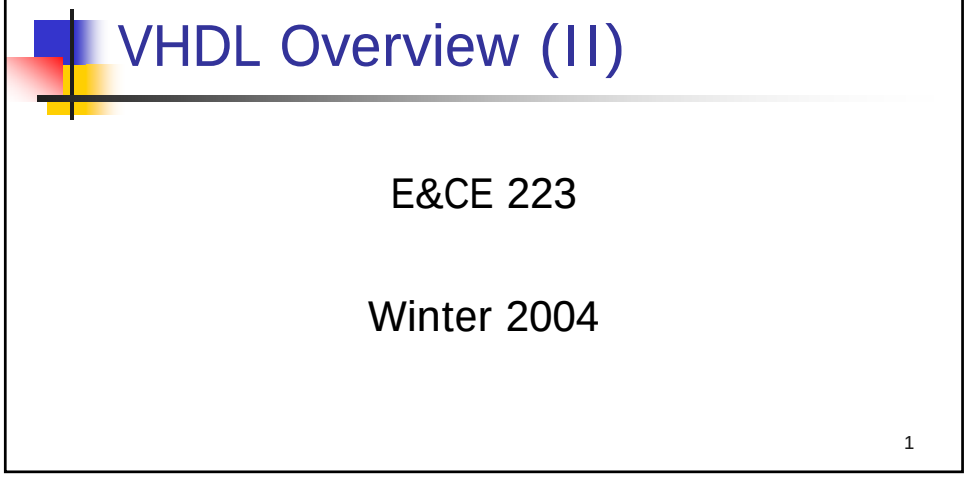
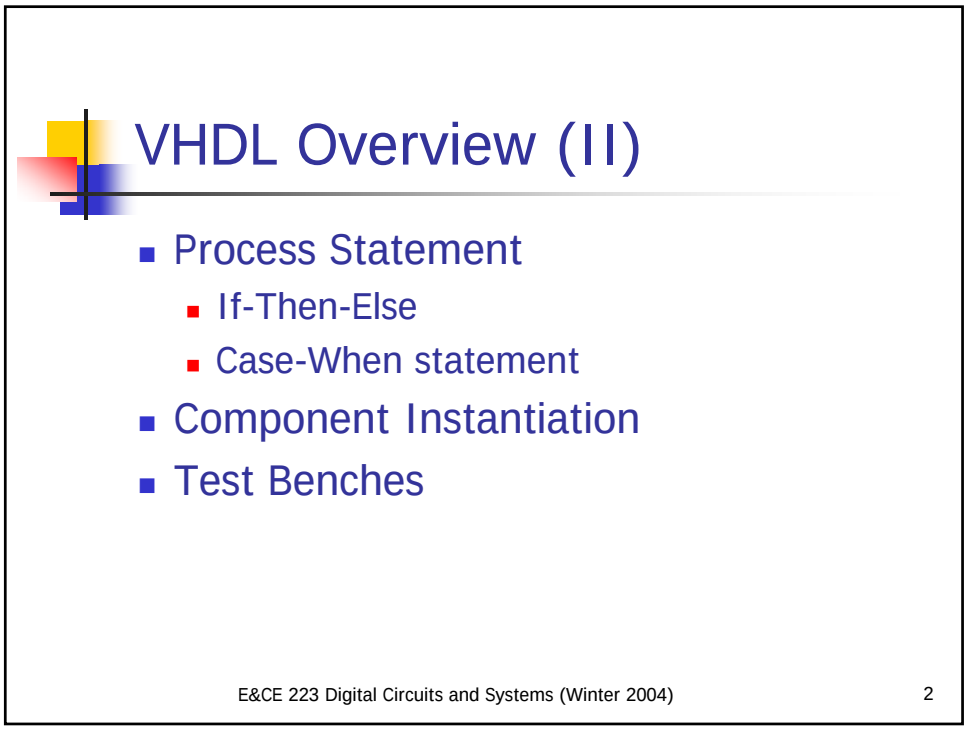
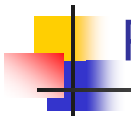

VHDL Overview (II)
E&CE 223
Winter 2004
1


VHDL Overview (II)
Process Statement
If-Then-Else
Case-When statement
Component Instantiation
Test Benches
E&CE 223 Digital Circuits and Systems (Winter 2004)
2



Concurrent Assignments Reviewed

- Recall that we have only considered concurrent assignments to this point
- All statements correspond to hardware and are occurring at the same time
  - Therefore, the order in which they appear is not important
- We need to introduce the concept of sequencing and control
- To accomplish this, we will introduce the **process statement**

Introduction to Process Statements

- In VHDL, a process is effectively a "block" of logic and control
  - **Each process operates in parallel with other processes and in parallel with other concurrent VHDL statements**
- **Inside** of a process, statements are **executed sequentially**
  - This will allow us to introduce sequencing and control
- **Inside** of a process, we can use additional VHDL syntax:
  - **If-Then-Else** statements
  - **Case** statements
- **Note: Operations with sequencing (like Case-When and If-Then-Else) must, and can only, be used inside of a process**

Illustration of a VHDL Process (Using an If-Then-Else)

- We can introduce the concept of a VHDL process by coding a 2-to-1 multiplexer (we have seen several ways to do a multiplexer):

```
library ieee;
use ieee.std_logic_1164.all;

entity multiplexer_2to1 is
port (x0,x1  : in  std_logic;       -- multiplexer inputs
      s      : in  std_logic;       -- multiplexer select line
      f      : out std_logic        -- multiplexer output
     );
end multiplexer_2to1;

architecture prototype of multiplexer_2to1 is
begin
  process (x0,x1,s)                -- process has a sensitivity list.
  begin
    if (s = '0') then              -- can use if-then-else inside of a process
      f <= x0;
    else
      f <= x1;
    end if;
  end process;
end prototype;
```


Breakdown of a Process

- The syntax of a process statement is:

```
[process_name] :    PROCESS sensitivity_list
                    BEGIN
                              -- statements.
                    END process;
```

- A process can be **named**; the naming of a process is **optional**
- The statements inside of the **BEGIN ... END** are evaluated **sequentially**
- A process has a **sensitivity list**
  - The execution of the statements inside of the process happens when the value of any signal in the sensitivity list changes
  - So, the signals that should be listed in the sensitivity list are those that can cause outputs to change inside of the process

More Comments on a Process

- Signals **changed inside** of the process are not changed until the process has completed evaluating every statement inside of the **BEGIN ... END**
- There is no **delay** associated with the process itself; i.e., a process is a programming construct and is assumed to execute in 0 time
  - Of course, statements inside of the process can have delays associated with them

Example Using If-Then-Elsif-...-Else (4-to-1 Multiplexer)

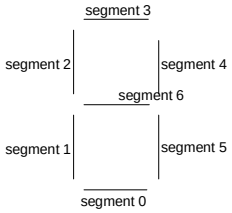
- We can write a 4-to-1 multiplexer using a VHDL process to illustrate and if-then-else construct with more than two choices:

```
library ieee;
use ieee.std_logic_1164.all;

entity multiplexer_4to1 is
port (x0,x1,x2,x3  : in  std_logic;                              -- multiplexer inputs
      s            : in  std_logic_vector(1 downto 0);           -- select lines
      f            : out std_logic);                 -- multiplexer outputs
end multiplexer_4to1;

architecture prototype of multiplexer_4to1 is
begin
  process (x0,x1,x2,x3,s)
  begin
    if    (s = "00") then
      f <= x0;
    elsif (s = "01") then  -- notice the syntax “elsif” without spaces
      f <= x1;
    elsif (s = "10") then
      f <= x2;
    else
      f <= x3;
    end if;
  end process;
end prototype;
```

Example Using Case Statement (BCD Decoder)

- Consider a circuit that receives a code for a decimal digit 0...9 encoded using 4-bits

inputs				decimal	outputs						
c_0	c_1	c_2	c_3	value	s_0	s_1	s_2	s_3	s_4	s_5	s_6
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	0	0	0	1	1	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	0	0	0	1	1	1	1
0	1	0	0	4	0	0	1	0	1	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	1	1	1	1	0	1	1
0	1	1	1	7	0	0	0	1	1	1	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	0	0	1	1	1	1	1



- We want to decide how to drive the segments of a 7-segment display in order to illuminate the decimal digit we receive at the circuit input.

Example Using Case-When Statement Continued

- We can write a VHDL Description using a process and a case statement:

```
library ieee;
use ieee.std_logic_1164.all;

entity bcd_decoder is
port (code  : in  std_logic_vector(3 downto 0);        -- coded inputs
      leds  : out std_logic_vector(6 downto 0)); -- signal outputs
end bcd_decoder;

architecture prototype of bcd_decoder is
begin
 process (code)
  begin
    case code is
      when "0000"  => leds <= "0111111" ;
      when "0001"  => leds <= "0110000" ;
      when "0010"  => leds <= "1011011" ;
      when "0011"  => leds <= "1111000" ;
      when "0100"  => leds <= "1110100" ;
      when "0101"  => leds <= "1101101" ;
      when "0110"  => leds <= "1101111" ;
      when "0111"  => leds <= "0111000" ;
      when "1000"  => leds <= "1111111" ;
      when "1001"  => leds <= "1111100" ;
      when others  => leds <= null ;    -- the default case.
    end case;
  end process;
end prototype;
```

Case Statement Comments

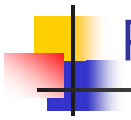
- Notice that our process still has a **sensitivity list**
- The syntax of the case statement is:

```
CASE signal_name IS
        when (condition1) = > (signal_assignment1) ;
        when (condition2) = > (signal_assignment2) ;
        ...
        when others = > (default_assignment);
ENS CASE;
```

- Note: Our case statement has a default for situations where no condition matches (we use the “others” keyword)
  - In the default assignment we can use the “null” keyword which is much like saying “doesn’t matter, so do nothing/whatever”

Final Comments on Process Statements

- In addition to statements like case statements and if-then-else statements, the **concurrent signal assignment**
  - i.e., **lhs < = rhs ;** -- works inside a process (we used it!)
- This is to keep concurrent and sequential statements **separated**



Process (Summary)

- Wrapper for “sequential” statements
  - (*sensitivity list*)
    - Tells simulator to re-simulate the process when any member of the list changes value
  - Sequential-type statements can *only* appear inside a process:
    <= If-Then-Elsif-Else Case-When
  - Concurrent-type statements can appear anywhere
- Processes together become concurrent blocks of logic

VHDL Overview (II)

- Process Statement
  - If-Then-Else
  - Case-When statement
- Component Instantiation
- Test Benches

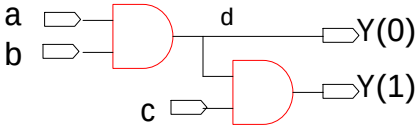
Component Instantiation

- Often we will have a VHDL Description of part of a circuit that we would like to use inside of another circuit
- An example of this is a 3-input AND gate:
  - We might have the VHDL Description of a 2-input AND gate
  - We might want to build a 3-input AND gate using a 2-input AND gate
- We can use VHDL Descriptions inside of other VHDL Descriptions by:
  - **Declaring components** we wish to use
  - **Creating** or **instantiating** copies of the **components**
  - **Connecting** together the **components**

Our First Example (AND Gate)

- A simple AND2 gate description:

```
-- VHDL for AND2 Gate.

library ieee;
use ieee.std_logic_1164.all;

entity AND2 is
    port( a,b : in  std_logic;
          z   : out std_logic);
end AND2;

architecture prototype of AND2 is
begin
        z   <= a and b;  -- output
end prototype;
```



- We are required to build a 3-input logic component using two 2-input AND gates

VHDL for a 3-input Logic component – Component Instantiation

- Before we can use the AND2 gate, we need to declare it inside of the declarative section of the architecture:

```
-- VHDL for Logic_component gate

library ieee;
use ieee.std_logic_1164.all;

entity logic_block is
  port (      a: in std_logic;
              b: in std_logic;
              c: in std_logic;
              y: out std_logic_vector(1 downto 0));
end logic_block;

architecture prototype of logic_block is
  -- need to declare the components (other VHDL descriptions) we will use.
  -- like the entity of the circuit we want to use, but notice the
  -- "component" keyword
  component AND2
    port (a,b     : in   std_logic;
          z       : out  std_logic);
  end component;

  signal d: std_logic;

begin
  -- description will go here.
end prototype;
```

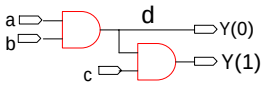
Component declaration

Temporary signal

VHDL for Logic_component – Instantiation of Components.

- Once we have declared components, we can instantiate copies and connect them together:

```
-- VHDL for the logic function.

-- continued from previous page (architecture body only).

architecture prototype of Logic_component is
  component AND2
    port (a,b     : in   std_logic;
          z       : out  std_logic);
  end AND2;

  signal d: std_logic;

Begin
 Y(0) <= d;
 AND_GATE1: AND2
       port map (a=>a, b=>b, z=>d);
 AND_GATE2: AND2
       port map (a=>c, b=>d, z=>y(1));
end prototype;
```



component declaration

component instantiations and wiring

Component Instantiation Summary

- Once we declare a component (another VHDL Description), we can create a copy of it using the syntax:

```
instance_name : component_name
   PORT MAP (signal_in_component => signal_in_current_circuit,
             signal_in_component => signal_in_current_circuit,
                      ...
             signal_in_component => signal_in_current_circuit
            );
```

- The notation "=>" inside of the "port map" is like connecting or wiring the signal in the current circuit to an input or output pin of the sub-circuit (the component)

Additional Notes

- Note that we needed temporary signals in order to connect the one component to another:

```
architecture prototype of Logic_component is
  -- component declarations…

  signal d : std_logic; -- temporary signals needed in the architecture.

begin
  -- cicuit description…
end prototype;
```

- The syntax for this is:

signal signal_name : type ;

VHDL Overview (II)

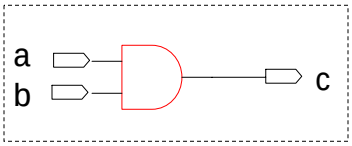
- Process Statement
  - If-Then-Else
  - Case-When statement
- Component Instantiation
- Test Benches

Test bench

- Verifies the functionality of a module
- Uses component instantiation of the module
- Applies *test vectors* to inputs
- Output can be verified

Test bench for AND2 gate

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY and_bench IS
END and_bench;

ARCHITECTURE testbench_arch OF and_bench IS

            COMPONENT AND2
                        PORT (
                                    a : In  std_logic;
                                    b : In  std_logic;
                                    c : Out  std_logic
                        );
            END COMPONENT;

            SIGNAL a : std_logic;
            SIGNAL b : std_logic;
            SIGNAL c : std_logic;

BEGIN
            UUT : AND2
            PORT MAP (
                        a => a,
                        b => b,
                        c => c
            );
```



Test bench for AND2 gate (cont.)

```
            PROCESS
            BEGIN
                        -- ---------------------
                        a <= '1';
                        b <= '0';
                        -- ---------------------
                        WAIT FOR 10 ns; -- Time=10 ns
                        a <= '0';
                        b <= '1';
                        -- ---------------------
                        WAIT FOR 10 ns; -- Time=20 ns
                        a <= '0';
                        b <= '0';
                        -- ---------------------
                        WAIT FOR 10 ns; -- Time=30 ns
                        a <= '1';
                        b <= '1';
                        -- ---------------------
                        WAIT FOR 10 ns; -- Time=40 ns
            END PROCESS;
END testbench_arch;

CONFIGURATION and2_cfg OF and_bench IS
            FOR testbench_arch
            END FOR;
END and2_cfg;
```

Test Vectors

Configuration section