CSE 591: Advanced Hardware Design & Verification

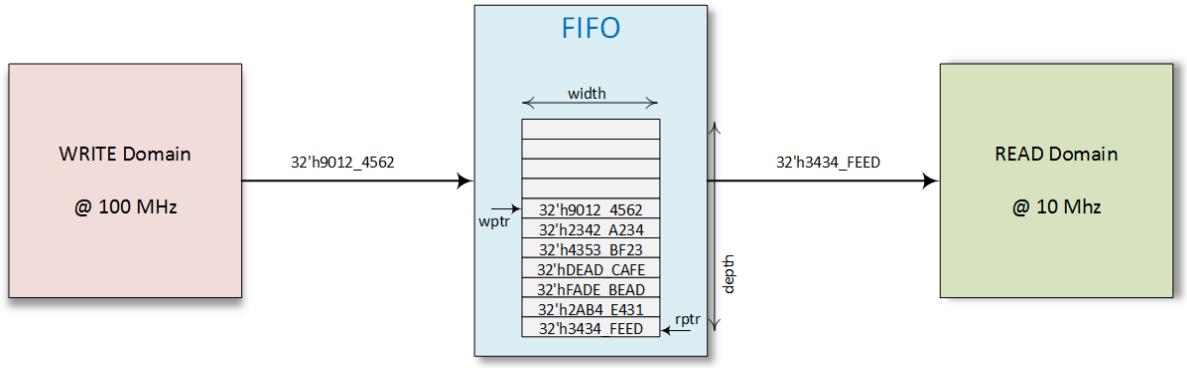
LAB #02 : *Development of Verification TestBench for Asynchronous FIFO*

Due on April 7th, before 11:59 pm

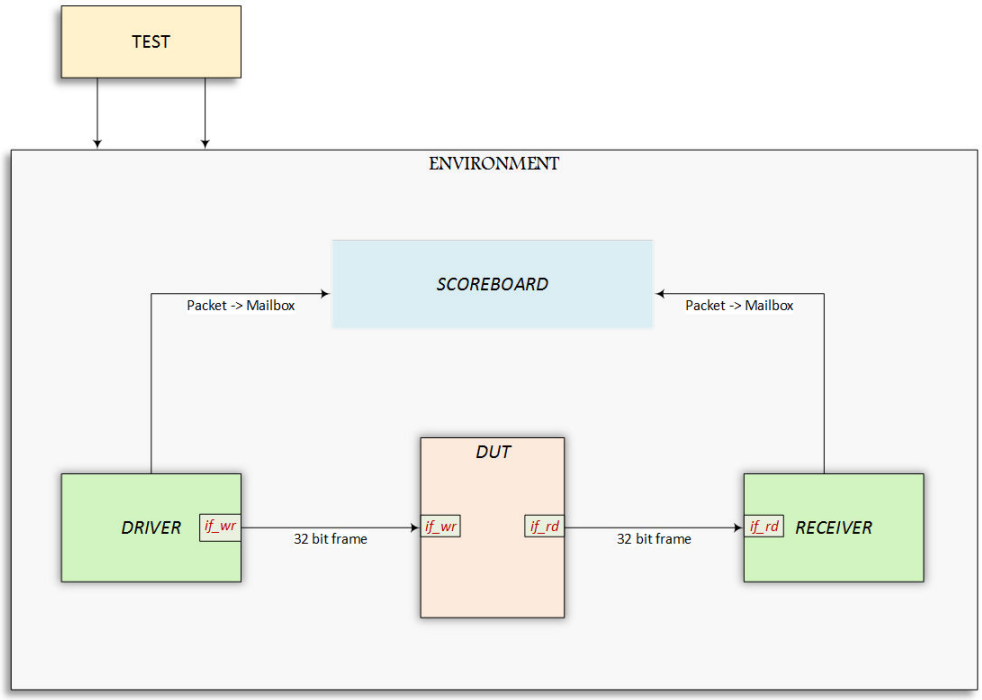
Background:

Shorter time-to-market requirements and verification of complex designs have led to the need for re-usable, self-checking and abstract models of the verification environment. In this lab, we'll develop an elaborative TestBench to verify the Asynchronous FIFO (DUT) we developed in the previous lab.

Description:

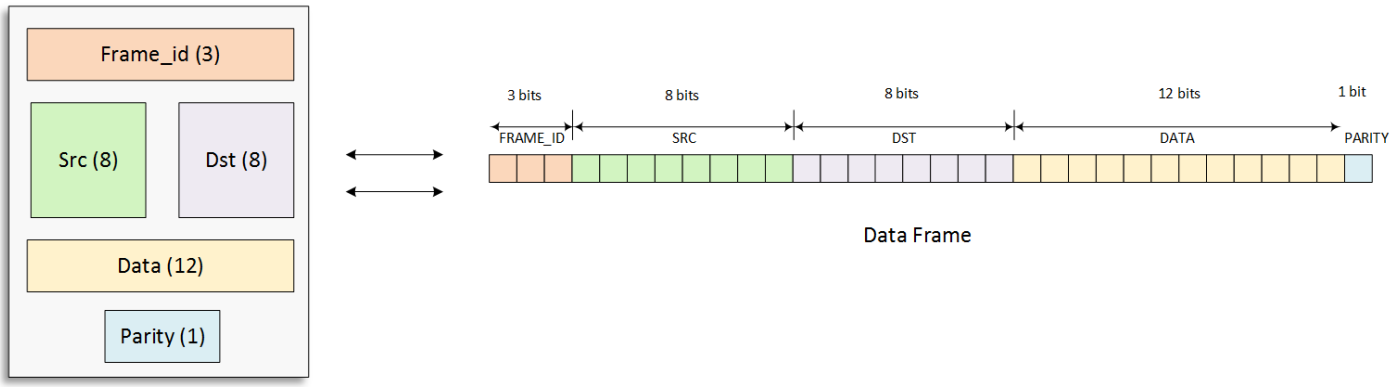


The FIFO has a data width of 4 bytes (32 bits), rest remains the same. The transmitted data is in the form of a frame made up of 32 bits. In the verification environment, data exists as a class object called Packet. The packet has different fields that contain data. This packet is generated and driven by the Driver to the DUT through the write interface. The Receiver acts as the Read Domain and reads data from the DUT through the read interface. Both the Driver and the Receiver sends the data packet to the Scoreboard which then checks if the data sent matches the data received. All tests are written in the program block and uses tasks from Driver/Receiver to write and read.



Question 1: Write a class called **Packet** which contains the following fields. The number in the bracket indicates the size in bits. The class should have the following methods:

1. **new ()** - set the fields to values passed from argument, default = 0.
2. **display ()** - display the contents of the packet in a formatted way.
3. **pack ()** - pack the contents into a frame of 32bits.
4. **unpack ()** - unpack the contents of a frame into a given Packet.
5. **compare ()** - compare the contents of one packet with another.



Packet

```
class Packet;
        function new ( … );
        function void pack (…);
        function void unpack (…);
        function void display (…);
        function compare (…);
endclass : Packet
```

Question 2: Create two separate interface modules with appropriate clocks as inputs.

1. **if_wr** - containing signals of the WRITE Domain
2. **if_rd** - containing signals of the READ Domain

Both the interfaces should use ***clocking blocks*** and ***modports*** appropriately.

```
interface if_wr # (parameter DATA = 8) (<clock>);
        // signals
        // clocking block
        // modports
endinterface : if_wr
```

```
interface if_rd # (parameter DATA = 8) (<clock>) ;
        // signals
        // clocking block
        // modports
endinterface : if_rd
```

Question 3: Create a class **Driver** with the following:

1. Instantiate a virtual interface **if_vi**.
2. Instantiate a mailbox for the connection with the scoreboard **drvr2sb**.
3. Task **write** that'll run for **n** cycles of the clock and do the following :
    a. Create conditions for writing into FIFO.
    b. Create a Packet, randomize it, and pack it into 32 bit frame.
    c. Drive the DUT with the frame
    d. Send the packet to the Scoreboard.

```
class Driver;
        // mailbox instantiation
        // virtual interface
        function new (…);
        task write (…);
endclass : Driver
```

Question 4: Create a class **Receiver** with the following:

1. Instantiate a virtual interface **if_vi**.
2. Instantiate a mailbox for the connection with the scoreboard **rcvr2sb**.
3. Task **read** that'll run for **n** cycles of the clock and do the following :
    a. Create conditions for reading from the FIFO.
    b. Receive a 32 bit data frame, unpack it into a Packet.
    c. Send the Packet to the scoreboard.

```
class Receiver;
        // mailbox instantiation
        // virtual interface
        function new (…);
        task read (…);
endclass : Receiver
```

Question 5: Create a class **Scoreboard** and instantiate two mailboxes for connecting to the Driver and the Receiver. Also write tasks to check for valid FULL and EMPTY conditions and to check if data received = data sent. Fork these tasks inside **start ().**

```
class Scoreboard;

        // mailbox instantiations

        function new (…);

        task start (…);

        // another other tasks that you think might be required.

endclass : Scoreboard
```

Question 6: Create a class **Environment** to instantiate all the TestBench components (Driver, Receiver, Scoreboard, 2 mailboxes and 2 virtual interfaces). It should have the following methods:

1. **new ()** - to assign virtual interfaces
2. **build ()** - Instantiate the Driver, Receiver, Scoreboard and the 2 mailboxes.
3. **reset ()** - Reset the DUT by driving all input signals low for 10 clocks and de-assert reset signal.
4. **start ()** - Start the Scoreboard.
5. **run ()** - Call the build, reset and start tasks.

```
class Environment;

        // TestBench components instantiations

        // virtual interfaces

        function new (…);

        task build (…);

        task reset (…);

        task start (…);

        task run (…);

endclass : Environment
```

Question 7: Create a program block called **test** and instantiate the Environment inside it. Write the TestCase to demonstrate FULL and EMPTY conditions.

Question 8: Create a SV wrapper around the DUT and connect the interface with signals of the DUT.

```
module dut_wrapper (<interfaces >)

        // instantiate the DUT and wire them

endmodule : dut_wrapper
```

Question 9: Develop and write atleast 4 test cases to validate the working of the FIFO.

Lab Submission:

Your lab must be in the following format:

1. It must be contained in a zip file. Rename the zip file to filename.**piz.**
2. The contents of the zip file must be exactly as follows:
    i. doc/Report.doc
    ii. rtl/ <all verilog design files for FIFO>
    iii. tb/class.sv
    iv. tb/test.sv
    v. tb/interface.sv
    vi. tb/tb_top.sv

The zip file must be named exactly as: **Lab1.<student_id>.<First_Name>_<Last_Name>.piz**

It must be e-mailed to the following e-mail accounts: ***Kyle.Gilsdorf@asu.edu***

NOTE: Clearly label and include the waveforms of your testcases in Report.doc

Include log files for different tests in the same folder.

You are free to make necessary changes to the method return types or arguments, if required.

Avoid using hierarchical references within your TestBench; it will impact its re-usability.