PROBING DUT WITH VIRTUAL INTERFACES

This is an example for probing the Design Under Test (DUT) with virtual interfaces. The SystemVerilog has deprecated the use of direct paths such as as the one shown below:

```
Execute_test_top.dut.Shifter_Inst.aluout
```

This is because this kind of probing causes lack of code re-usability which defeats the purpose of having classes.

Given that we are hoping to check the correctness of our golden model with the IOs of the internal blocks of the DUT, we need a means of probing down the hierarchy of the DUT. This is achieved by the use of virtual interfaces within the class. The following example gives you the necessary know-how to do this. The example is an extension of the code for Lab3 where we are going to attempt probing the IO connections of the Shifter Instance within the Execute unit. All the files remain the same except for the following:

http://www.ece.ncsu.edu/asic/asic_verification/av/Projects_files/Trial_Probing/Execute.if.sv

http://www.ece.ncsu.edu/asic/asic_verification/av/Projects_files/Trial_Probing/Execute.test_top.sv

http://www.ece.ncsu.edu/asic/asic_verification/av/Projects_files/Trial_Probing/Execute.tb.sv

http://www.ece.ncsu.edu/asic/asic_verification/av/Projects_files/Trial_Probing/Scoreboard.sv

Modifications in Interface file: In `Execute.if.sv`, we have created an interface `Shifter_Probe_if` for probing the shifter instance. Note that all the connections are of type input logic given that we are attempting observe and not drive any of the values within the DUT. In this case we are going to observe the `enable, in1, shift, operation` and `aluout` signals of the Shifter instance.

Modifications in Top level integration: In the `Execute.test_top.sv` file we have an instance of the interface (called `shifter_io`) which is connected to the correct hierarchy of the DUT as shown below:

```
Shifter_Probe_if shifter_io (       dut.Shifter_Inst.enable,
                                    dut.Shifter_Inst.in1,
                                    dut.Shifter_Inst.shift,
                                    dut.Shifter_Inst.operation,
                                    dut.Shifter_Inst.aluout
                             );
```

We also send a handle to interface into the test routine to make sure that it gets passed onto the class that needs to use the above values for checks (here `scoreboard`).

```
Execute_test test(top_io, shifter_io);
```

Modifications in class declaration: In this case, the scoreboard is the class that needs to gain access to the above signals. To this end, the Scoreboard needs to be augmented to handle virtual interfaces. Virtual interfaces are pointers to physical interfaces. The physical interface that can be assigned to the virtual interface can be decided at run-time. The `scoreboard.sv` file is modified to utilize virtual interfaces by:

- Declaring a virtual interface (here for type Shifter_Probe_If):
  ```
  virtual Shifter_Probe_if ports1; //PROBECHANGE
  ```

- Modify the constructor to accept an interface definition as an input
  ```
  extern function new(string name = "Scoreboard", virtual
  Shifter_Probe_if ports2);
  ```

  As another example, for the receiver (if it required this interface) the constructor would change to
  ```
  extern function new(string name = "Receiver", rx_box_type
  rx_out_box, virtual Execute_io.TB Execute, virtual
  Shifter_Probe_if ports1);
  ```

- The definition of `new()` would change to
  ```
  function Scoreboard::new(string name, virtual Shifter_Probe_if
  ports2);
  ```

- Within the constructor `new()` we would have the mapping of all of the ports in the interface to the local copy of the ports
  ```
  this.ports1 = ports2;
  ```

- To use the signals for checking we would now be able to access the signals using `ports1.aluout, ports1.shift, ports1.operation` etc

Modifications in testing program: The `Execute.tb.sv` file containing the test program is modified to accept the new interface and will also have to pass the interface onto the instance of the Scoreboard class. This leads to modifications of the program header to

```
program Execute_test(Execute_io.TB Execute, Shifter_Probe_if shift_io);
```

and the scoreboard instantiation would change to

```
sb = new("Scoreboard", shift_io);
```

In the above `shift_io` is the instance of the `Shifter_Probe_if` that was used in the header.