VHDL and Verilog for Modeling

Module 10
Overview

• General examples
  – AND model
  – Flip-flop model
  – SRAM Model

• Customizing Models
  – Generics in VHDL
    • DDR SDRAM Model
  – Parameters in Verilog

• Using Files in Verilog for Test Benches
VHDL for Modeling

- We have covered
  - VHDL for Synthesis
  - VHDL for testing (simulation)
- Now - VHDL and Verilog for modeling
- Describes the expected behavior of a component or device
- Can be used to test other components
  - for example a model of a CPU could be used to test:
    - UART
    - DRAM memory controller
    - cache controller
AND gate model

```vhdl
-- 74LS08 model (typical delays)
ENTITY ls08 IS
  PORT(a, b : IN std_logic;
       c : OUT std_logic);
END ls08;

ARCHITECTURE behav OF ls08 IS
BEGIN
  c <= '1' AFTER 8 ns WHEN a = '1' AND b = '1' ELSE 
      '0' AFTER 10 ns;
END behav;
```
Simulation Results
D flip-flop model

-- 74LS74 model (typical delays)
ENTITY ls74 IS
    PORT(d, clr, pre, clk : IN std_logic;
         q :  OUT std_logic);
END ls74;

ARCHITECTURE behav OF ls74 IS
BEGIN
    PROCESS(clk, clr, pre)
    BEGIN
        IF clr = '0' THEN
            q <= '0' AFTER 25 ns;
        ELSIF pre = '0' THEN
            q <= '1' AFTER 13 ns;
        ELSIF clk'EVENT AND clk = '1' THEN
            IF d = '1' THEN
                q <= '1' AFTER 13 ns;
            ELSE
                q <= '0' AFTER 25 ns;
            END IF;
        END IF;
    END PROCESS;
END behav;
Simulation Results
Adding Constants - same as previous

```vhdl
-- 74LS74 model (typical delays)
ENTITY ls74 IS
    PORT(d, clr, pre, clk : IN std_logic;
         q : OUT std_logic);
    CONSTANT t_rise : TIME := 13 ns;
    CONSTANT t_fall : TIME := 25 ns;
END ls74;

ARCHITECTURE behav OF ls74 IS
BEGIN
    PROCESS(clk, clr, pre)
    BEGIN
        IF clr = '0' THEN
            q <= '0' AFTER t_fall;
        ELSIF pre = '0' THEN
            q <= '1' AFTER t_rise;
        ELSIF clk'EVENT AND clk = '1' THEN
            IF d = '1' THEN
                q <= '1' AFTER t_rise;
            ELSE
                q <= '0' AFTER t_fall;
            END IF;
        END IF;
    END PROCESS;
END behav;
```
Adding setup and pulse width checks

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ls74 IS
  PORT(d, clr, pre, clk : IN std_logic;
       q : OUT std_logic);
  CONSTANT t_rise : TIME := 13 ns;
  CONSTANT t_fall : TIME := 25 ns;
  CONSTANT t_setup : TIME := 20 ns;
  CONSTANT t_width : TIME := 25 ns;
END ls74;
Basic D flip-flop description same

ARCHITECTURE behav OF ls74 IS
BEGIN

PROCESS(clk, clr, pre)
BEGIN
  IF clr = '0' THEN
    q <= '0' AFTER t_fall;
  ELSIF pre = '0' THEN
    q <= '1' AFTER t_rise;
  ELSIF clk'EVENT AND clk = '1' THEN
    IF d = '1' THEN
      q <= '1' AFTER t_rise;
    ELSE
      q <= '0' AFTER t_fall;
    END IF;
  END IF;
END PROCESS;

Jim Duckworth, WPI 10  VHDL and Verilog for Modeling - Module 10
Add setup and pulse width checks

```vhdl
-- process to check data setup time
PROCESS(clk)
BEGIN
  IF clk'EVENT AND clk = '1' THEN
    ASSERT d'LAST_EVENT > t_setup
    REPORT "D changed within setup time"
    SEVERITY ERROR;
  END IF;
END PROCESS;

-- process to check clock high pulse width
PROCESS(clk)
  VARIABLE last_clk : TIME := 0 ns;
BEGIN
  IF clk'EVENT AND clk = '0' THEN
    ASSERT NOW - last_clk > t_width
    REPORT "Clock pulse width too short"
    SEVERITY ERROR;
  ELSE
    last_clk := NOW;
  END IF;
END PROCESS;
END behav;
```
Complete LS74 Model

```
entity ls74_model is
  Port (G : in std_logic;
        clk : in std_logic;
        en : in std_logic;
        r : in std_logic;
        q : out std_logic);
constant t_rise : time := 12 ns;
constant t_fall : time := 12 ns;
constant t_setup : time := 10 ns;
constant t_width : time := 10 ns;
end ls74_model;
architecture behavioral of ls74_model is
begin
  process(clk, en, pres)
  begin
    if clk = '0' then
      q <= '0' after t_fall;
    elsif pres = '0' then
      q <= '1' after t_fall;
    elsif clk'event and clk = '1' then
      if q = '1' then
        q <= '1' after t_fall;
      elsif q = '0' after t_fall;
      end if;
    end if;
  end process;
  -- process to check data setup time
  process(clk)
  begin
    if clk'event and clk = '1' then
      report "Data changed within setup time";
      severity error;
    end if;
  end process;
  -- process to check clock high pulse width
  process(clk)
  variable last_q : time := 0 ns;
  begin
    if clk'event and clk = '0' then
      report "Clock pulse width too short";
      severity error;
    else
      last_q := now;
    end if;
  end process;
end Behavioral;
```
Adding the SRAM model

- New testbench
Generics

- Generics useful for making design units more general purpose
- Generics allow information to be passed into a design description
- Example information
  - propagation delays
  - size of component (changing input and output ports)
LS08 example with generics

-- 74LS08 model
ENTITY ls08 IS
  GENERIC(t_rise, t_fall : TIME);
  PORT(a, b : IN std_logic;
       c : OUT std_logic);
END ls08;

ARCHITECTURE behav OF ls08 IS
BEGIN
  c <= '1' AFTER t_rise WHEN a = '1' AND b = '1' ELSE
       '0' AFTER t_fall;
END behav;

• This is now a parameterized model (general purpose) rather than hard-coded version
• Actual delay is determined at simulation time (or synthesis) by value passed to model
-- example use of LS08 with generic values

ENTITY test IS
    PORT(in1, in2, in3 : IN std_logic;
         out1     : OUT std_logic);
END test;

ARCHITECTURE behav OF test IS
    COMPONENT ls08
        GENERIC(t_rise, t_fall : TIME);
        PORT(a, b : IN std_logic;
             c   : OUT std_logic);
    END COMPONENT;
    SIGNAL int : std_logic;
BEGIN
    u1: ls08 GENERIC MAP(8 ns, 10 ns) -- typical delays
        PORT MAP(a => in1, b => in2, c => int);
    u2: ls08 GENERIC MAP(15 ns, 20 ns) -- max delays
        PORT MAP(a => int, b => in3, c => out1);
END behav;
Another Example (with default values)

- Can provide default values for generics
- Only need GENERIC MAP if necessary to change them

ENTITY and2 IS
  GENERIC(t_rise, t_fall : TIME := 10 ns;
           load : INTEGER := 3);
  PORT(a, b IN : IN std_logic;
       c : OUT std_logic);
END and2;

ARCHITECTURE generic_model OF and2 IS
BEGIN
  c <= '1' AFTER (t_rise + (load * 2 ns)) WHEN a = '1' AND b = '1'
     ELSE '0' AFTER (t_fall + (load * 2 ns));
END generic_model;

u1: and2 GENERIC MAP(5 ns, 7 ns, 4) PORT MAP (a => ......); -- default overridden
u2: and2 PORT MAP(a => ......); -- uses default values
Modifying Component Size (also for synthesis)
Making two copies – different size
Schematic of two shift registers
Double Data Rate (DDR) SDRAM

MT46V32MW4 - 8 Meg x 4 x 4 Banks
MT46V16MW8 - 4 Meg x 8 x 4 Banks
MT46V32M16 - 2 Meg x 16 x 4 Banks

For the latest data sheet revisions, please refer to the Micron Website: www.micron.com/ddr

Features
128MB: x4, x8, x16 - DDR SDRAM

- VDD = +2.5V ±0.3V, VDDQ = ±2.5V ±0.3V
- VDD = +2.5V ±0.3V, VDDQ = ±2.5V ±0.3V (DDR-400)
- Differential data strobe (DQS) transmitted/received with data - i.e., source synchronous data capacitor (20nF) per line per cycle
- Internal, pipelined double-data-rate (DDR) architecture for data access within each clock cycle
- Differential clock inputs CK and CK' (DQ)
- Commands stored on each positive CK edge
- DQS edge aligned with data for READs, center aligned with data for WRITEs
- DLL not align DQ and DQS transitions with CK
- Four internal banks for concurrent operation
- Data mask (DM) for enabling multi-bank (4 of 8, 8 of 16, 16 of 32) per row
- Programmed burst length: 2, 4, or 8
- Auto Refresh and Self Refresh Modes
- Lower level TSP for improved reliability (OCP)
- 2.5V/1.8V (OCP compatible)
- Automatic write precharge option is supported
- 800 clock cycles supported (20MHz, 100ns)

Options
- Configuration
  - 8 Meg x 4 (8 Meg x 8 banks)
  - 16 Meg x 4 (4 Meg x 8 banks)
  - 32 Meg x 4 (2 Meg x 8 banks)
- DRAM Type: RDRAM
- DQ: 32-bit (128-bit)
- Power-up: Self-refresh
- Timing: Cycle Time
  - tRAS ≤ 3.5 (DDR-400)
  - tRAS ≤ 3.5 (DDR-400)
  - tRAS ≤ 3.5 (DDR-400)
  - tRAS ≤ 3.5 (DDR-400)

Marking
- Standard: T
- Low Power: T
- Temperature Range: 0°C to 70°C
- Revision: 

Table 1: Configuration Addressing

<table>
<thead>
<tr>
<th>Configuration</th>
<th>8 Meg x 4</th>
<th>16 Meg x 4</th>
<th>32 Meg x 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh Count</td>
<td>6043</td>
<td>6043</td>
<td>6043</td>
</tr>
<tr>
<td>Row Addressing</td>
<td>4K (0x-1F)</td>
<td>4K (0x-1F)</td>
<td>4K (0x-1F)</td>
</tr>
<tr>
<td>Bank Addressing</td>
<td>4 (0x-3F)</td>
<td>4 (0x-3F)</td>
<td>4 (0x-3F)</td>
</tr>
<tr>
<td>Column Addressing</td>
<td>16 (0x-FF)</td>
<td>16 (0x-FF)</td>
<td>16 (0x-FF)</td>
</tr>
</tbody>
</table>

Table 2: Key Timing Parameters

CL = CAS Activation Time. Minimum clock rate: 8 CL = 2 x TCK - 75ns. CL = 2.5 = 2.5 x TCK - 75ns. CL = 3 = 3 x TCK - 75ns

<table>
<thead>
<tr>
<th>Speed Grade</th>
<th>CL = 2</th>
<th>CL = 2.5</th>
<th>CL = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>300 MHz</td>
<td>2ns</td>
<td>2.5ns</td>
<td>3ns</td>
</tr>
<tr>
<td>200 MHz</td>
<td>2ns</td>
<td>2.5ns</td>
<td>3ns</td>
</tr>
<tr>
<td>150 MHz</td>
<td>2ns</td>
<td>2.5ns</td>
<td>3ns</td>
</tr>
<tr>
<td>100 MHz</td>
<td>2ns</td>
<td>2.5ns</td>
<td>3ns</td>
</tr>
</tbody>
</table>

Jim Duckworth, WPI

21 VHDL and Verilog for Modeling - Module 10
DDR SDRAM Model (cont’d)

Functional Block Diagrams

Figure 1: 32 Meg x 4
Figure 11: READ Burst

Notes:
1. DO n = data-out from column n.
2. BL = 4.
3. Three subsequent elements of data-out appear in the programmed order following DO n.
4. Shown with nominal tAC, tDQSC, and tDSQ.

DDR SDRAM Model (cont’d)
DDR SDRAM Model (cont’d)
```
-- Control Logic

IF Sys_clk'EVENT AND Sys_clk = '1' THEN
    -- tRAP = tRAS_min - (Burst Length \* tCK / 2)
    IF Burst_length_2 = '1' THEN
        tRAP := tRAS - [4 \* tCK / 2];
    END IF;
    IF Burst_length_4 = '1' THEN
        tRAP := tRAS - [4 \* tCK / 2];
    END IF;
    IF Burst_length_8 = '1' THEN
        tRAP := tRAS - [8 \* tCK / 2];
    END IF;

    -- Auto Refresh
    IF Arcf_enable = '1' THEN
        -- Auto Refresh to Auto Refresh
        ASSERT (NOW = RFC_tbl := 'tRFC')
        REPORT "RFC Violation during Auto Refresh"
        SEVERITY WARNING;
    END IF;

    -- Freecharge to Auto Refresh
    ASSERT (NOW = RF_chk0 := 'tRP') AND (NOW = RF_chk1 := 'tRP') AND
    (NOW = RF_chk2 := 'tRP') AND (NOW = RF_chk3 := 'tRP')
    REPORT "RFC Violation during Auto Refresh"
    SEVERITY WARNING;

    -- Freecharge to Auto Refresh
    ASSERT (Pe_b0 = '1' AND Pe_b1 = '1' AND Pe_b2 = '1' AND Pe_b3 = '1')
    REPORT "All banks must be Freecharged before Auto Refresh"
    SEVERITY WARNING;

    -- Record current tRFC time
    RFC_chk := NOW;
END IF;

-- Extended Load Node Register
IF Ext_mode_enable = '1' THEN
    IF (Po_b0 = '1' AND Po_b1 = '1' AND Po_b2 = '1' AND Po_b3 = '1') THEN
        IF (Addr = '0') THEN
            DLL_enable := '1';
        ELSE
            DLL_enable := '0';
        END IF;
    END IF;
END IF;

-- Freecharge to ERR
ASSERT (Pe_b0 = '1' AND Pe_b1 = '1' AND Pe_b2 = '1' AND Pe_b3 = '1')
REPORT "All banks must be Freecharged before Extended Mode Register";
```
Verilog Lower Level Module with Parameters

```verilog
module xyz(
    input clk,
    input reset,
    output reg x,
    output reg y,
    output reg z
);

parameter X = 1175; // pass in the divider constants
parameter Y = 1175;
parameter Z = 1175;

reg [10:0] divider_x; // to divide by 1765
reg [10:0] divider_y; // to divide by 1765
reg [10:0] divider_z; // to divide by 1765

// X square wave output
always @ (posedge reset, posedge clk)
    if (reset)
        divider_x <= 11'b0;
    else if (divider_x == X)
        divider_x <= 11'b0;
    else
        divider_x <= divider_x + 1;
```

Jim Duckworth, WPI
parameter Y_2 = 1149 - 1;
parameter Z_2 = 1148 - 1;
parameter X_3 = 1126 - 1;
parameter Y_3 = 1125 - 1;
parameter Z_3 = 1124 - 1;

wire clk;
wire tx_i_x, tx_2_x, tx_3_x;
wire tx_i_y, tx_2_y, tx_3_y;
wire tx_i_z, tx_2_z, tx_3_z;

// instantiate the dcm
dcm dcm_1 |
  .ClkIN_IN(fpga_clk),
  .RST_IN(reset),
  .ClkFX_OUT(clk), // 4X input = 200MHz
  .ClkIN_IBUF_OUT(),
  .ClkO_OUT(),
  .Clk2X_OUT(),
  .Locked_OUT();

// instantiate 3 copies of the XYZ transmitter module
xyz #(.X(X_1), .Y(Y_1), .Z(Z_1) | tx_1{ .clk (clk), .reset(reset), .x(tx_i_x), .y(tx_i_y), .z(tx_i_z) });
xyz #(.X(X_2), .Y(Y_2), .Z(Z_2) | tx_2{ .clk (clk), .reset(reset), .x(tx_2_x), .y(tx_2_y), .z(tx_2_z) });
xyz #(.X(X_3), .Y(Y_3), .Z(Z_3) | tx_3{ .clk (clk), .reset(reset), .x(tx_3_x), .y(tx_3_y), .z(tx_3_z) });

always @ (posedge clk)
  case(#2:Cl)
    3'b000 :
      // Course Name: VHDL and Verilog for Modeling
      // Module: 10
      // Author: Jim Duckworth, WPI
Three copies of XYZ module

```vhdl
dce << clk;
wire tx_1_x, tx_2_x, tx_3_x;
wire tx_1_y, tx_2_y, tx_3_y;
wire tx_1_z, tx_2_z, tx_3_z;

// instantiate the dcm
dce dcm_1 (  
 .CLKIN(clk),  
 .RESET_IN(reset),  
 .CLKOUT0(clk), // 4X input = 300MHz  
 .CLKOUT1(),  
 .CLKOUT2(),  
 .CLKOUT3();  
);

// instantiate 3 copies of the XYZ transmitter module
xrc #(.X(X_1), .Y(Y_1), Z(Z_1)) tx_1 (.clk (clk), .reset(reset),
xrc #(.X(X_2), .Y(Y_2), Z(Z_2)) tx_2 (.clk (clk), .reset(reset),
xrc #(.X(X_3), .Y(Y_3), Z(Z_3)) tx_3 (.clk (clk), .reset(reset),
always @(posedge clk)
  case[0b0] {
    0'1b0:  

Jim Duckworth, WPI

28 VHDL and Verilog for Modeling - Module 10
RTL Schematic showing 3 modules
(Misc) Using UCF to specify pin options

```plaintext
ISE Text Editor (M.63c) - [mantenna_tx.ucf*]

6 NET "dip<0>" LOC = "V8"; # SW0
7 NET "dip<1>" LOC = "U10"; # SW1
8 NET "dip<2>" LOC = "U8"; # SW2
9
10 # ==== 6-pin header J18 ====
11 NET "tx_x" LOC = "AA21" | IOSTANDARD = LVC莫斯33 | SLEW = SLOW | DRIVE = 8 ;
12 NET "tx_y" LOC = "AB21" | IOSTANDARD = LVC莫斯33 | SLEW = SLOW | DRIVE = 8 ;
13 NET "tx_z" LOC = "AA19" | IOSTANDARD = LVC莫斯33 | SLEW = SLOW | DRIVE = 8 ;
14
15 # reset - use bottom push button
16 NET "reset" LOC = "T15" | IOSTANDARD = LVC莫斯33 | PULLDOWN ;
```

Jim Duckworth, WPI

30 VHDL and Verilog for Modeling - Module 10
List 1
World 1

Block 1 - View Site AA21 with Comp tx_x

For Help, press F1

Jim Duckworth, WPI
Timing Check Tasks in Verilog

- Specify block can be used to specify setup and hold times for signals
  - `specify` and `endspecify` (Use `specparam` to define parameters in specify block)
- `$setup (data, clock edge, limit)
  - Displays warning message if setup timing constraint is not met
  - `$setup(d, posedge clk, 10)"
- `$hold (clock edge, data, limit)
  - Displays warning message if hold timing constraint is not met
  - `$hold(d, posedge clk, 2)"
- `$width (pulse event, limit)
  - Displays warning message if pulse width is shorter than limit
  - `$width(d, posedge clk, 20) – specify start edge of pulse`
- `$period (pulse event, limit) – specify start edge of pulse`
  - Check if period of signal is sufficiently long
  - `$period(posedge clk, 50)"
Original Simple SRAM model

```vhdl
module sram_model(
  input [9:0] addr,
  inout [7:0] data,
  input oe_n,
  input we_n
);

// this model simulates how we expect the external SRAM to respond
// we will add some simple checks and delays later

// create an array of 1024 memory cells, each 8 bits wide
reg [7:0] sram[0:1023];

// load data on rising edge of we signal
always @ (posedge we_n)
  sram[addr] = data;

// create tri-state signal for data
// drive data lines when oe_n is low, else tri-state
assign data = (oe_n == 0) ? sram[addr] : 8'b0;

endmodule
```
Adding Timing Checks and Delay

```vhdl
input we_n;

specify
  specparam tUP = 45,  // Min write pulse width;
  tDW = 20,  // Data setup time before write
  tDH = 0;  // Data hold time after write
$setup(data, posedge we_n, tDW);  // verify setup time requirements
$hold(posedge we_n, data, tDH);
$width(negedge we_n, 45);  // can just add value
endspecify

// this model simulates how we expect the external SRAM to respond
// we will add some simple checks and delays later

// create an array of 1024 memory cells, each 8 bits wide
reg [7:0] sram[0:1023];

// load data on rising edge of we signal
always @(posedge we_n)
  if (ce_n == 0)
    sram[addr] = data;

// create tri-state signal for data
// drive data lines when ce_n is low, else tri-state
// delay data output by 70ns to model SRAM access time
assign #70 data = (oe_n == 0 & ce_n == 0) ? sram[addr] : 8'b0;

endmodule
```
Testing with Test Bench
### Warning Messages

<table>
<thead>
<tr>
<th>Console</th>
<th>0003 : 78 : 56 : 34 : 12 : 970ns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1230 : 78 : 56 : 34 : 12 : 1000ns</td>
</tr>
<tr>
<td><strong>ISim&gt; run 1us</strong></td>
<td>01zz : 78 : 56 : 34 : 12 : 1100ns</td>
</tr>
<tr>
<td></td>
<td>0107 : 78 : 56 : 34 : 12 : 1170ns</td>
</tr>
<tr>
<td></td>
<td>1230 : 78 : 56 : 34 : 12 : 1200ns</td>
</tr>
</tbody>
</table>

Write to SRAM - location 1 (width violation)

```
WARNING: at 1440 ns: Timing violation in /risc_sram_tf/sram/ $width( we_n:1400 ns, :1440 ns, 45 ns)
```

Write to SRAM - location 0 (setup violation)

```
WARNING: at 1740 ns: Timing violation in /risc_sram_tf/sram/ $setup( data:1725 ns, we_n:1740 ns,20 ns)
```

```
WARNING: at 1740 ns: Timing violation in /risc_sram_tf/sram/ $setup( data:1725 ns, we_n:1740 ns,20 ns)
```

```
WARNING: at 1740 ns: Timing violation in /risc_sram_tf/sram/ $setup( data:1725 ns, we_n:1740 ns,20 ns)
```

```
3412 : 78 : 56 : 34 : 12 : 1840ns
```

**ISim>**
ISSI SRAM – Verilog Model (partial)

- // IS61LV25616 Asynchronous SRAM, 256K x 16 = 4M; speed: 10ns.
- // Note; 1) Please include "+define+ OEb" in running script if you want to check
- // timing in the case of OE_ being set.
- // 2) Please specify access time by defining tAC_10 or tAC_12.

- // `define OEb
- `define tAC_10
- `timescale 1ns/10ps

module IS61LV25616 (A, IO, CE_, OE_, WE_, LB_, UB_);

- parameter dqbits = 16;
- parameter memdepth = 262143;
- parameter addbits = 18;
- parameter Toha  = 2;
- parameter Tsa   = 2;

- `ifdef tAC_10
- parameter Taa   = 10,
- Thzce = 3,
- Thzwe = 5;
- `endif

- `ifdef tAC_12
- parameter Taa   = 12,
- Thzce = 5,
- Thzwe = 6;
- `endif

- input CE_, OE_, WE_, LB_, UB_;
- input [(addbits - 1) : 0] A;
- inout [(dqbits - 1) : 0] IO;

- wire [(dqbits - 1) : 0] dout;
- reg [(dqbits/2 - 1) : 0] bank0 [0 : memdepth];
Micron SRAM on Nexys2 board
Complete model is > 1300 lines!
```vhdl
// Memory arrays
reg [DQ_BITS-1:0] memory [31:0][MEN_BITS-1:0];
reg [ADDR_BITS-1:0] memory_addr [31:0][MEN_BITS-1:0];
reg [MEN_BITS:0] memory_index;
reg [MEN_BITS:0] memory_used;

// Software access registers
reg [1:0] software_access_unlock;
reg [1:0] software_access WHICH_REG;
reg software_access FOR_WRITE;

// System registers
parameter IDLE = 0;
parameter WR = 1;
parameter RD = 2;
parameter CFG_RD = 3;
parameter CFG_WR = 4;

// Asynchronous registers
reg [2:0] async_state;
reg async_wr_lockout;
wire data_out_enable;
wire data_out_valid;
wire wait_out_enable;
wire wait_out_valid;
reg async_crs;
reg [ADDR_BITS-1:0] async_addr;
reg deep_power_down_exit;
reg tow_check;
reg tewh_check;

// Synchronous registers
reg async_access;
reg last_access;
reg last.wr;
reg [ADDR_BITS-1:0] async_addr;
```
Jim Duckworth, WPI

41 VHDL and Verilog for Modeling - Module 10

---

```vhdl
-- Parameters current with P26Z datasheet rev H
-- Timing parameters based on Speed Grade

    // SYMBOL UNITS DESCRIPTION
    // ______ ______ ______

`ifndef sg7013
  parameter tAClk = 7.0; // ns CLK to output delay
  parameter tAPA = 20.0; // ns Page access time
  parameter tAW = 70.0; // ns Address valid to end of WRITE
  parameter tBW = 70.0; // ns BY# select to end of WRITE
  parameter tCBPH = 5.0; // ns CE# HIGH between subsequent burst or mixed-mode operations
  parameter tCLK = 7.5; // ns CLK period
  parameter tCSP = 2.5; // ns CE# setup time to active CLK edge
  parameter tCW = 70.0; // ns Chip enable to end of WRITE
  parameter tDH = 1.5; // ns Hold time from active CLK edge
  parameter tKP = 3.0; // ns CLK HIGH or LOW time
  parameter tPC = 20.0; // ns Page READ cycle time
  parameter tRC = 70.0; // ns READ cycle time
  parameter tS = 2.0; // ns Setup time to active CLK edge
  parameter tVP = 5.0; // ns ADV# pulse width LOW
  parameter tVS = 70.0; // ns ADV# setup to end of WRITE
  parameter tWC = 70.0; // ns WRITE cycle time
  parameter tWP = 45.0; // ns WRITE pulse width
`else `ifdef sg701
  parameter tAClk = 7.0; // ns CLK to output delay
  parameter tAPA = 20.0; // ns Page access time
```
always @ (posedge ce_n or negedge oe_n)
begin
    sync_access <= '1'b0; // a burst is terminated by bringing CE# HIGH for greater than 15ns.
end

// Main Asynchronous block
always @ (sync_access or ce2v1 or we_n or oe_n or by_n or by_n_in or async_ce or async_addr)
begin
    if (sync_access) begin
        software_access_unlock <= '2'b0;
        async_state = IDLE;
    end else begin

    // commit write to memory
    if (async_state == WR) & (last_ce2v1 & !ce2v1) || (!last_we_n & we_n) || (~last_by_n & by_n) begin
        if ($realtime - tm_ce2v < tCE)
            $display("%t ERROR: CE# violation on CE# by %t", $realtime, tm_ce2v + tCE - $realtime);
        if ($realtime - tm_adv_neg < tVS)
            $display("%t ERROR: VS# violation on VS# by %t", $realtime, tm_adv_neg + tVS - $realtime);
        if ($realtime - tm_ce2v < tBU)
            $display("%t ERROR: BU# violation on BU# by %t", $realtime, tm_ce2v + tBU - $realtime);
        if ($realtime - tm_ce2v < tUP)
            $display("%t ERROR: UP# violation on UP# by %t", $realtime, tm_ce2v + tUP - $realtime);
        if ($realtime - tm_async_ce < tAW)
            $display("%t ERROR: AW# violation on AW# by %t", $realtime, tm_async_ce + tAW - $realtime);
        if ($realtime - tm_async_addr < tAW)
            $display("%t ERROR: AW# violation on ADDR by %t", $realtime, tm_async_addr + tAW - $realtime);
        if ($realtime - tm_dq_tdw < tDW)
            $display("%t ERROR: DW# violation on DW# by %t", $realtime, tm_dq_tdw + tDW - $realtime);
        if ($realtime - tm_async_addr < tAW & $realtime - tm_dq_tdw < tDW)
            $display("%t ERROR: DW# violation on ADDR by %t", $realtime);
    end
begin
    if (async_addr == (ADDR_BEST[1:1]'b1)) & (software_access_unlock == 2) begin
        case (last_dq[1:0])
            (2'b00 : software_access which_reg <= DCR;
            2'b10 : DIBA_MASK[17:16] : software_access which_reg <= DIBA;
            2'b11 : DIBA_MASK[17:16] : software_access which_reg <= DCR;
            default: $display("%t ERROR: Illegal Register Select = %b", $realtime, last_dq[1:0].endcase
    if (DEBUG(0))
        $display("%t INFO: Async - Software Access Unlock = %d, Register Select = %d", $realtime, software_access unlock <= software_access unlock + 1
        and also <= software_access which_reg <= DCR);
end
Jim Duckworth, WPI

42 VHDL and Verilog for Modeling - Module 10
Additional Verilog System Tasks

- `@ (posedge clk)` – event control, wait for rising edge of clk
- `wait (state == ENTER && ready == 1’b1)` – wait for specific condition
- `#123` – wait for 123 timescale units
- `$display` (print out information)
- `$monitor` (displays information when signal changes value)
- `$time` and `$realtime`
- `$fopen` and `$fclose`
- `Integer log_file = $fopen(“my_log”)`
- `$readmemb` and `$readmemh`
Decoder Test Bench in Verilog with Files

```verilog
module decoder_tf;

    // Inputs
    reg [2:0] sel;

    // Outputs
    wire [7:0] y;

    parameter NUM_VECTORS = 8;

    // multi-channel descriptors associated with files
    integer results_file, console_file, out_files;

    integer i; // for loop variable

    reg [7:0] y_expected;

    reg [10:0] vectors_mem [0:NUM_VECTORS-1]; // storage for vectors from file

    // Instantiate the Unit Under Test (UUT)
    decoder uut (  
        .sel(sel),  
        .y(y)  
    );

    initial
    begin
        // Setup output file
        results_file = $fopen("results_file.txt");
        if |results_file == 0|
```

Jim Duckworth, WPI  44  VHDL and Verilog for Modeling - Module 10
Opening file in Verilog

```verilog
initial
begin

    // setup output file
    results_file = $fopen("results_file.txt");
    if (results_file == 0)
        $display("Cannot open log file");
    console_file = 32'b0000_0001; // console output
    out_files = results_file | console_file;

    // read vectors from vector file
    $readmem("test_vec.txt", vectors_mem);

    for (i=0; i < NUM_VECTORS; i=i+1)
        begin
            // now get sel input and apply to decoder
            sel = vectors_mem[i][10:8]; // top 3 bits for input;
            #50;
            // now check if decoder outputs are correct
            y_expected = vectors_mem[i][7:0];
            if (y != y_expected)
                begin
                    $display("Error output incorrect: y = "y, y_expected = "y at time "$time;
                    (0:xdisplay(results_file, "Error output incorrect: y = "y, y_expected = "y at time "$time);
                end
            end
        end
    $fclose(results_file);
end
```

Jim Duckworth, WPI
for (i=0; i < NUM_VECTORS; i=i+1)
begin
// now get sel input and apply to decoder
sel = vectors_mem[i][10:8];  // top 3 bits for input:
#50;
// now check if decoder outputs are correct
y_expected = vectors_mem[i][7:0];
if (y != y_expected)
begin
    $display("ERROR output incorrect: y = %b, y expected = %b at time %d ns", y, y_ex:
    $display(results_file, "ERROR output incorrect: y = %b, y expected = %b at time :
end
$fclose(results_file);
$stop;
end

initial // display information on console and also in file
begin
    $display(out_files, "Results from Decoder testing, October 13, 2009");
    $display(out_files, "time (ns) : input to decoder, output from decoder");
    $tmonitor(out_files, "%10d : %b    %b", $time, sel, y);
end
Waveform Generated
Console (and results file)

Results from Decoder testing, October 13, 2009
Time (ns) : input to decoder, output from decoder
0 : 000 00000001
50 : 001 00000010
100 : 010 0000100
150 : 011 0001000
200 : 100 0010000
250 : 101 0010000
Error: output incorrect: y = 00100000, y expected = 00110000 at time 300 ns
300 : 110 0100000
350 : 111 1000000