OVM

Driving into wires
Driving into signals (Driving_into_wires_md.sv)

1) module ports defined as direction "output" can be written from an
   - always block or
   - A single "assign statement within or outside the always

```vhdl
always @ (posedge clk) begin : FF_LD
    r_out <= d_in;
    assign d_out=8'b10X_XZZ0;
end : FF_LD
```

2) module ports defined as direction "inout"
   - must be driven by the "assign" statement.

```vhdl
assign data1 = ~ r_out;
assign k_out=8'b10X_XZZ0;
endmodule : ld_reg
```
module ld_reg #(SIZE=8) 
  (input logic clk, ld, 
   input logic [SIZE-1:0] d_in, 
   output logic [SIZE-1:0] r_out, d_out, k_out, 
   inout logic [SIZE-1:0] data1, data2, data3);

logic a, b=1'b1; // Local variable 
wire [SIZE-1:0] wdata1, wdata2, wdata3;
always @(posedge clk) begin : FF_LD 
  // The begin statement is only needed 
  // if multiple statements in body of always 
  r_out <= d_in;
  // data1 <= d_in; // illegal 
  // line above: A net is not a legal lvalue in this context 
  // wdata1 <= d_in; // illegal 
  // line above: A net is not a legal lvalue in this context 
  assign d_out=8'b10X_XZZ0;
end : FF_LD 
assign data1 = ~ r_out; 
assign k_out=8'b10X_XZZ0;
endmodule : ld_reg | 
Not recommended as a style from an always or always_ff with clocking 
OK from always_comb or always used as combinational
SV Interface  Driving_into_wires_if.sv

- Nets (declared as wire) behave the as wires, same as in modules.

```vhdl
parameter SIZE=8;

interface test_if (input logic clk);
  logic ld;
  logic[SIZE-1:0] d_in;
  logic[SIZE-1:0] r_out, r_out2;
  wire logic[SIZE-1:0] data1, data2, data3;
endinterface

module top2;
  bit clk;
  test_if m_if(clk);
  initial forever #10 clk=!clk;
  task t();
    @ (posedge clk) begin
      m_if.r_out <= m_if.d_in;
      //m_if.data1 <= m_if.d_in;
      // A net is not a legal lvalue in this context
    end
  endtask : t
endmodule
```
SV interface and classes

- Tasks in classes have the same rules as tasks or always blocks from modules when driving into interface variables.
- Exception for clocking blocks (see next slide)

```vhdl
parameter SIZE=8;
interface test_if (input logic clk);
  logic ld;
  logic[SIZE-1:0] d_in;
  logic[SIZE-1:0] r_out, r_out2;
  wire logic[SIZE-1:0] data1, data2, data3;
  clocking driver_cb @ (posedge clk);
end interface;

class c;

virtual interface test_if vif_bdl;
virtual task t();
  @ vif.driver_cb begin
    // vif_bdl.data3 <= 8'b10ZZ ZZ01; // illegal
    // line above: A net is not a legal lvalue in this context
  end
endtask : t
```
SV interface and clocking blocks

1800-2012 14.3 Clocking block declaration
- A clockvar whose clocking_direction is inout shall behave as if it were two clockvars, one input and one output, having the same name and the same clocking_signal.
- Reading the value of such an inout clockvar shall be equivalent to reading the corresponding input clockvar.
- Writing to such an inout clockvar shall be equivalent to writing to the corresponding output clockvar.

```vhdl
parameter SIZE=8;

interface test_if (input logic clk);
  logic ld;
  logic[SIZE-1:0] d_in;
  logic[SIZE-1:0] r_out, r_out2;
  wire logic[SIZE-1:0] data1, data2, data3;
  clocking driver_cb @ (posedge clk);
    default input #2 output #3;
  output data1, r_out2;
  input d_in;
  inout data2;
  endclocking : driver_cb
modport drvr_if_mp (clocking driver_cb);
endinterface : test_if
```
parameter SIZE=8;

interface test_if (input logic clk);
  logic ld;
  logic[SIZE-1:0] d_in;
  logic[SIZE-1:0] r_out, r_out2;
  wire logic[SIZE-1:0] data1, data2, data3;
  clocking driver_cb @ (posedge clk);
    default input #2 output #3;
    output data1, r_out2;
    input d_in;
    inout data2;
  endclocking : driver_cb
  modport drvr_if_mp (clocking driver_cb);
endinterface : test_if

class c;
  virtual interface test_if.drvr_if_mp vif;
  virtual interface test_if vif_bd1;
  virtual task t();
  @ vif.driver_cb begin
    vif.driver_cb.r_out2 <= vif.driver_cb.d_in;
    vif.driver_cb.data2 <= vif.driver_cb.d_in;
  end
  endtask : t
endclass : c
parameter SIZE=8;
interface test_if (input logic clk);
  logic ld;
  logic[SIZE-1:0] d_in;
  logic[SIZE-1:0] r_out, r_out2;
  wire logic[SIZE-1:0] data1, data2, data3;
clocking driver_cb @ (posedge clk);

module top2;
  bit clk;
  test_if m_if(clk);
  initial forever #10 clk=!clk;
  c c_h=new();

  task t();
  @ (posedge clk) begin
    m_if.r_out <= m_if.d_in;
    //m_if.data1 <= m_if.d_in;
    // A net is not a legal lvalue in this context
    end
  endtask : t