VHDL And Synthesis Review
VHDL In Detail

• Things that we will look at:
  – Port and Types
  – Arithmetic Operators
  – Design styles for Synthesis
VHDL Ports

• Four Different Types of Ports
  – in:
    • signal values are read-only
  – out:
    • signal values are write-only
    • Possible to have multiple drivers <- Depends on type of port, will discuss later
  – buffer:
    • comparable to out
    • signal values may be read, as well
    • only 1 driver
  – inout:
    • bidirectional port
Types of VHDL Ports / Signals

• VHDL a strongly typed language, so all types assigned between signals and ports have to match

• Standard VHDL Types:
  – type BOOLEAN is (FALSE, TRUE);
  – type BIT is (‘0’, ‘1’);
  – type CHARACTER is (– ascii set);
  – type INTEGER is range
    -- implementation_defined
  – type REAL is range
    -- implementation_defined
  – BIT_VECTOR, STRING, TIME
VHDL Types

• However instead of using bits, or bit vectors better to use the Std_logic_1164 types:
  – Instead of a simple 0 or 1 have more states that you can use during simulation to get more information on what is going on with design

  – TYPE STD_ULOGIC IS (  
      `U`, -- uninitialized  
      `X`, -- Forcing Unknown  
      `0`, -- Forcing 0  
      `1`, -- Forcing 1  
      `Z`, -- High Impedance  
      `W`, -- Weak Unknown  
      `L`, -- Weak 0  
      `H`, -- Weak 1  
      `\-\`, -- don’t care);  

  – To use Std_logic_1164 need to include its library with command
    • use IEEE.std_logic_1164.all;
  – Should use Std_logic type on all input/output ports
Std_Logic_1164

- Two different base types for the library
  - STD_ULOGIC AND STD_ULOGIC_VECTOR
    - Gives you error messages in cases of multiple concurrent signal assignments
    - More work to do arithmetic as type conversions are needed, some libraries predefined for std_logic
  - STD_LOGIC AND STD_LOGIC_VECTOR
    - When multiple signal assignments present uses resolution function to decide which signal ‘wins’ over the other signal and is assigned, or whether an unknown signal is assigned
    - Can use the port mode “buffer” to overcome the resolution function being used (buffer allows only a single signal driver)
Arithmetic Operations

• Arithmetic operators you will want to use are included in the NUMERIC_STD package
  – Include its libraries with this command
    • use ieee.numeric_std.all;

• Have two basic modes of operation
  – Signed : 2-complement (sign+absolute value)
  – Unsigned : binary representation of positive integers
Arithmetic Operators

• Might have to convert std_logic_vector to signed or unsigned type if compiler complains

• Typecasting to unsigned
  – someSignalUnsignedType <= 6 + unsigned(someSignal)

• NOTE: Can use Logical and Comparison operators directly on std_logic and std_logic_vector types without type conversion!
  – i.e. – not, and, <, =, /=
Understanding Synthesis

• How does design translate during synthesis?
  – Sensitivity list is usually ignored during synthesis (It is NOT ignored during simulation)
  – A bad sensitivity list could give you a design that looks like it works during simulation, but could synthesize to something that doesn’t!
Understanding Synthesis

• Latches
  – Happen when you have incomplete assignment statements, synthesizer then infers that you want to have memory (this is in an un-clocked process)
  – Example:
    architecture WRONG of MUX is
    begin
      process (A, B, SEL)
      begin
        if SEL = `1` then
          Z <= A;
        end if;
      end process;
    end WRONG;

  – Why is a latch generated for Z instead of a logic path?
Understanding Synthesis

• Correct way to do assignment statements with “if statements”
  architecture OK_1 of MUX is
  begin
    process (A, B, SEL)
    begin
      Z <= B;
      if SEL = `1` then
        Z <= A;
      end if;
    end process;
  end OK_1;

• For CASE statements, can use the default case (“when others”), and assign signals there to avoid latches
Understanding Synthesis

• Clocked Processes
  – In synthesis all assignments to signals inside a clocked process will make those signals stored as memory (flip-flop, registers)
  – Have to follow a certain format for this to happen
    • Need to have a clock event in the sensitivity list (or wait on clk’event statements)
    • For single edge triggered need to check for specified edge (can do dual edge triggered by checking for both edges)
Understanding Synthesis

- Example of Register
  - process(CLK, RST)
    begin
      if (CLK`event and CLK=``1``) then
        -- combinatorics
        end if;
    end process;

OR
  - Here can use function to check for rising edge of clock, synthesizer may or may not support this
  - process(CLK, RST)
    begin
      if (rising_edge(CLK) then
        -- combinatorics
        end if;
    end process;

- **Cant have any other elsif, else in main outer if block!! Nested if blocks ok!!**
Understanding Synthesis

• Example of asynchronous reset FlipFlop

process(CLK, RST)
begin
  if (RST = `1`) then
    -- asynchronous register reset
  elsif (CLK`event and CLK=`1`) then
    -- combinatorics
      --Signals assigned here inferred as FF’s, reg
  end if;
end process;