
The Process Statement

Module 4

Overview

- Process
- Sequential Statements
 - IF statements
 - CASE statements
 - LOOP statement
 - Variable Assignment
 - Signal Assignment

Architecture - Review

- An Architecture describes the functionality of an Entity
- Consists of **concurrent** statements, e.g.
 - Process Statement
 - Concurrent Signal Assignments
 - Conditional Signal Assignments
- Concurrent statements - order *does not* matter

```
-- example of concurrent signal assignments
ARCHITECTURE arch OF full_adder IS
BEGIN
    sum <= a XOR b XOR c;
    temp <= a AND b;
    cout <= temp AND c;
END arch;
```

Process Statement

- A process is the fundamental building block of architecture bodies.
 - A Process statement is an example of a *concurrent* statement. It is composed of a set of *sequential* statements - executes in sequence

```
[process_label:]  
PROCESS [(sensitivity_list)]  
    [process declarations]  
BEGIN  
    sequential statements  
END PROCESS [process_label];
```

- Sensitivity List shows which signals the process is *sensitive* to
 - any change on these signals causes process to be executed
 - suspends after executing last sequential statement - waits for another event to occur on signal in sensitivity list

IF statement

- Example of a sequential statement - only used in process or subprogram
- Selects a sequence of statements for execution based on the value of a condition
- Each condition is checked sequentially until the first condition is true - priority is implied

```
IF opcode = add_op THEN
    result := abus + b_bus;
ELSIF opcode = inc_op THEN
    IF flag = false THEN          -- nested IF statement
        result := 0;
    ELSE
        result := result + 1;
    END IF;
END IF;
```

CASE statement

- Selects one of a number of branches based on the value of an expression
 - Expression must be integer, enumerated type or one-dimensional character array (like `bit_vector` or `std_logic_vector`)
- All possible values must be covered exactly once

```
CASE state IS -- state is an enumerated type in this example
  WHEN s0 =>      -- branch 1
    counter := 0;
    state <= s2;
  WHEN s1 =>      -- branch 2
    state <= s4;
  WHEN s2 | s3 => -- can use set of choices (not logical OR)
    counter := counter + 1;
    state <= s0;
  WHEN OTHERS => -- can use OTHERS to cover remaining values
    state <= s3;
END CASE;
```

LOOP STATEMENTS

- Used to iterate through a set of sequential statements
- Three types

```
FOR identifier IN range LOOP
```

```
END LOOP;
```

```
WHILE boolean_expression LOOP
```

```
END LOOP;
```

```
LOOP
```

```
    EXIT WHEN condition_test  
END LOOP;
```

Process example - BCD_COUNT

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;  -- required for addition

ENTITY bcd_count IS          -- bcd counter
    PORT (clk, reset: IN std_logic;
          q : OUT std_logic_vector (3 DOWNTO 0));
END bcd_count;
```

BCD_COUNT (cont'd)

```
ARCHITECTURE behav OF bcd_count IS
    SIGNAL temp : std_logic_vector (3 DOWNTO 0);
BEGIN    -- two concurrent statements (Process and signal assignment)
    PROCESS (clk, reset)    -- sensitivity list for process
    BEGIN
        IF reset = '1' THEN
            temp <= "0000";
        ELSIF clk'EVENT AND clk = '1' THEN
            IF temp = "1001" THEN    -- check if '9'
                temp <= "0000";    -- back to '0'
            ELSE
                temp <= temp + 1;    -- increment by one
            END IF;
        END IF;
    END PROCESS;
    q <= temp;
END behav;
```

Synthesis Results

The screenshot displays the Xilinx Project Navigator interface. The top window shows the source tree with the project files. The middle window shows the list of processes for the source file, with 'View Technology Schematic' selected. The right window shows the VHDL source code for the `bcd_count` entity, which is a 4-bit up counter. The bottom window shows the synthesis results, including the HDL Synthesis summary and the Advanced HDL Synthesis section.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity bcd_count is
7     Port ( clk : in std_logic;
8           reset : in std_logic;
9           q : out std_logic_vector(3 downto 0));
10 end bcd_count;
11
12 architecture Behavioral of bcd_count is
13     SIGNAL temp : std_logic_vector (3 DOWNTO 0);
14 begin
15     -- two concurrent statements (Process and signal assignment)
16     PROCESS (clk, reset) -- sensitivity list for process
17     BEGIN
18         IF reset = '1' THEN
19             temp <= "0000";
20         ELSIF clk'EVENT AND clk = '1' THEN
21             IF temp = 9 THEN -- check if '9'
22                 temp <= "0000"; -- back to '0'
23             ELSE
24                 temp <= temp + "0001"; -- increment by one
25             END IF;
26         END IF;
27         -- q <= temp;
28     END PROCESS;
29     q <= temp;
30
31 end Behavioral;
```

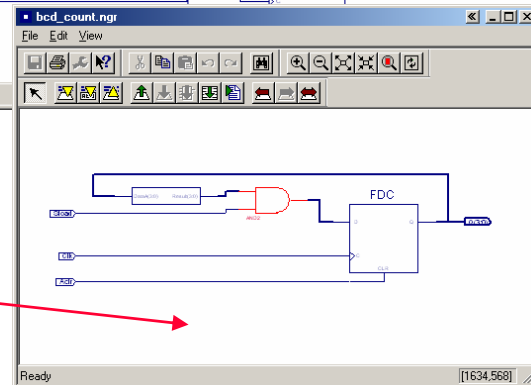
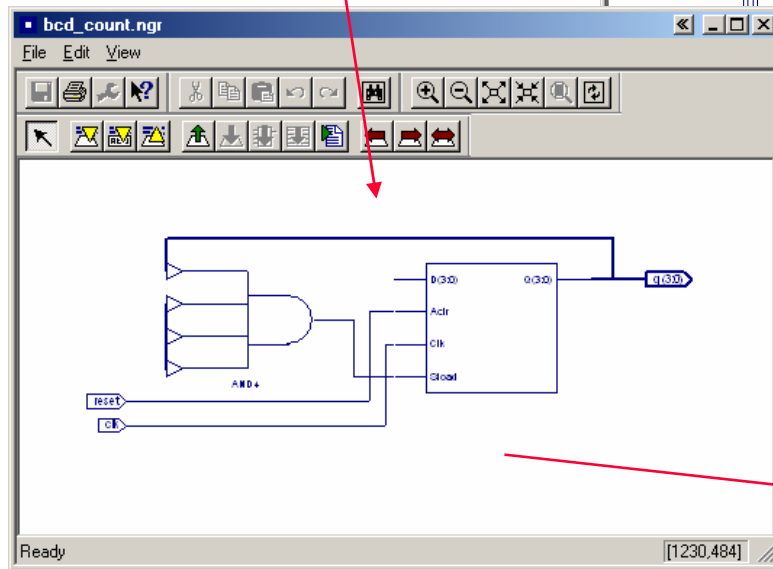
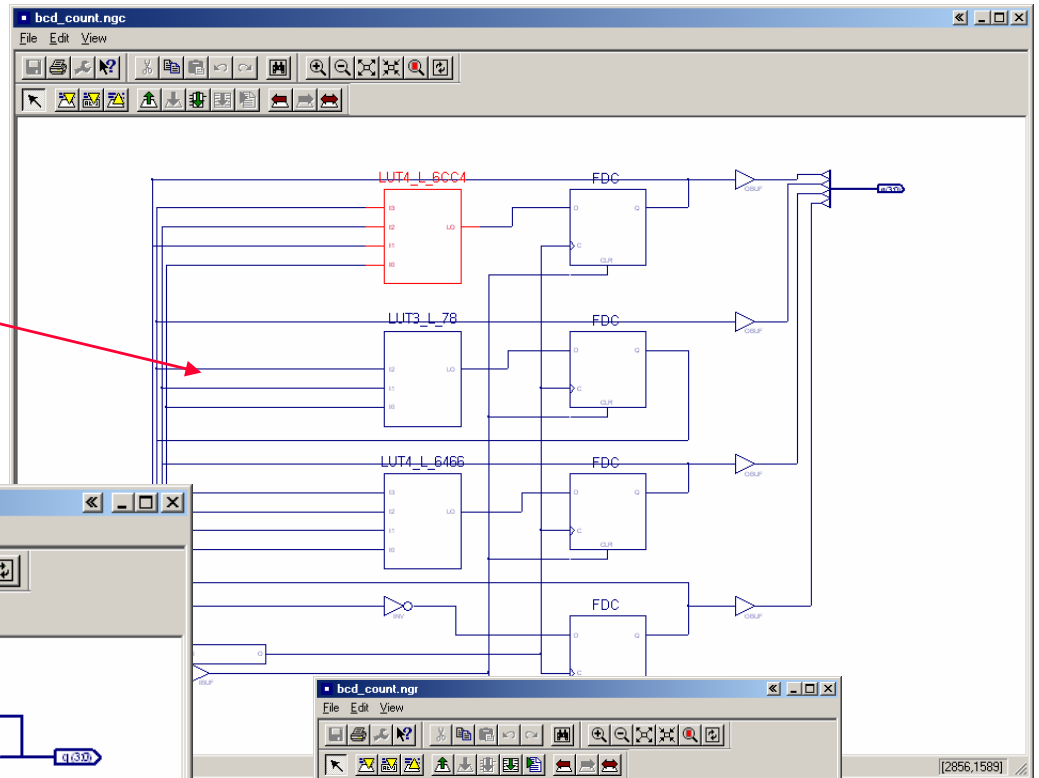
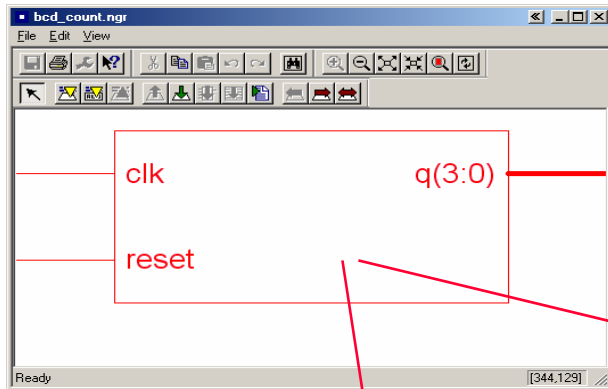
* HDL Synthesis *

Synthesizing Unit <bcd_count>.
Related source file is "C:/ee574/bcd_count/bcd_count.vhd".
Found 4-bit up counter for signal <temp>.
Summary:
inferred 1 Counter(s).
Unit <bcd_count> synthesized.

* Advanced HDL Synthesis *

Advanced RAM inference ...
Advanced multiplier inference ...

RTL and Technology Schematic



Variable Assignment

- Assigns a new value (immediately) to a variable
`a := b + 45; -- a and b are integers`
- Variables only used in process or subprograms
 - Declared inside process
 - Not accessible outside of process
 - Need to assign to signal for access outside process
- Used for temporary storage

Example

```
ENTITY example2 IS      --
    PORT(a_bus : IN integer RANGE 0 TO 127;
          flag   : OUT std_logic);
END example2;

ARCHITECTURE arch OF example2 IS
BEGIN
    PROCESS(a_bus)
        VARIABLE j : integer RANGE 0 to 127; --only visible in process
    BEGIN
        j := a_bus / 2;
        j := j + 4;           -- update immediately
        IF j > 50 THEN
            flag <= '1';
        ELSE
            flag <= '0';
        END IF;
    END PROCESS;
END arch;
```

Signal Assignments inside a Process

- Changes the value of a signal (wire or net)
- (If outside a process then it is a *concurrent* statement)
- If inside a process then executed sequentially with other statements.
- A signal assignment will supersede a previous assignment to the same signal
- **A signal update occurs after a *delta delay* (very small delay) - allows for ordering of events.**
- Very important difference:
 - variables updated immediately
 - signals get new values at a later time (usually end of process or when simulation time advances)

Example

```
ARCHITECTURE behav OF incorrect_example IS
    SIGNAL a : integer;      -- declare internal signal
BEGIN
    PROCESS(f)
        VARIABLE j, k : integer;
    BEGIN
        j := f + k;  -- j updated immediately
        a <= j + 1;  -- a updated at end of process
        k := a;      -- k gets old value of a
    END PROCESS
END behav;
```

Concurrent Signals - Reminder

```
ENTITY test3 IS
    PORT (a , clk      : IN std_logic;
          d            : OUT std_logic);
END test3;

ARCHITECTURE arch OF test3 IS
    SIGNAL b, c        : std_logic;
BEGIN
    b <= a; -- order does not matter
    d <= c;
    c <= b;
END arch;
```

Synthesis Results

The screenshot displays the Xilinx Project Navigator interface. The top window shows the project structure with 'test3-behavioral (test3.vhd)' selected. The middle window shows the 'Processes for Source' list, with 'View RTL Schematic' highlighted. The right window shows the VHDL code for 'test3.vhd':

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity test3 is
7      Port ( a : in std_logic;
8            clk : in std_logic;
9            d : out std_logic);
10 end test3;
11
12 architecture Behavioral of test3 is
13     SIGNAL b, c : std_logic;
14 begin
15     b <= a;
16     d <= c;
17     c <= b;
18
19 end Behavioral;
20
```

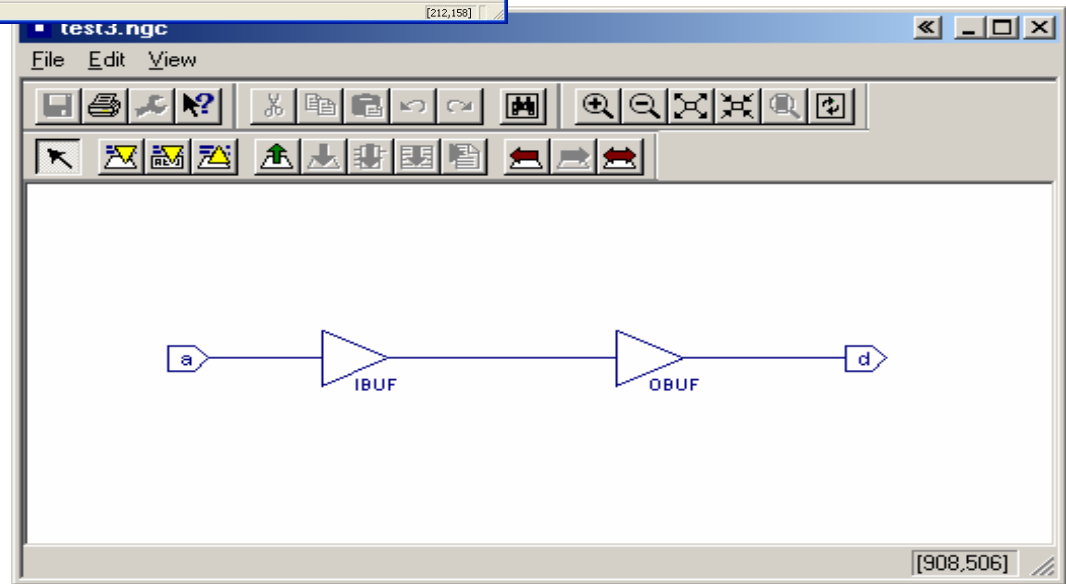
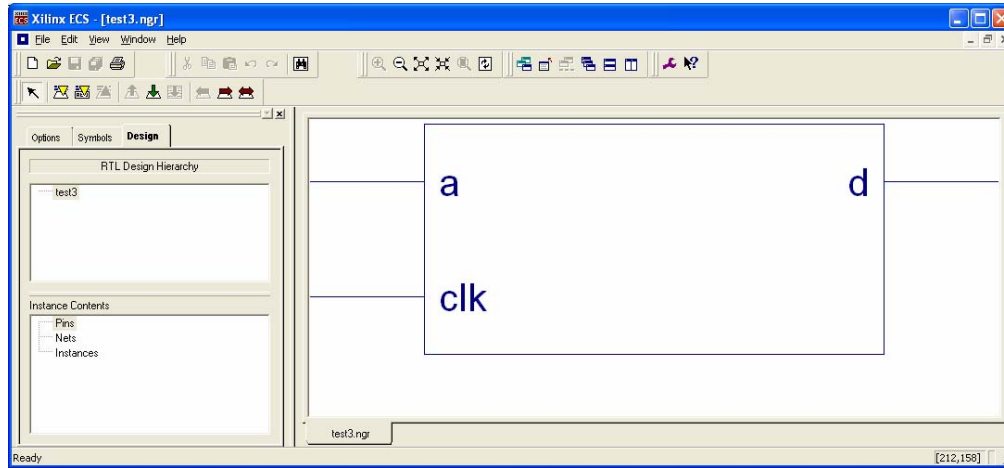
The bottom window shows the synthesis results in the console:

```
-----
*                               HDL Synthesis                               *
-----

Synthesizing Unit <test3>.
  Related source file is c:/ee574/test3/test3.vhd.
[warn] WARNING:Xst:647 - Input <clk> is never used.
Unit <test3> synthesized.
```

The status bar at the bottom indicates 'Process "View RTL Schematic" is up to date.' and 'Ln 6 Col 1'.

RTL Schematic



Signals in Clocked Process (flip-flops)

```
ENTITY test3a IS
    PORT (a , clk : IN std_logic;
          d       : OUT std_logic);
END test3a;

ARCHITECTURE arch OF test3a IS
    SIGNAL b, c : std_logic;
BEGIN
    PROCESS (clk)
    BEGIN
        IF clk'EVENT AND clk = '1' THEN
            b <= a;
            c <= b;
            d <= c;
        END IF;
    END PROCESS; -- b, c, d updated with previous values of a, b, c
END arch;
```

Synthesis Results

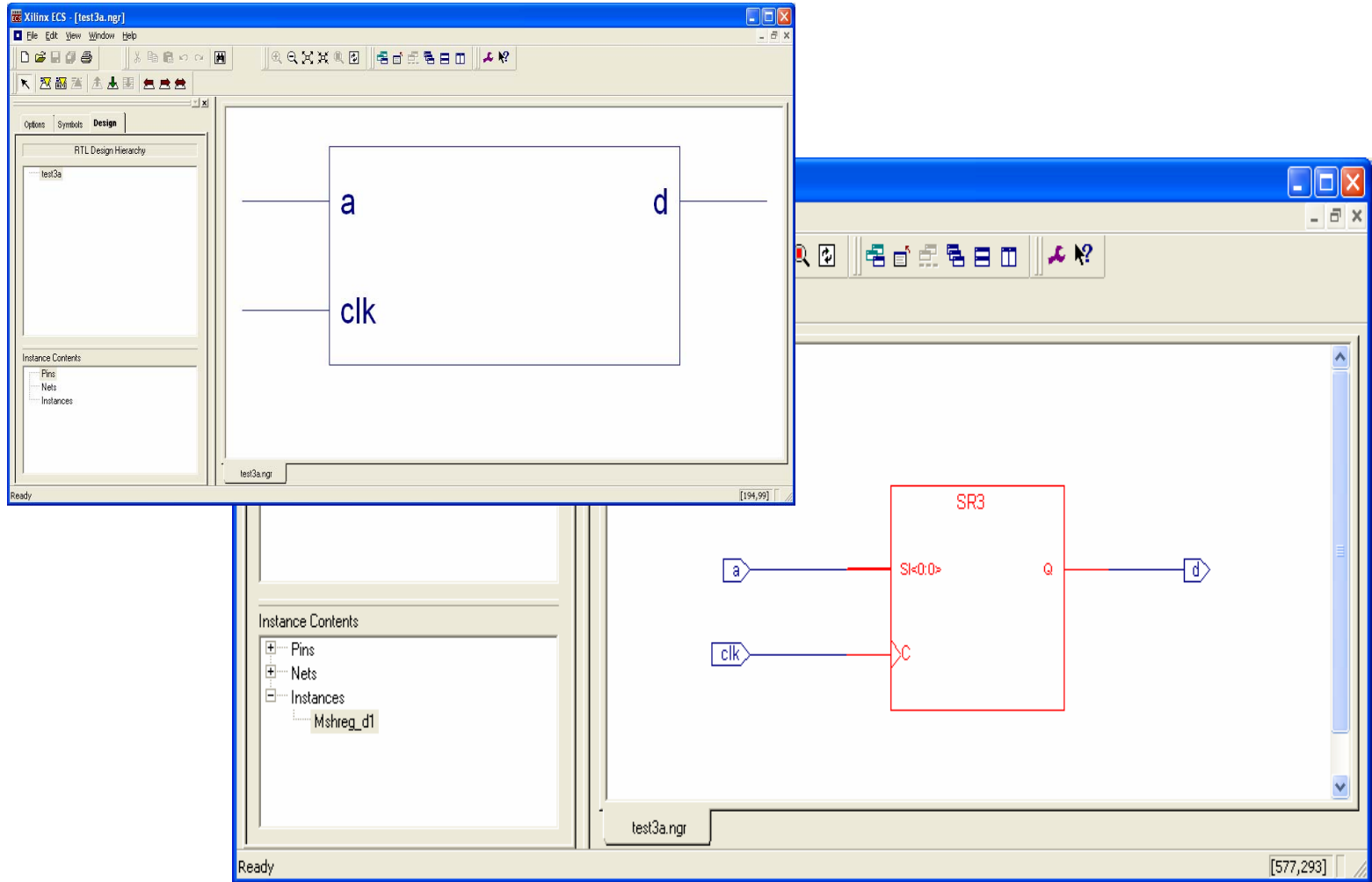
The screenshot displays the Xilinx Project Navigator interface. The top window shows the project structure with 'test3a-behavioral (test3a.vhd)' selected. The middle window shows the 'Processes for Source' list, with 'Synthesize ->XST' highlighted. The right window shows the VHDL code for 'test3a.vhd':

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7 entity test3a is
8     Port ( a : in std_logic;
9           clk : in std_logic;
10          d : out std_logic);
11 end test3a;
12
13 architecture Behavioral of test3a is
14     SIGNAL b, c : std_logic;
15 begin
16     PROCESS (clk)
17     BEGIN
18         IF clk'EVENT AND clk = '1' THEN
19             b <= a;
20             c <= b;
21             d <= c;
22         END IF;
23     END PROCESS; -- b, c, d updated with previous values of a, b, c
24 end Behavioral;
```

The bottom window shows the synthesis results in the console:

```
=====  
Synthesizing Unit <test3a>.  
  Related source file is c:/ee574/test3a/test3a.vhd.  
  Found 3-bit shift register for signal <d>.  
  Summary:  
    inferred 1 Shift register(s).  
  Unit <test3a> synthesized.  
=====  
*           Advanced HDL Synthesis           *  
=====  
Advanced RAM inference ...  
Advanced multiplier inference ...  
Advanced Registered AddSub inference ...  
Dynamic shift register inference ...  
=====  
HDL Synthesis Report  
  
Macro Statistics  
# Shift Registers           : 1  
3-bit shift register       : 1  
=====  
Process "View RTL Schematic" is up to date.
```

RTL Schematic



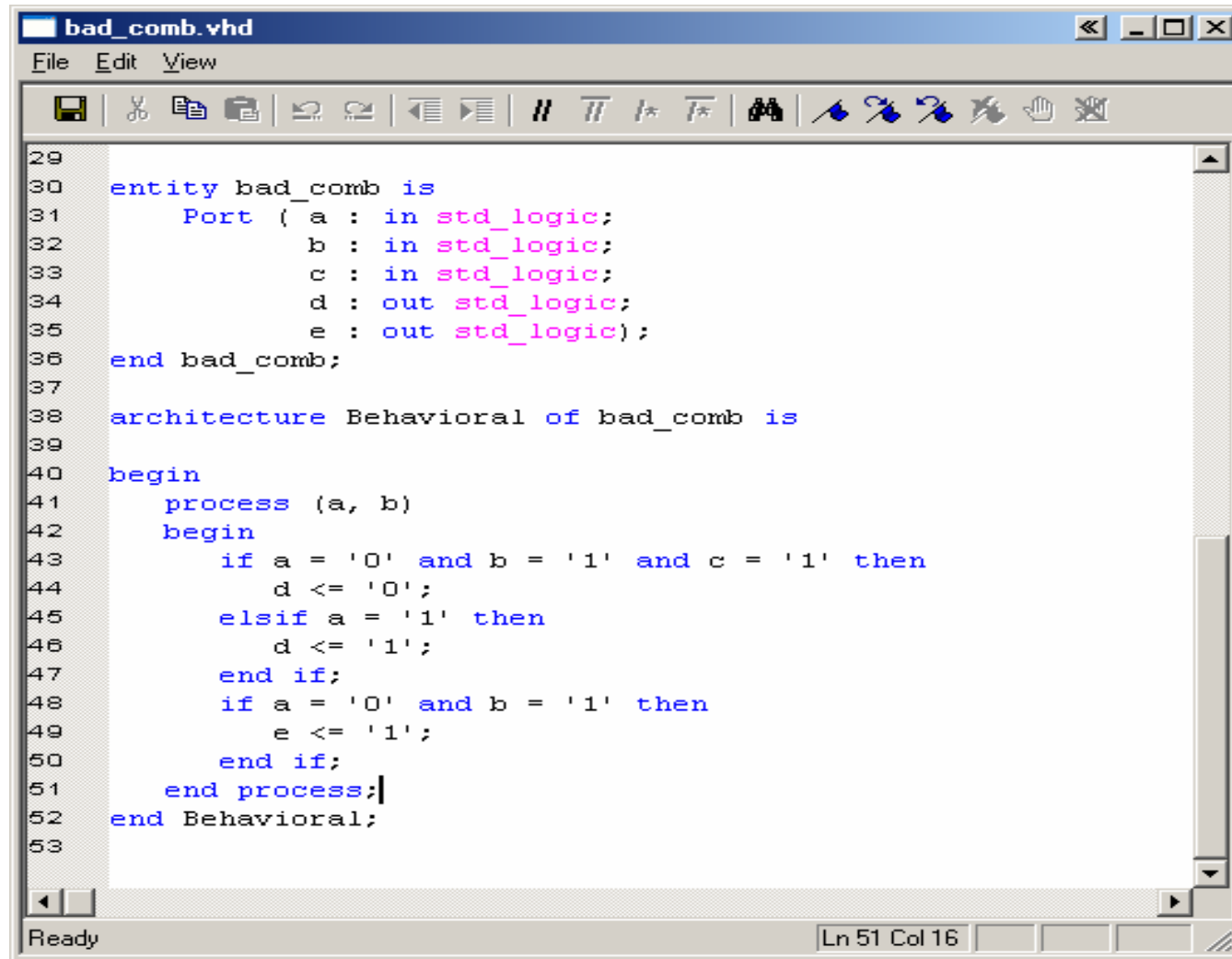
Process Overview

- A process may be used to describe combinational or sequential (clocked) logic.
 - Combinational Logic
 - in combinational logic the outputs are only dependent on the inputs
 - no latches or flip-flops should be generated
 - Sequential Logic
 - contains memory elements (storage) - outputs dependent on both current inputs and past events
 - see next module for examples

Process Style for Combinational Logic

- General Rules
 - sensitivity list is required and MUST include all signals used in process
 - Synthesis tools will only provide warning, simulation will fail
 - variables must NOT be used before being set
 - last successive assignment to a signal is last one implemented
 - all outputs should have default values
 - if not, a latch will be generated to hold the current value
 - No WAIT statements allowed in process

Incorrect Combinational Process



```
bad_comb.vhd
File Edit View
[Icons]
29
30 entity bad_comb is
31     Port ( a : in std_logic;
32           b : in std_logic;
33           c : in std_logic;
34           d : out std_logic;
35           e : out std_logic);
36 end bad_comb;
37
38 architecture Behavioral of bad_comb is
39
40 begin
41     process (a, b)
42     begin
43         if a = '0' and b = '1' and c = '1' then
44             d <= '0';
45         elsif a = '1' then
46             d <= '1';
47         end if;
48         if a = '0' and b = '1' then
49             e <= '1';
50         end if;
51     end process;
52 end Behavioral;
53
Ready Ln 51 Col 16
```

Latch and constant value generated

The screenshot displays the Xilinx Project Navigator interface. The top window shows the HDL code for an entity named `bad_comb`. The code defines a process with a sensitivity list containing `a`, `b`, and `c`. Inside the process, there is an `if` statement that branches based on the value of `a`. In one branch, a signal `d` is assigned a constant value `0`. In another branch, a signal `e` is assigned the value of `b`. The code ends with `end Behavior`.

The middle window shows a logic schematic for the synthesized design. It features a LUT3_40 block and an LDC (LUT-to-DRAM Converter) block. The LUT3_40 block has three inputs (`a`, `b`, `c`) and one output (`d`). The LDC block has two inputs (`d`, `e`) and one output (`f`). The schematic shows the LUT3_40 block connected to the LDC block, and the LDC block connected to an output buffer.

The bottom window shows the synthesis report, which includes the following warnings:

```
***** HDL Analysis *****
Analyzing Entity <bad_comb> (Architecture <Behavioral>).
WARNING: Xst:819 - "C:/ee574/bad_comb/bad_comb.vhd" line 41: The following signals are missing in the process sensitivity list:
c
INFO: Xst:1304 - Contents of register <e> in unit <bad_comb> never changes during circuit operation. The register is replaced by logic.
Entity <bad_comb> analyzed. Unit <bad_comb> generated.
***** HDL Synthesis *****
Synthesizing Unit <bad_comb>.
Related source file is "C:/ee574/bad_comb/bad_comb.vhd".
WARNING: Xst:737 - Found 1-bit latch for signal <d>.
Unit <bad_comb> synthesized.
***** Advanced HDL Synthesis *****
```

Incorrect Process for Combinational Logic

```
ENTITY comparator IS      -- 4-bit magnitude comparator
    PORT(a,b : IN std_logic_vector(3 DOWNT0 0);
          equal, less, great      : OUT std_logic);
END comparator;

ARCHITECTURE incorrect OF comparator IS
BEGIN
    PROCESS(a, b)
    BEGIN
        IF a = b THEN
            equal <= '1';
        ELSIF a <= b THEN
            less <= '1';
        ELSIF a >= b THEN
            great <= '1';
        END IF;
    END PROCESS;
END incorrect;
```

Correct Process for Combinational Logic

```
ENTITY comparator IS      -- 4-bit magnitude comparator
    PORT(a,b : IN std_logic_vector(3 DOWNTO 0);
          equal, less, great      : OUT std_logic);
END comparator;

ARCHITECTURE correct OF comparator IS
BEGIN
    PROCESS(a, b)
    BEGIN
        equal <= '0';
        less <= '0';
        great <= '0';
        IF a = b THEN
            equal <= '1';
        ELSIF a <= b THEN
            less <= '1';
        ELSIF a >= b THEN
            great <= '1';
        END IF;
    END PROCESS;
END correct;
```

Alternative Combinational Process

```
ARCHITECTURE correct2 OF comparator IS -- same as previous ?
BEGIN
    PROCESS(a, b)
    BEGIN
        IF a = b THEN
            equal <= '1';
        ELSE
            equal <= '0';
        END IF;
        IF a <= b THEN
            less <= '1';
        ELSE
            less <= '0';
        END IF;
        IF a >= b THEN
            great <= '1';
        ELSE
            great <= '0';
        END IF;
    END PROCESS;
END correct2;
```

Concurrent Statements (preferred)

```
ARCHITECTURE preferred OF comparator IS
BEGIN
    equal <= '1' WHEN a = b ELSE
        '0';
    less <= '1' WHEN a <= b ELSE
        '0';
    great <= '1' WHEN a >= b ELSE
        '0';
END preferred;
```