The Process Statement

Module 4
Overview

• Process
• Sequential Statements
  – IF statements
  – CASE statements
  – LOOP statement
  – Variable Assignment
  – Signal Assignment
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

```vhdl
-- 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

```plaintext
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;
```

Jim Duckworth, WPI 5 The Process Statement - Module 4
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

```vhdl
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;
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;
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

Jim Duckworth, WPI
Variable Assignment

- Assigns a new value (immediately) to a variable
  
  \[ a := b + 45; \quad -- \text{a and b are integers} \]

- 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
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;
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
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

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity test1a is
  Port ( a : in STD_LOGIC;
        b : in STD_LOGIC;
        d : out STD_LOGIC);
end test1a;

architecture Behavioral of test1a 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;
end Behavioral;
```

Jim Duckworth, WPI 20 The Process Statement - Module 4
RTL Schematic
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

```vhdl
entity bad_comb is
  Port ( a : in std_logic;
        b : in std_logic;
        c : in std_logic;
        d : out std_logic;
        e : out std_logic);
end bad_comb;

architecture Behavioral of bad_comb is
begin
  process (a, b)
  begin
    if a = '0' and b = '1' and c = '1' then
      d <= '0';
    elsif a = '1' then
      d <= '1';
    end if;
    if a = '0' and b = '1' then
      e <= '1';
    end if;
  end process;
end Behavioral;
```
Latch and constant value generated
Incorrect 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 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;
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;