate a read to the duv.It breaks when the master has initiated the read we asked it to initiate.

    b)**task write**- it sends the values needed to initiate a write to the duv. It breaks when the master has initiated the write we asked it to initiate.

    c**)task determine**- it drives deterministic inputs covering different cases like fixed length,incrementing,wrapping read bursts of different lengths and sizes, write bursts,different scenarios.

    d)**task execute-** It is enabled to take $plusargs inputs from the command line to select between different tasks. It has static constrained randomization to choose between different operations like single read burst, write burst,overlapping read bursts, transaction ordering. It has repeat 1000 loop to drive constrained random inputs into duv.

    e)**task reset_bus**- it resets the bus by driving reset signal low and later making high.

3. **Class coverage:** It has a virtual interface instance ,different cover groups are written to capture inputs , output coverage. It has a task execute to sample the covergroups.
    - **Covergroup inputs**- has coverpoints which captures both read ,write bursts signals and cross coverages between them.
    - **Covergroup outputs**- has coverpoints which capture interface signals between master and slave.

    Covergroup masterstates- Written in AXI_master.sv design . It has coverpoints WAState_M,WState_M ,BState_M ,ARState_M,RState_M which captures the internal state transitions (IDLE,START,WAIT,VALID)of different channels -write address master, write data master,write response, read address master. And their sampling at posedge of clock.

    Covergroup slavestates - Written in AXI_slave.sv design . It has coverpoints WAState_S,WState_S,BState_S,ARState_S,RState_S which captures the internal state transitions (IDLE,START,WAIT,VALID)of different channels -write address slave, write data slave,write response, read address slave. And their sampling at posedge of clock.

4. **Class scoreboard:** Has a constructor task new with a virtual interface and an execute task. The execute task has two queues: one to manage read operations and another to manage write operations. The body of the execute task is composed of fork-join_none forever @ blocks. These blocks emulate always @ blocks. One of these is to check that read commands are valid, and then enter them in the read queue. There is a block that serves the same function for writes. Another one of these blocks checks to make sure the master can initiate overlapping read commands. Two of the blocks are to check if RLAST or WLAST are asserted in an invalid context. The final two blocks check that out of order operations are functioning correctly and that any operation is sending the correct number of data bursts and following the handshaking protocol.

5. **Class testbench:** It has a constructor with virtual interface as arguments initializing virtual interface. It has a task execute which creates handles for tester, coverage, scoreboard and initialises them and by using fork -join calls execute tasks of tester, coverage, scoreboard classes.

iv)top.sv : The top module imports axiprotocol package. Instantiate testbench interface, instantiates duv by passing testbench interface as arguments, it creates handle to class

testbench and initializes it and calls execute task with new handle.It has simulation time specified to make the testbench finishes the simulation.

v)**Makefile :** contains list of files to compile,simulate,run. Has commands to collect code,functional,state,branch coverage.