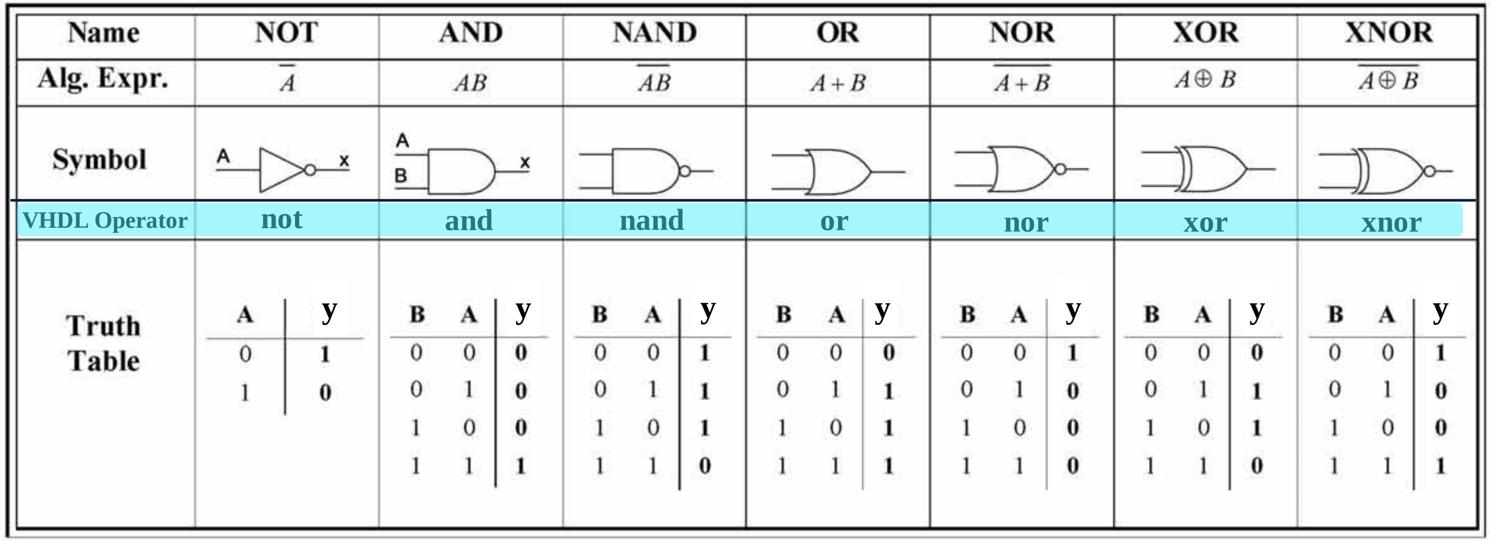
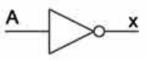
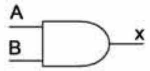
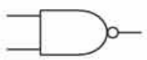
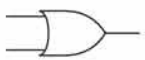
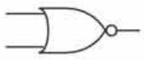
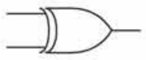
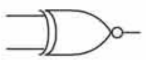
3- Combinational Circuits

- **Gate Level Combinational Circuits**
- **Concurrent vs. Sequential**
- **Concurrent Statements**
  - Signal Assignments
  - Component Instantiations
  - Process Statements
- **Sequential Statements**
  - IF Statement
  - CASE Statement
  - FOR LOOP Statement
- **Examples**

Basic Logic Gates

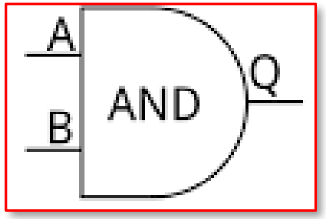
- Let’s start with describing basic gates in VHDL



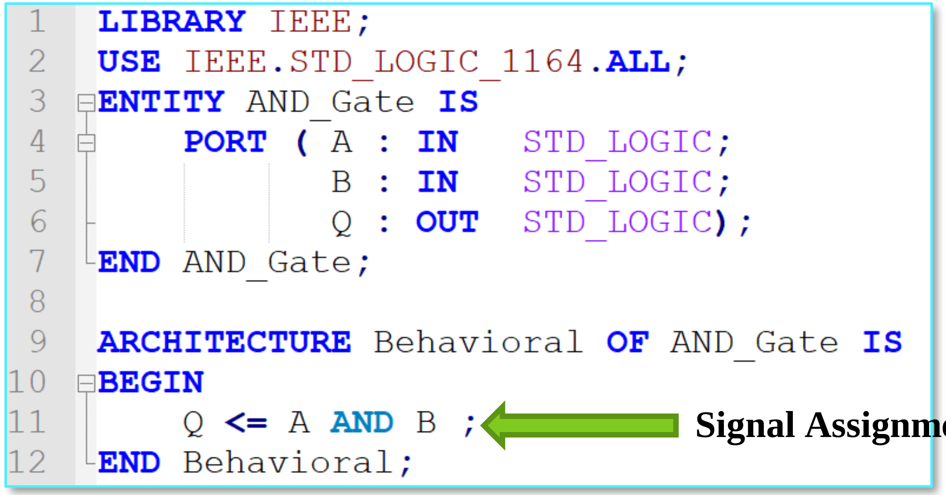
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
VHDL Operator	not	and	nand	or	nor	xor	xnor
Truth Table	A y 0 1 1 0	B A y 0 0 0 0 1 0 1 0 0 1 1 1	B A y 0 0 1 0 1 1 1 0 1 1 1 0	B A y 0 0 0 0 1 1 1 0 1 1 1 1	B A y 0 0 1 0 1 0 1 0 0 1 1 0	B A y 0 0 0 0 1 1 1 0 1 1 1 0	B A y 0 0 1 0 1 0 1 0 0 1 1 1

Gate Level Combinational Circuits

- We use *IEEE.STD_LOGIC_1164*
- Architecture uses a concurrent signal assignment



- AND Gate



```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AND_Gate IS
    PORT ( A : IN   STD_LOGIC;
           B : IN   STD_LOGIC;
           Q : OUT  STD_LOGIC);
END AND_Gate;

ARCHITECTURE Behavioral OF AND_Gate IS
BEGIN
    Q <= A AND B ;
END Behavioral;
```

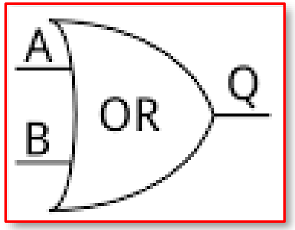
Signal Assignment

Gate Level Combinational Circuits

- Using OR operator

  - OR Gate

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY OR_Gate IS
    PORT ( A : IN   STD_LOGIC;
           B : IN   STD_LOGIC;
           Q : OUT  STD_LOGIC) ;
END OR_Gate;

ARCHITECTURE Behavioral OF OR_Gate IS
BEGIN
    Q <= A OR B ;
END Behavioral;
```

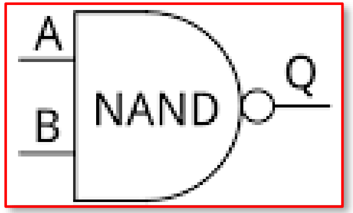


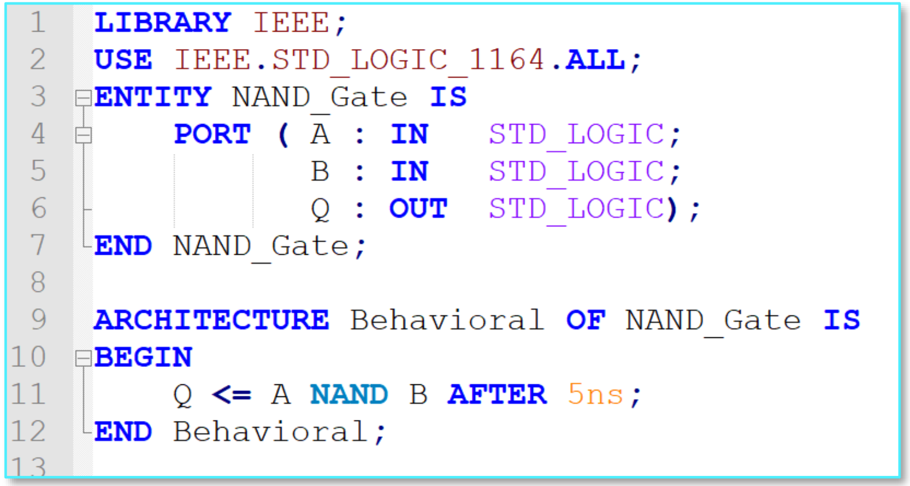
3- Combinational Circuits

Gate Level Combinational Circuits

- Like others the statement is sensitive to events on its right hand side
- For precision in modeling, use real time delay

  - NAND Gate



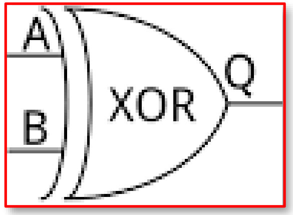


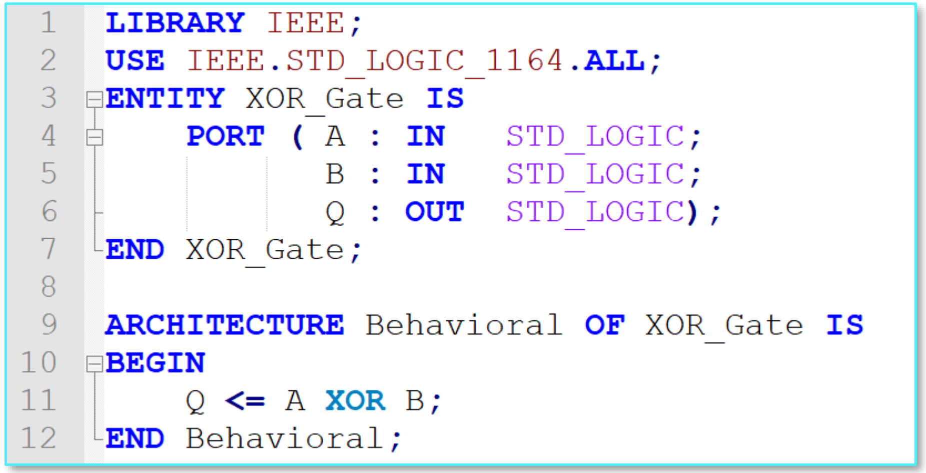
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY NAND_Gate IS
    PORT ( A : IN   STD_LOGIC;
           B : IN   STD_LOGIC;
           Q : OUT  STD_LOGIC);
END NAND_Gate;

ARCHITECTURE Behavioral OF NAND_Gate IS
BEGIN
    Q <= A NAND B AFTER 5ns;
END Behavioral;
```

Gate Level Combinational Circuits

- Using XOR operator
- Q gets A or B after a delta cycle

- XOR Gate



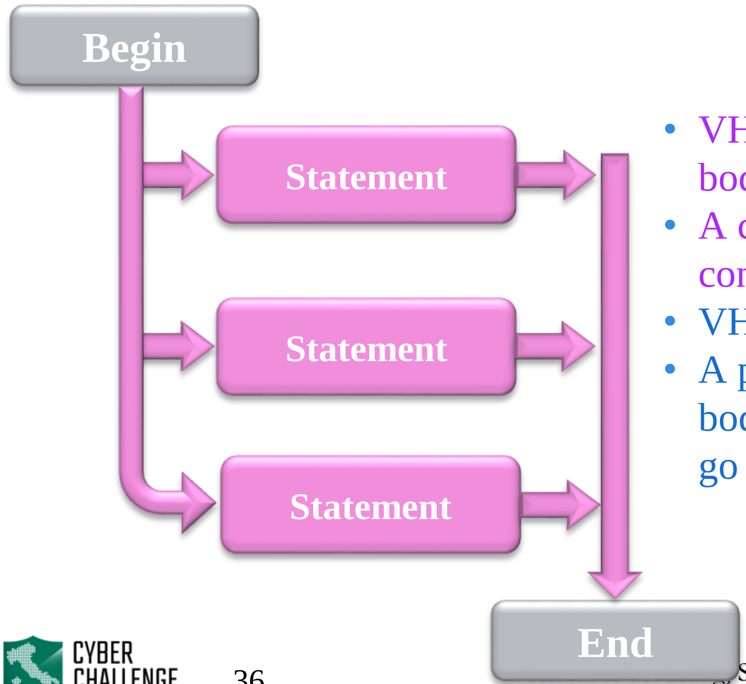


```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY XOR_Gate IS
    PORT ( A : IN   STD_LOGIC;
           B : IN   STD_LOGIC;
           Q : OUT  STD_LOGIC);
END XOR_Gate;

ARCHITECTURE Behavioral OF XOR_Gate IS
BEGIN
    Q <= A XOR B;
END Behavioral;
```

3- Combinational Circuits

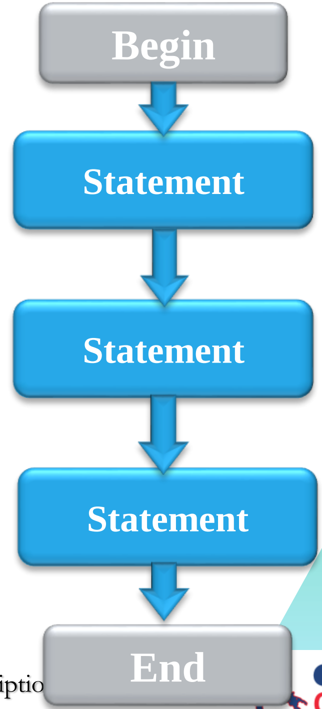
Concurrent vs. Sequential

Concurrent:



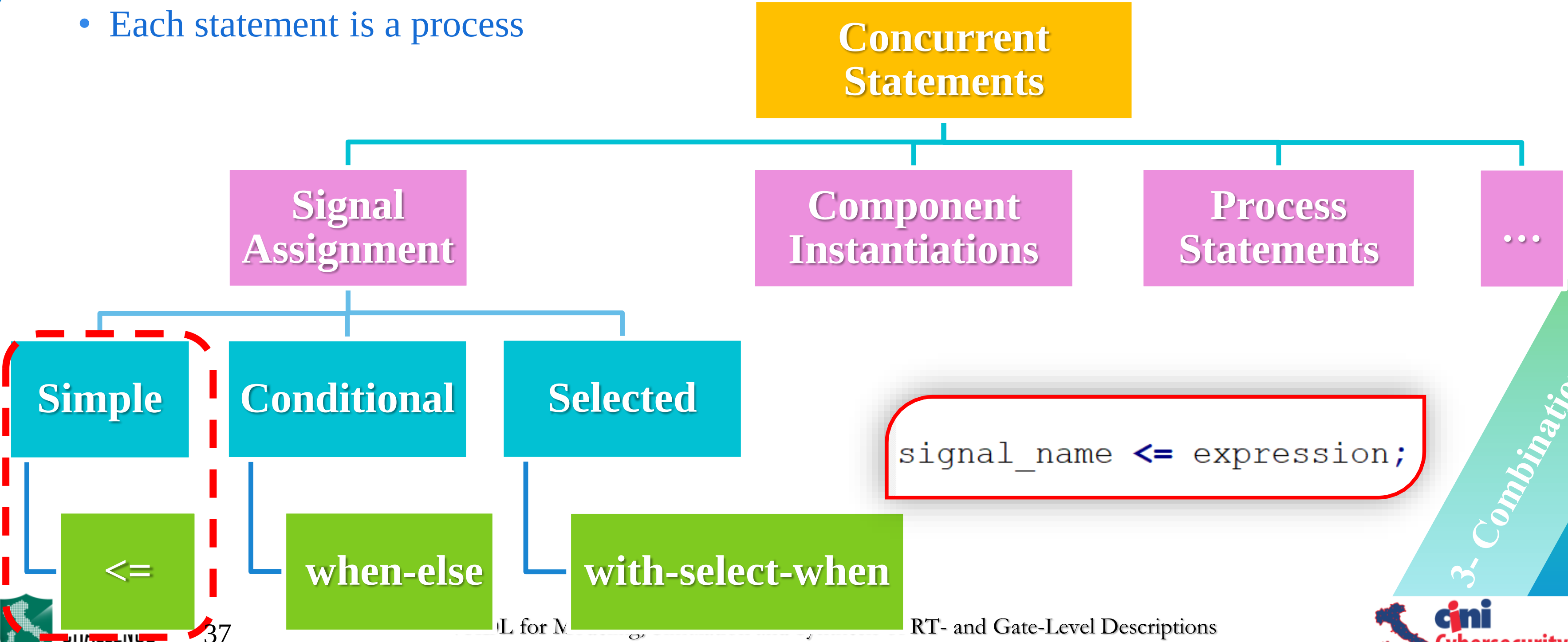
- VHDL architecture is a concurrent body
- A concurrent body can only include concurrent processes
- VHDL also allows sequential bodies
- A process statement is a sequential body, only sequential statement can go to sequential body

Sequential:



Concurrent Statements

- Concurrent statements are placed in the architecture body
- Each statement is a process

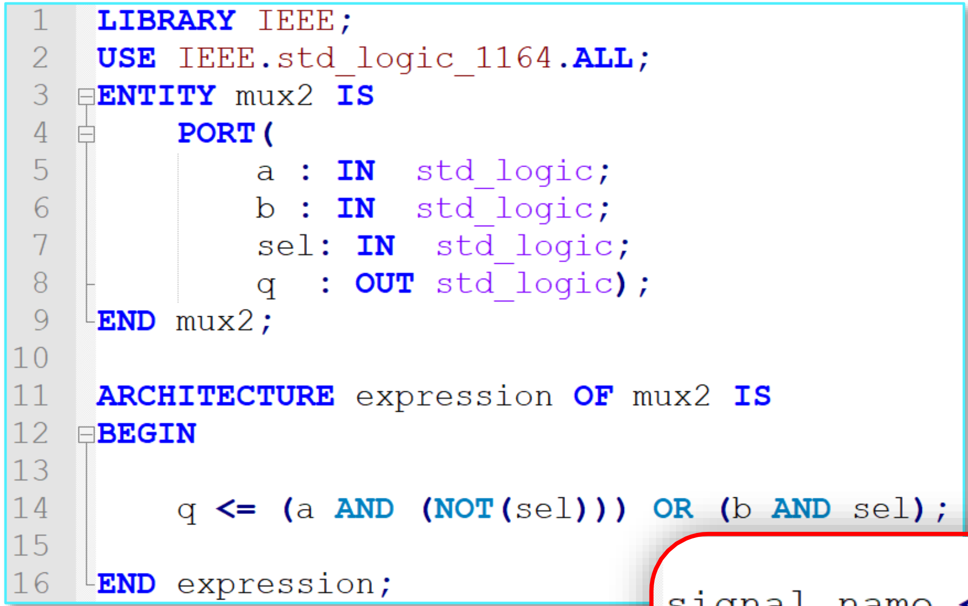


```
signal_name <= expression;
```

Simple Signal Assignment

- A concurrent signal assignment can have an expression on RHS
- Statements wait until an event occurs on any RHS signal

- Multiplexer

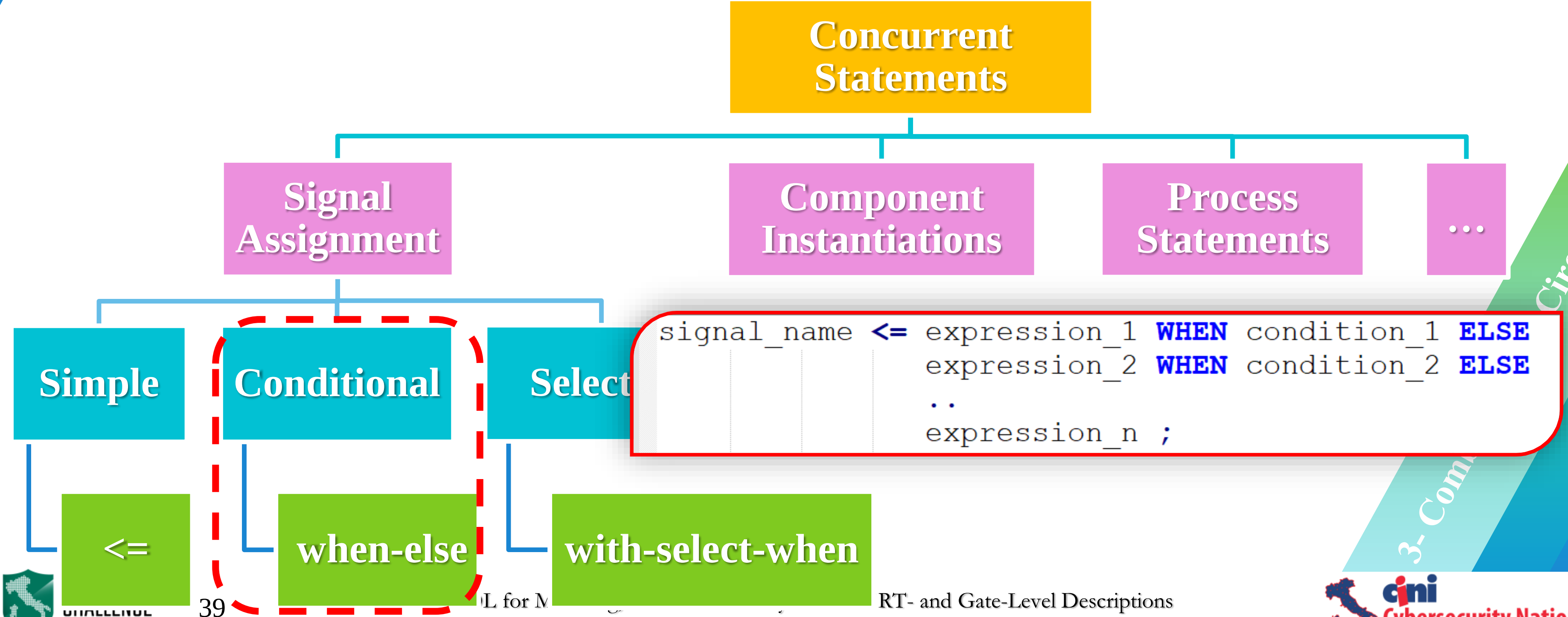


```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY mux2 IS
    PORT(
        a  : IN  std_logic;
        b  : IN  std_logic;
        sel: IN  std_logic;
        q  : OUT std_logic);
END mux2;

ARCHITECTURE expression OF mux2 IS
BEGIN

    q <= (a AND (NOT(sel))) OR (b AND sel);

END expression;
```

```
signal_name <= expression;
```

Concurrent Statements

- An architecture body can include a conditional signal assignment



```
signal_name <= expression_1 WHEN condition_1 ELSE
               expression_2 WHEN condition_2 ELSE
               ..
               expression_n ;
```

Conditional Concurrent Signal Assignment

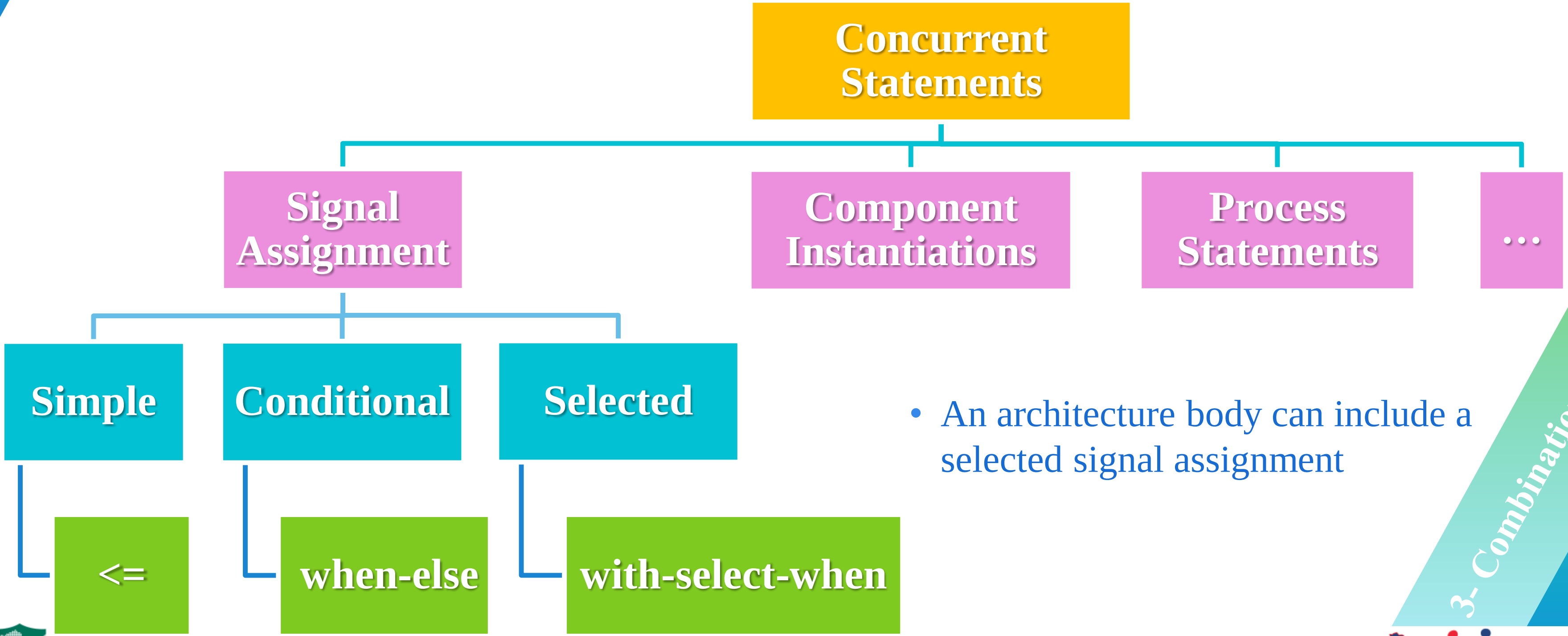
- Multiplexer

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY mux4 IS
    PORT(
        a1  : IN  std_logic_vector(2 DOWNTO 0);
        a2  : IN  std_logic_vector(2 DOWNTO 0);
        a3  : IN  std_logic_vector(2 DOWNTO 0);
        a4  : IN  std_logic_vector(2 DOWNTO 0);
        sel: IN  std_logic_vector(1 DOWNTO 0);
        q   : OUT std_logic_vector(2 DOWNTO 0));
END mux4;

ARCHITECTURE Dataflow OF mux4 IS
BEGIN

    q <= a1 WHEN (sel = "00") ELSE
         a2 WHEN (sel = "01") ELSE
         a3 WHEN (sel = "10") ELSE
         a4 WHEN (sel = "11") ELSE
         "XXX";

END Dataflow;
```

- A conditional signal assignment can include a number of expressions followed by their corresponding conditions
- Sensitive to events on the RHS

```
signal_name <= expression_1 WHEN condition_1 ELSE
               expression_2 WHEN condition_2 ELSE
               ..
               expression_n ;
```

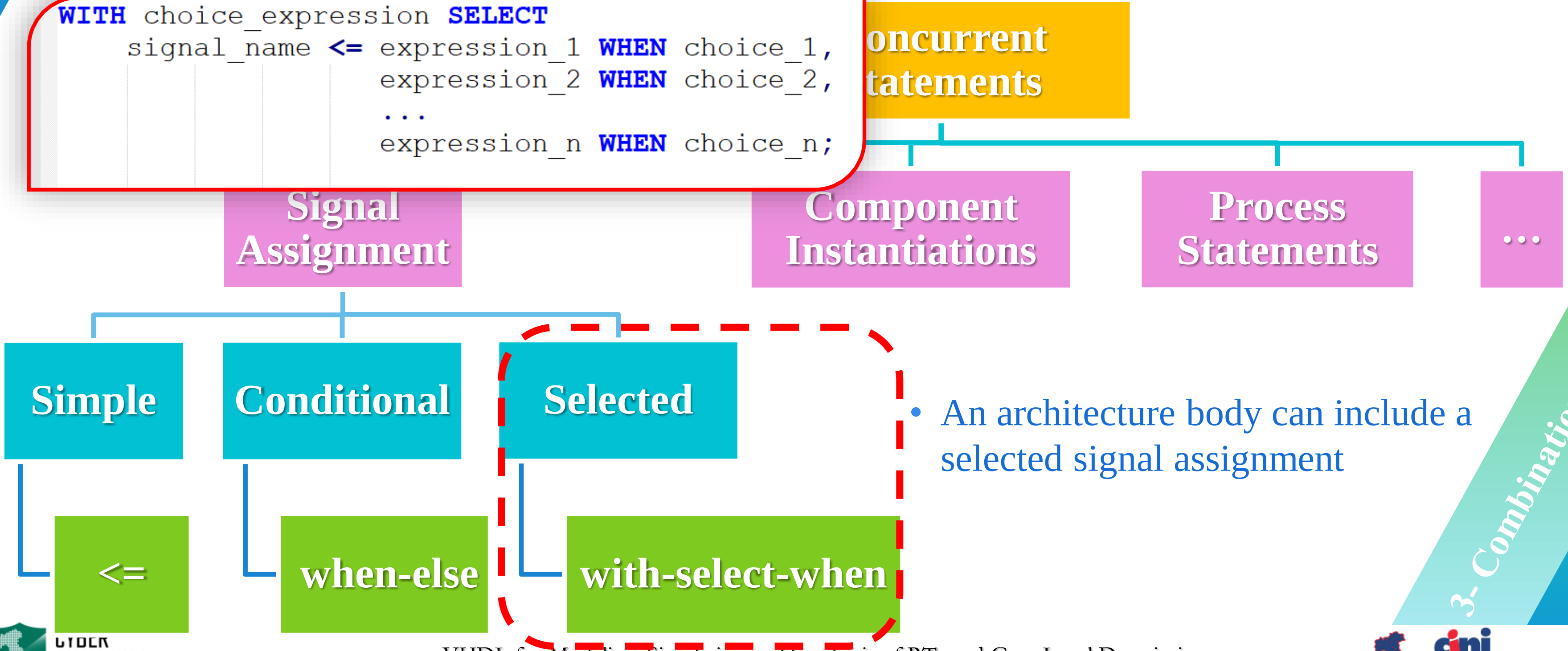
Concurrent Statements



- An architecture body can include a selected signal assignment

Concurrent Statements

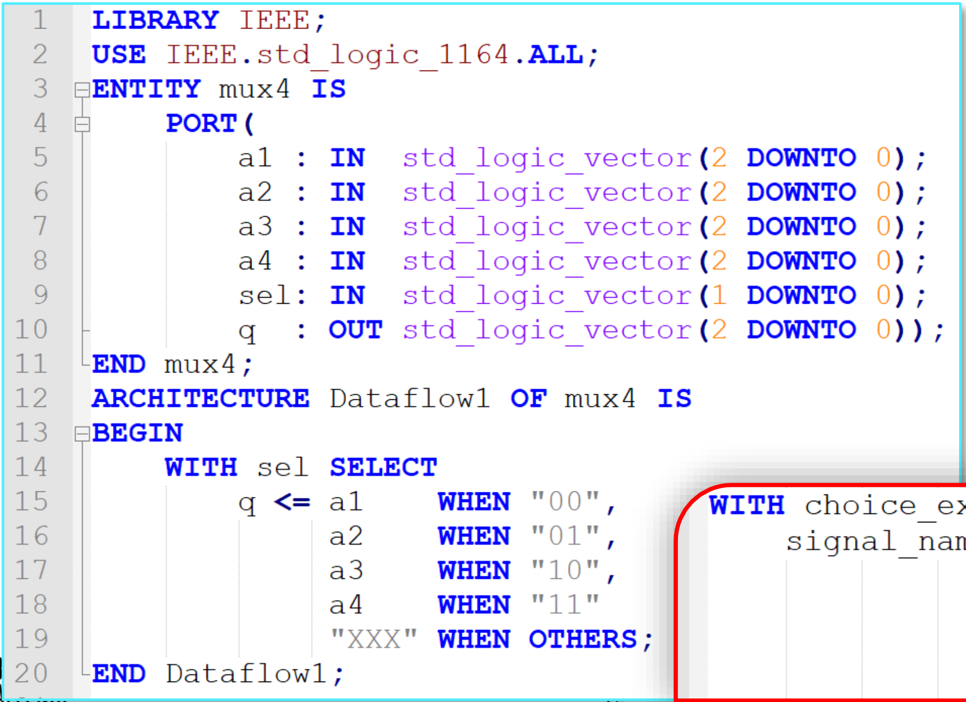
```
WITH choice_expression SELECT
    signal_name <= expression_1 WHEN choice_1,
                   expression_2 WHEN choice_2,
                   ...
                   expression_n WHEN choice_n;
```



- An architecture body can include a selected signal assignment

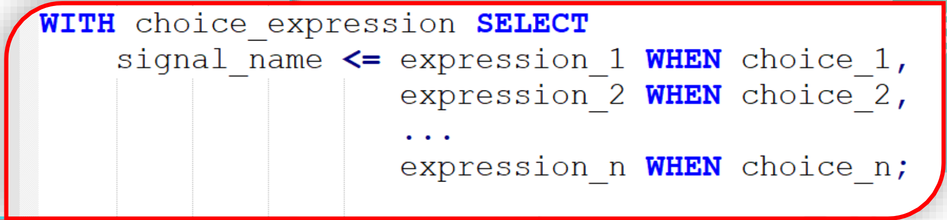
Selected Concurrent Signal Assignment

- Multiplexer



```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY mux4 IS
    PORT(
        a1  : IN  std_logic_vector(2 DOWNTO 0);
        a2  : IN  std_logic_vector(2 DOWNTO 0);
        a3  : IN  std_logic_vector(2 DOWNTO 0);
        a4  : IN  std_logic_vector(2 DOWNTO 0);
        sel : IN  std_logic_vector(1 DOWNTO 0);
        q   : OUT std_logic_vector(2 DOWNTO 0));
END mux4;
ARCHITECTURE Dataflow1 OF mux4 IS
BEGIN
    WITH sel SELECT
        q <= a1    WHEN "00",
             a2    WHEN "01",
             a3    WHEN "10",
             a4    WHEN "11"
             "XXX" WHEN OTHERS;
END Dataflow1;
```

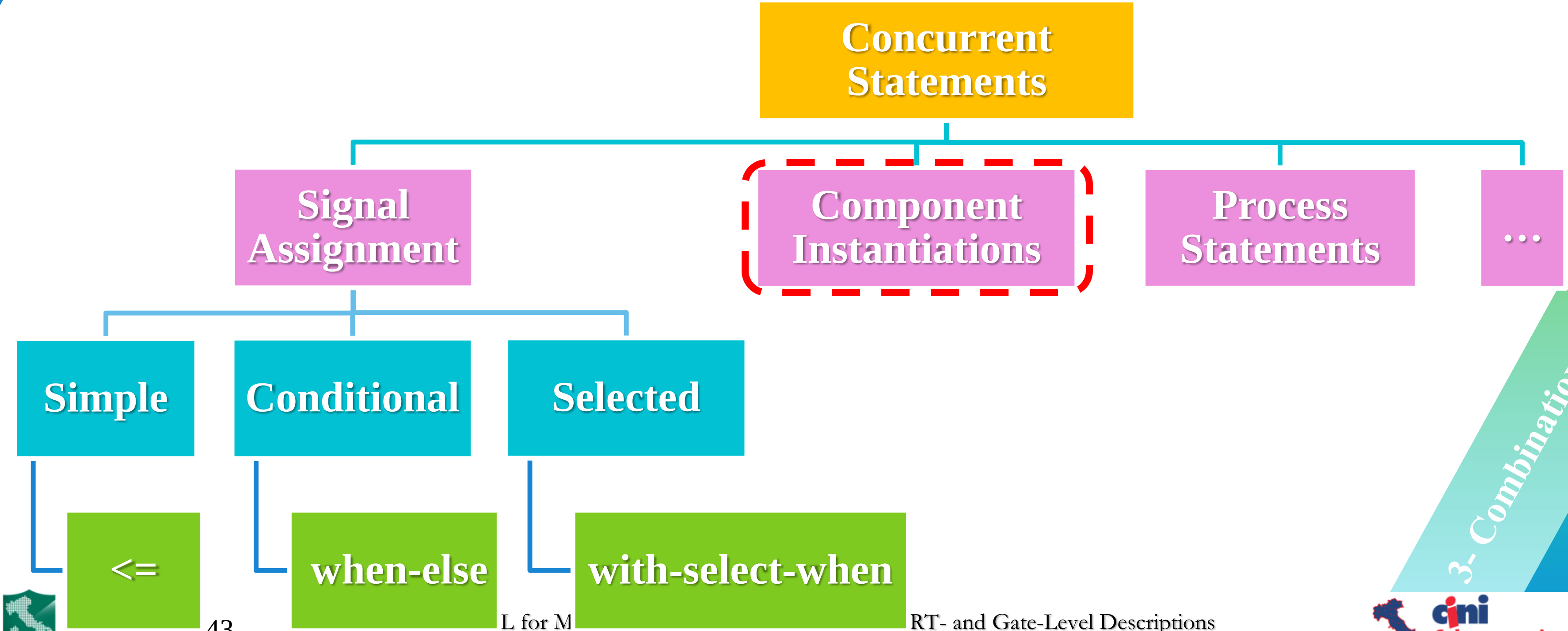
- There are expressions and choices
- **OTHERS** means none of the above
- Sensitive to events on the RHS



```
WITH choice_expression SELECT
    signal_name <= expression_1 WHEN choice_1,
                   expression_2 WHEN choice_2,
                   ...
                   expression_n WHEN choice_n;
```

Concurrent Statements

- Another concurrent process that fits in an architecture is component instantiation

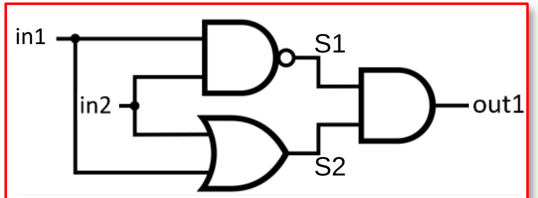


Component Direct Instantiation

- Component instantiation identifies an instantiated component and how ports are connected
- Component instantiation is sensitive to inputs of the component being instantiated

- XOR using direct instantiations

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY XOR_Gate_Dire IS
    PORT ( in1 : IN   STD_LOGIC;
           in2 : IN   STD_LOGIC;
           out1: OUT  STD_LOGIC);
END XOR_Gate_Dire;

ARCHITECTURE Behavioral OF XOR_Gate_Dire IS
    SIGNAL S1, S2: STD_LOGIC;
BEGIN
    U1: ENTITY WORK.NAND_Gate PORT MAP (A => in1, B => in2, Q => S1);
    U2: ENTITY WORK.OR_Gate   PORT MAP (A => in1, B => in2, Q => S2);
    U3: ENTITY WORK.AND_Gate  PORT MAP (A => S1, B => S2, Q => out1);
END Behavioral;
```



$$out1 = (int1 + int2).(\overline{int1} + \overline{int2})$$

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AND_Gate IS
    PORT ( A : IN   STD_LOGIC;
           B : IN   STD_LOGIC;
           Q : OUT  STD_LOGIC);
END AND_Gate;

ARCHITECTURE Behavioral OF AND_Gate IS
BEGIN
    Q <= A AND B ;
END Behavioral;
```

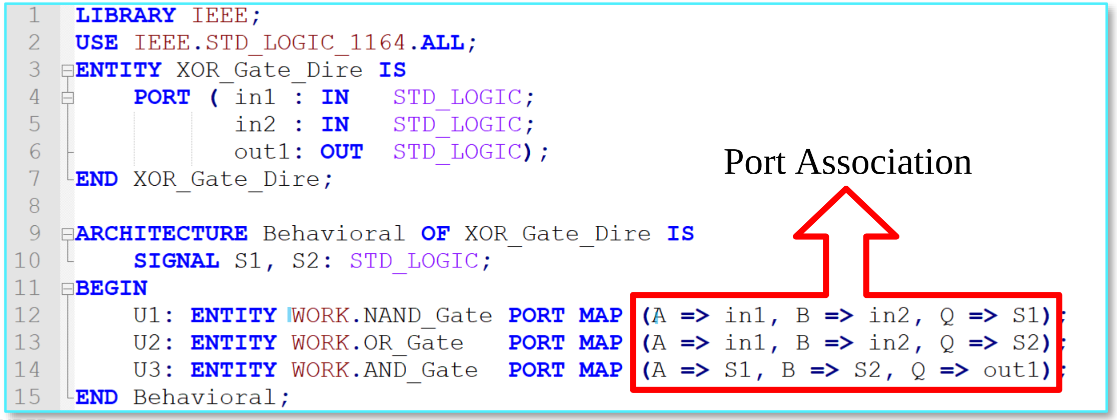
Concurrent Statements

Combinational Circuits

Component Direct Instantiation

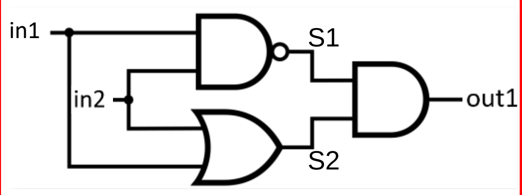
- Component instantiation identifies an instantiated component and how ports are connected
- Component instantiation is sensitive to inputs of the component being instantiated

- XOR using direct instantiations

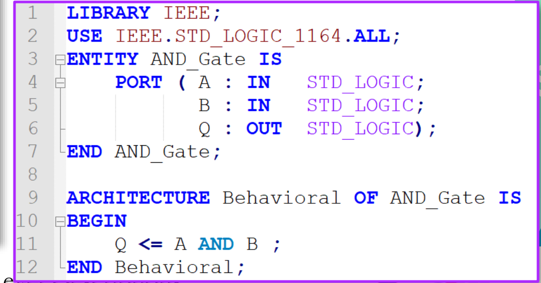


```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY XOR_Gate_Dire IS
    PORT ( in1  : IN   STD_LOGIC;
           in2  : IN   STD_LOGIC;
           out1: OUT  STD_LOGIC);
END XOR_Gate_Dire;

ARCHITECTURE Behavioral OF XOR_Gate_Dire IS
    SIGNAL S1, S2: STD_LOGIC;
BEGIN
    U1: ENTITY WORK.NAND_Gate PORT MAP (A => in1, B => in2, Q => S1);
    U2: ENTITY WORK.OR_Gate   PORT MAP (A => in1, B => in2, Q => S2);
    U3: ENTITY WORK.AND_Gate  PORT MAP (A => S1, B => S2, Q => out1);
END Behavioral;
```

Port Association



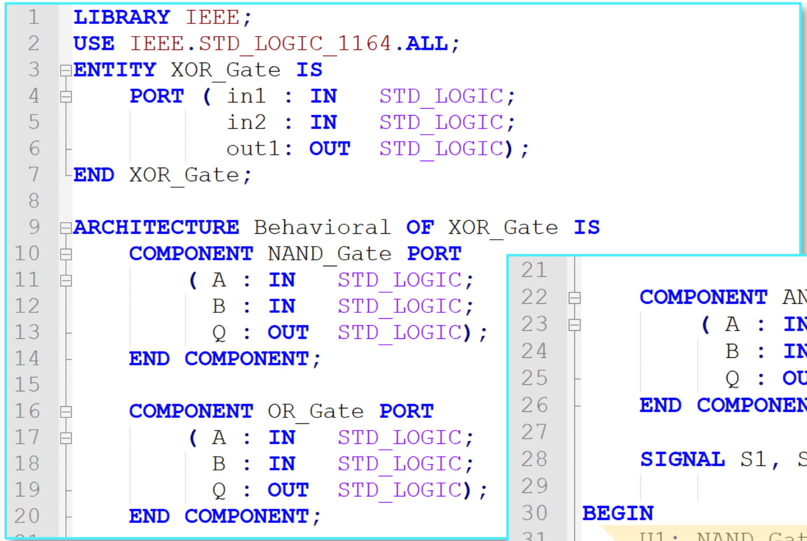
$$out1 = (int1 + int2).(\overline{int1} + \overline{int2})$$



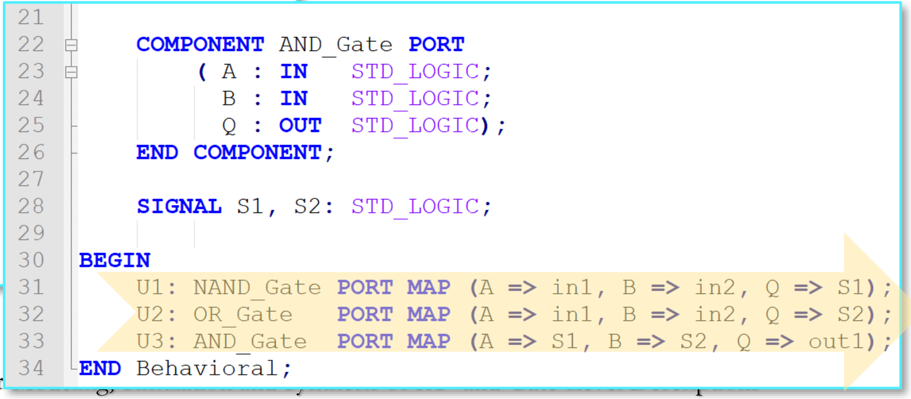
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AND_Gate IS
    PORT ( A : IN   STD_LOGIC;
           B : IN   STD_LOGIC;
           Q : OUT  STD_LOGIC);
END AND_Gate;

ARCHITECTURE Behavioral OF AND_Gate IS
BEGIN
    Q <= A AND B ;
END Behavioral;
```

Component Instantiations

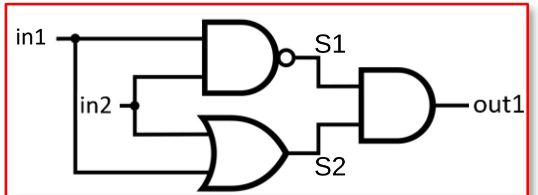
- A more flexible but harder way for component instantiation
  - XOR using component declaration



```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY XOR_Gate IS
    PORT ( in1 : IN   STD_LOGIC;
           in2 : IN   STD_LOGIC;
           out1: OUT  STD_LOGIC);
END XOR_Gate;

ARCHITECTURE Behavioral OF XOR_Gate IS
    COMPONENT NAND_Gate PORT
        ( A : IN   STD_LOGIC;
          B : IN   STD_LOGIC;
          Q : OUT  STD_LOGIC);
    END COMPONENT;

    COMPONENT OR_Gate PORT
        ( A : IN   STD_LOGIC;
          B : IN   STD_LOGIC;
          Q : OUT  STD_LOGIC);
    END COMPONENT;
```



```vhdl

    COMPONENT AND_Gate PORT
        ( A : IN   STD_LOGIC;
          B : IN   STD_LOGIC;
          Q : OUT  STD_LOGIC);
    END COMPONENT;

    SIGNAL S1, S2: STD_LOGIC;

BEGIN
    U1: NAND_Gate PORT MAP (A => in1, B => in2, Q => S1);
    U2: OR_Gate   PORT MAP (A => in1, B => in2, Q => S2);
    U3: AND_Gate  PORT MAP (A => S1, B => S2, Q => out1);
END Behavioral;
```

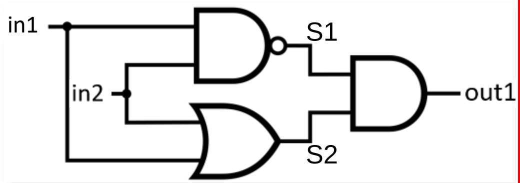


$$out1 = (int1 + int2).(\overline{int1} + \overline{int2})$$

Component Instantiations

- A more flexible but harder way for component instantiation
  - XOR using component declaration

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY XOR_Gate IS
     PORT ( in1  : IN   STD_LOGIC;
            in2  : IN   STD_LOGIC;
            out1: OUT   STD_LOGIC);
END XOR_Gate;

ARCHITECTURE Behavioral OF XOR_Gate IS
     COMPONENT NAND_Gate PORT
          ( A : IN   STD_LOGIC;
            B : IN   STD_LOGIC;
            Q : OUT  STD_LOGIC);
     END COMPONENT;

     COMPONENT OR_Gate PORT
          ( A : IN   STD_LOGIC;
            B : IN   STD_LOGIC;
            Q : OUT  STD_LOGIC);
     END COMPONENT;

     COMPONENT AND_Gate PORT
          ( A  :  IN    STD_LOGIC;
            B  :  IN    STD_LOGIC;
            Q  :  OUT   STD_LOGIC);
     END COMPONENT;

     SIGNAL S1, S2: STD_LOGIC;

BEGIN
     U1: NAND_Gate PORT MAP (A => in1, B => in2, Q => S1);
     U2: OR_Gate   PORT MAP (A => in1, B => in2, Q => S2);
     U3: AND_Gate  PORT MAP (A => S1, B => S2, Q => out1);
END Behavioral;
```



$$out1 = (int1 + int2).(\overline{int1} + \overline{int2})$$

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AND_Gate IS
     PORT ( A : IN   STD_LOGIC;
            B : IN   STD_LOGIC;
            Q : OUT  STD_LOGIC);
END AND_Gate;

ARCHITECTURE Behavioral OF AND_Gate IS
BEGIN
     Q <= A AND B ;
END Behavioral;
```

3- Combinational Circuits

Concurrent Statements

Examples:

- Select a waveform based on its corresponding choice that matches *sel*
- Vertical bar can be used for multiple choices, e.g.

```
"0010" WHEN "00" | "01",
```

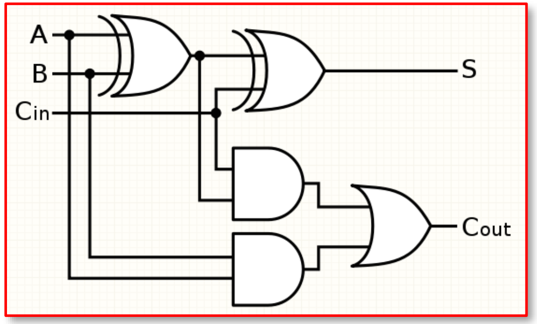
- Decoder

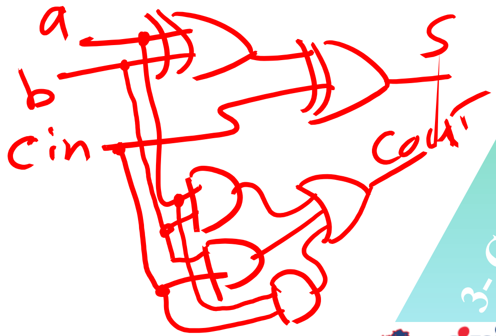
```
ENTITY dcd2to4 IS
    PORT (sel: IN  std_logic_vector (1 DOWNTO 0);
          y:   OUT std_logic_vector (3 DOWNTO 0));
END dcd2to4;

ARCHITECTURE RTL OF dcd2to4 IS
BEGIN

    WITH sel SELECT
        y <= "0001" WHEN "00",
             "0010" WHEN "01",
             "0100" WHEN "10",
             "1000" WHEN "11",
             "0000" WHEN OTHERS;

END ARCHITECTURE RTL;
```

sel		y3	y2	y1	y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Examples:

- An architecture can include any number of concurrent statements



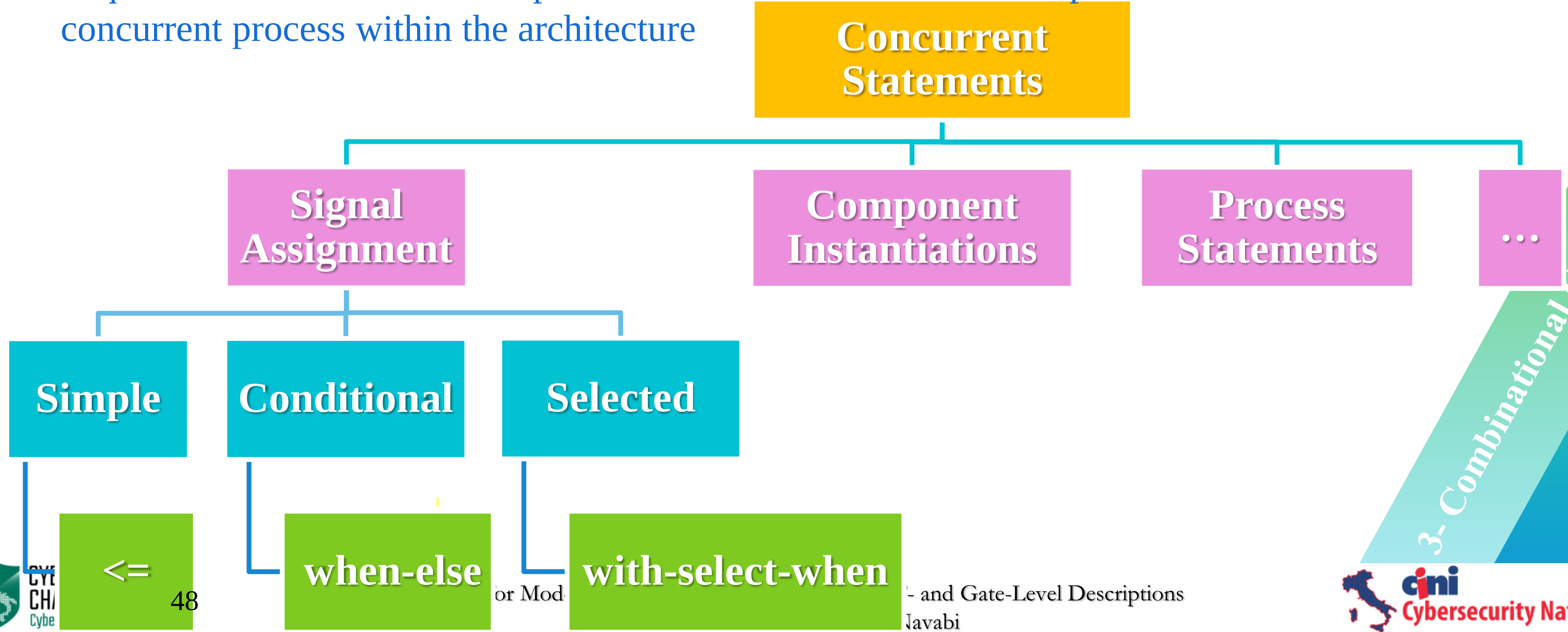


- Full Adder

```vhdl
ENTITY full_adder IS
    PORT (a, b, cin : IN  std_logic;
          sum, cout : OUT std_logic);
END ENTITY full_adder;
--
ARCHITECTURE expression OF full_adder IS
BEGIN

    sum  <= a XOR b XOR cin;
    cout <= (a AND b) OR (a AND cin) OR (b AND cin);

END ARCHITECTURE expression;
```

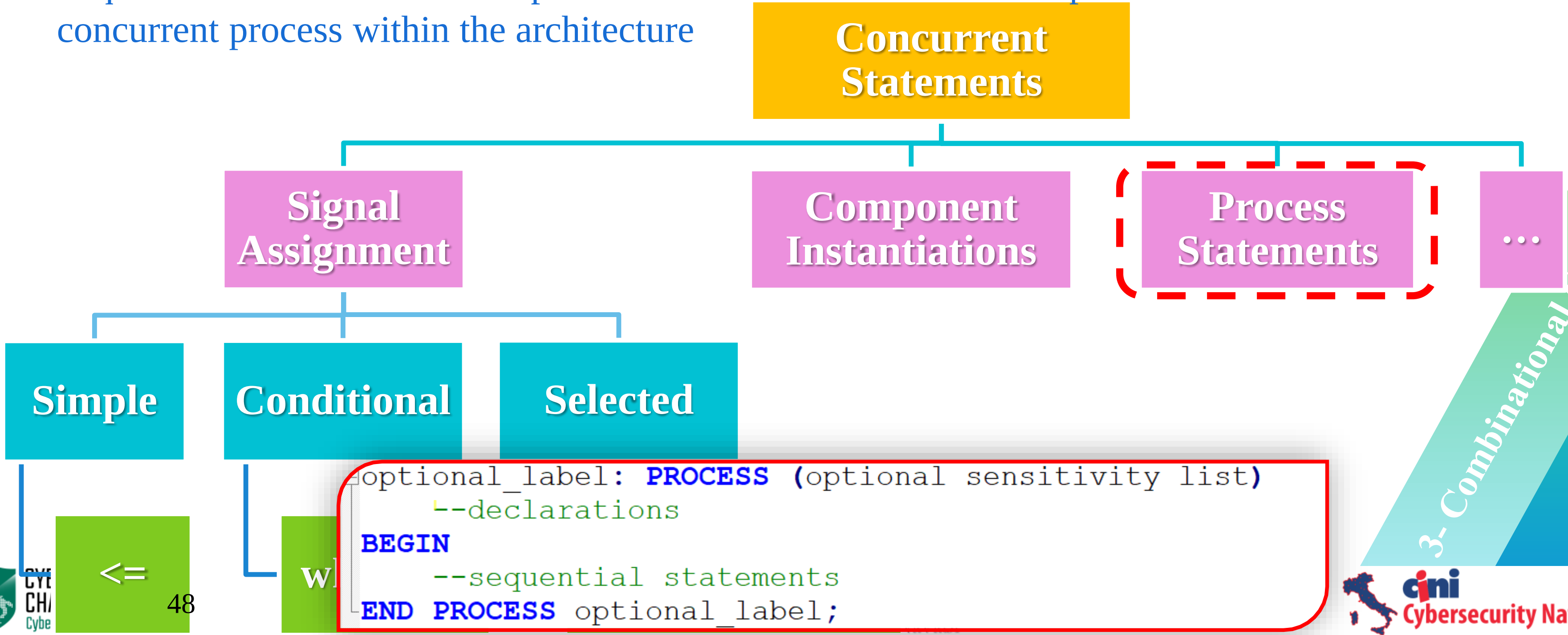
Concurrent Statements

- Body of a process statement is a sequential body within which sequential statements fit
- Sequential statements within a process statements make the entire process statement a concurrent process within the architecture



Concurrent Statements

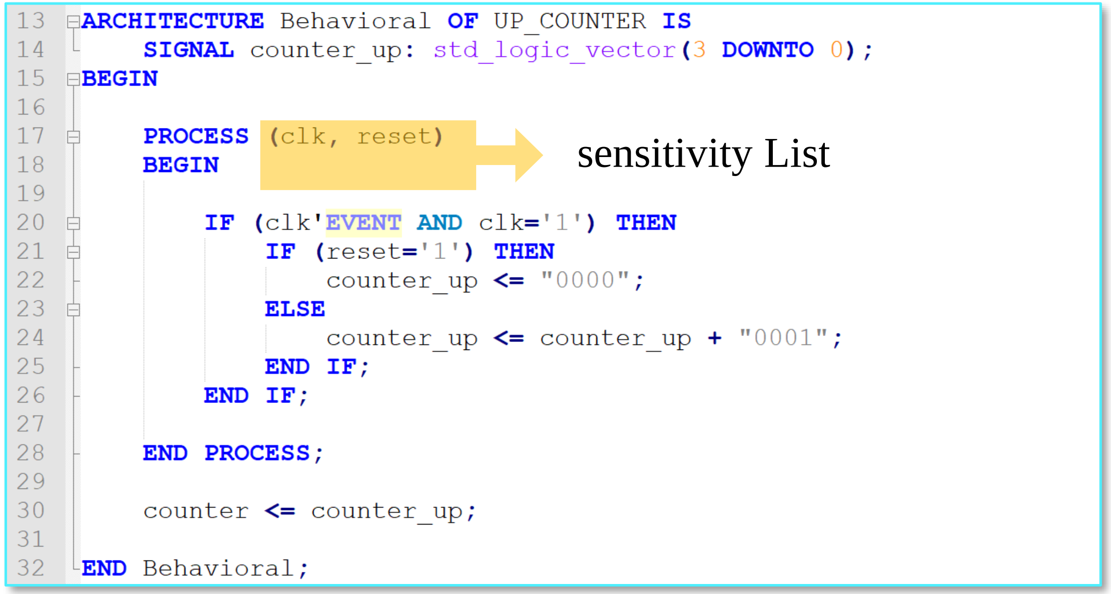
- Body of a process statement is a sequential body within which sequential statements fit
- Sequential statements within a process statements make the entire process statement a concurrent process within the architecture



```
optional_label: PROCESS (optional sensitivity list)
    --declarations
BEGIN
    --sequential statements
END PROCESS optional_label;
```

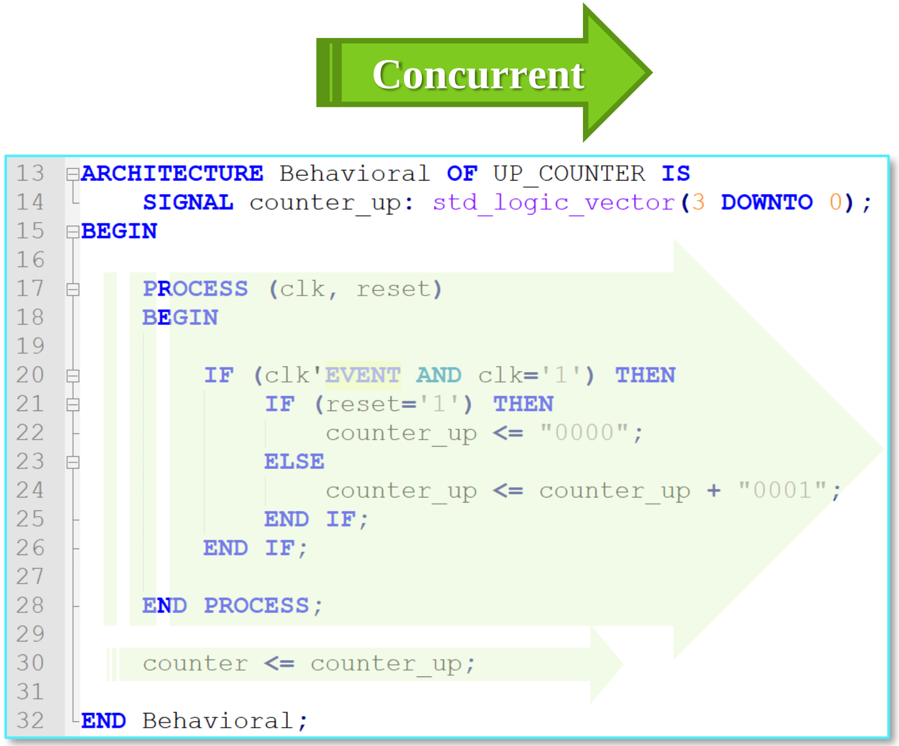
Process Statement => Sensitivity List

- The process sensitivity list, lists the signals that will cause the process statement to be executed
- Any transition on any of the signals in the signal sensitivity list will cause the process to execute



```vhdl
ARCHITECTURE Behavioral OF UP_COUNTER IS
    SIGNAL counter_up: std_logic_vector(3 DOWNTO 0);
BEGIN

    PROCESS (clk, reset)
    BEGIN

        IF (clk'EVENT AND clk='1') THEN
            IF (reset='1') THEN
                counter_up <= "0000";
            ELSE
                counter_up <= counter_up + "0001";
            END IF;
        END IF;

    END PROCESS;

    counter <= counter_up;

END Behavioral;
```

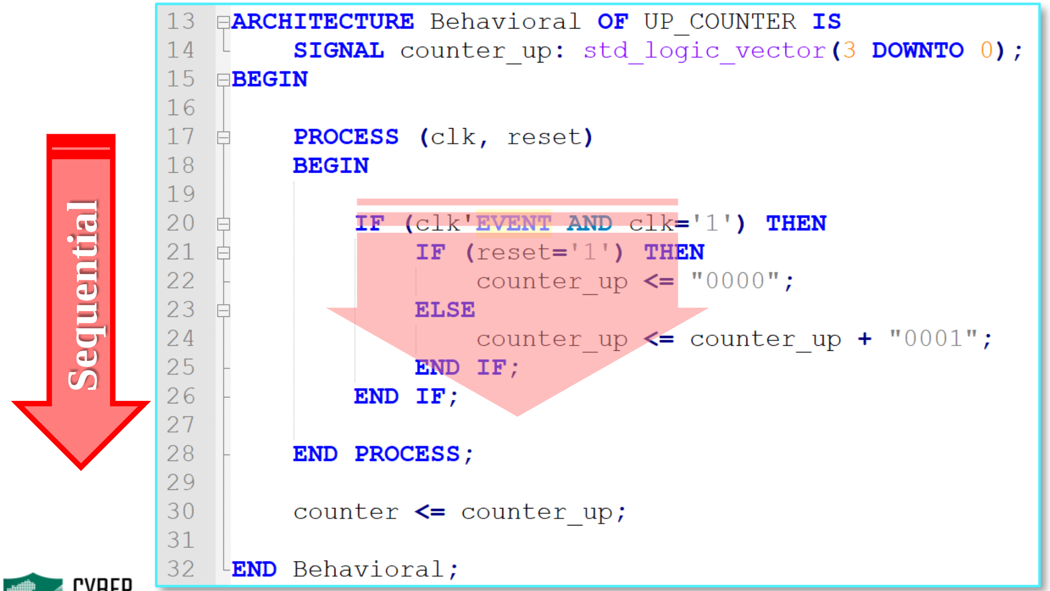
Process Statements => Body



```
ARCHITECTURE Behavioral OF UP_COUNTER IS
    SIGNAL counter_up: std_logic_vector(3 DOWNTO 0);
BEGIN

    PROCESS (clk, reset)
    BEGIN

        IF (clk'EVENT AND clk='1') THEN
            IF (reset='1') THEN
                counter_up <= "0000";
            ELSE
                counter_up <= counter_up + "0001";
            END IF;
        END IF;

    END PROCESS;

    counter <= counter_up;

END Behavioral;
```

- Two concurrent processes in this architecture
- One is process statement sensitive to *clk* and *reset*
- The other is a signal assignment sensitive to *counter_up*

Process Statements => Body

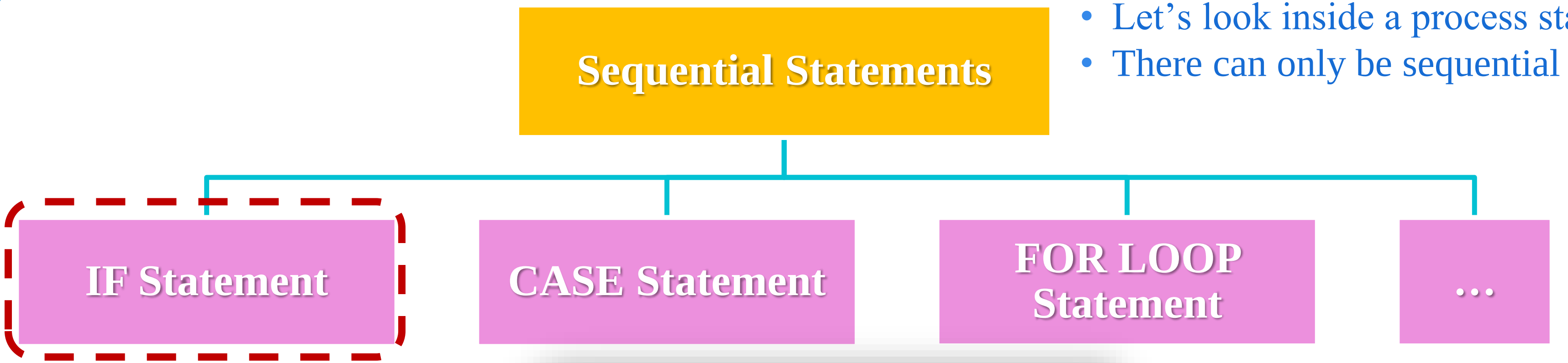


```
ARCHITECTURE Behavioral OF UP_COUNTER IS
    SIGNAL counter_up: std_logic_vector(3 DOWNTO 0);
BEGIN

    PROCESS (clk, reset)
    BEGIN

        IF (clk'EVENT AND clk='1') THEN
            IF (reset='1') THEN
                counter_up <= "0000";
            ELSE
                counter_up <= counter_up + "0001";
            END IF;
        END IF;

    END PROCESS;

    counter <= counter_up;

END Behavioral;
```

- Two concurrent processes in this architecture
- One is process statement sensitive to *clk* and *reset*
- The other is a signal assignment sensitive to *counter_up*

Sequential Statements

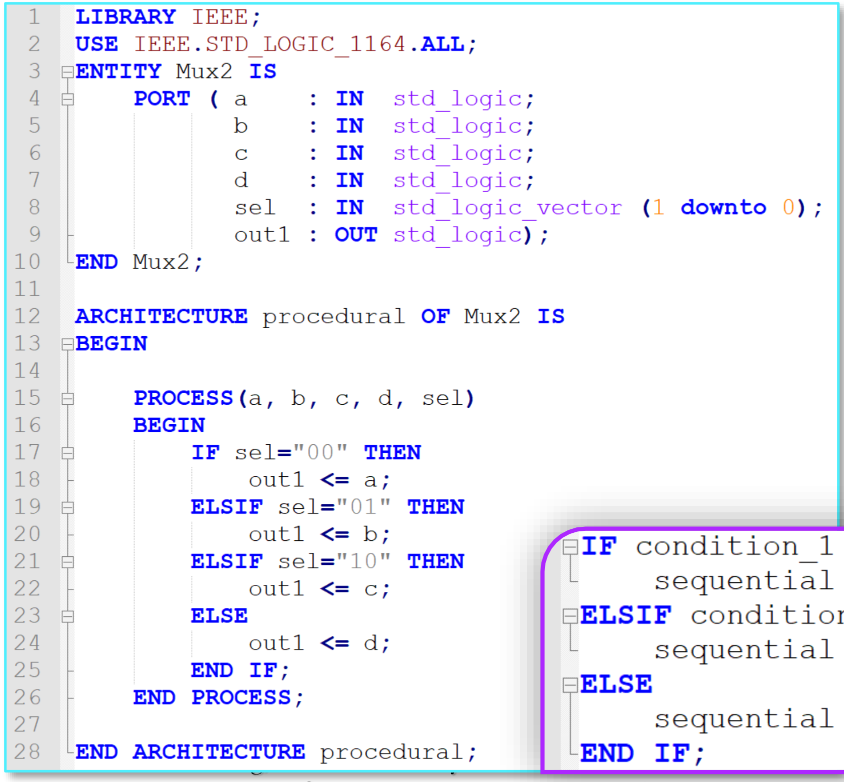
- Let’s look inside a process statement
- There can only be sequential statements



```
IF condition_1 THEN
     sequential statements;
ELSIF condition2 THEN
     sequential statements;
ELSE
     sequential statements;
END IF;
```

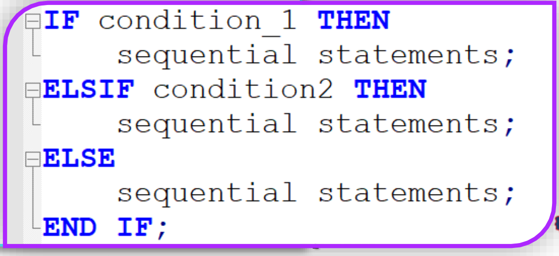
IF Statement

- Procedural Multiplexer



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Mux2 IS
    PORT ( a    : IN  std_logic;
           b    : IN  std_logic;
           c    : IN  std_logic;
           d    : IN  std_logic;
           sel  : IN  std_logic_vector (1 downto 0);
           out1 : OUT std_logic);
END Mux2;

ARCHITECTURE procedural OF Mux2 IS
BEGIN

    PROCESS(a, b, c, d, sel)
    BEGIN
        IF sel="00" THEN
            out1 <= a;
        ELSIF sel="01" THEN
            out1 <= b;
        ELSIF sel="10" THEN
            out1 <= c;
        ELSE
            out1 <= d;
        END IF;
    END PROCESS;

END ARCHITECTURE procedural;
```

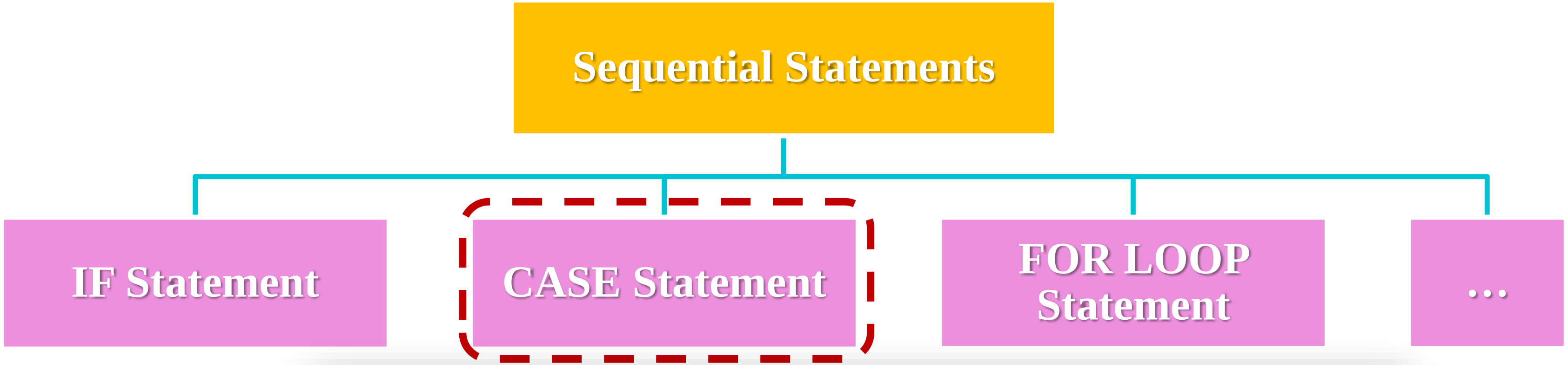
- A process statement with sensitivity list
- A concurrent process inside which is a conditional sequential if statement



```
IF condition_1 THEN
    sequential statements;
ELSIF condition2 THEN
    sequential statements;
ELSE
    sequential statements;
END IF;
```

Sequential Statements

- Case statement is another sequential statement

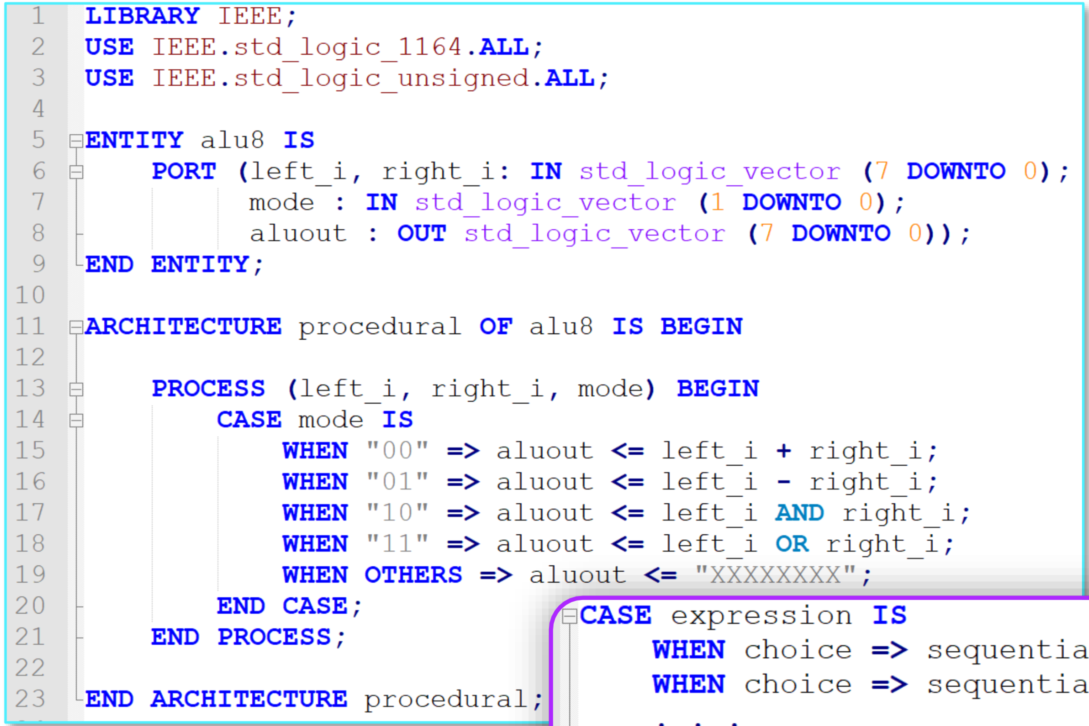


```
CASE expression IS
    WHEN choice => sequential statements;
    WHEN choice => sequential statements;
    . . .
    END CASE;
```

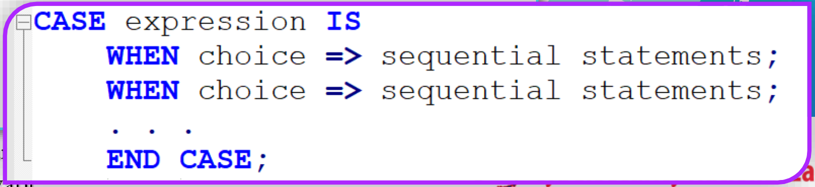
CASE Statement

- Procedural ALU

- A concurrent process with the sensitivity list
- A sequential case statement with choices selecting other sequential statements
- When multiple choices exist, separate them with vertical bar, e.g., “00” | “0Z”



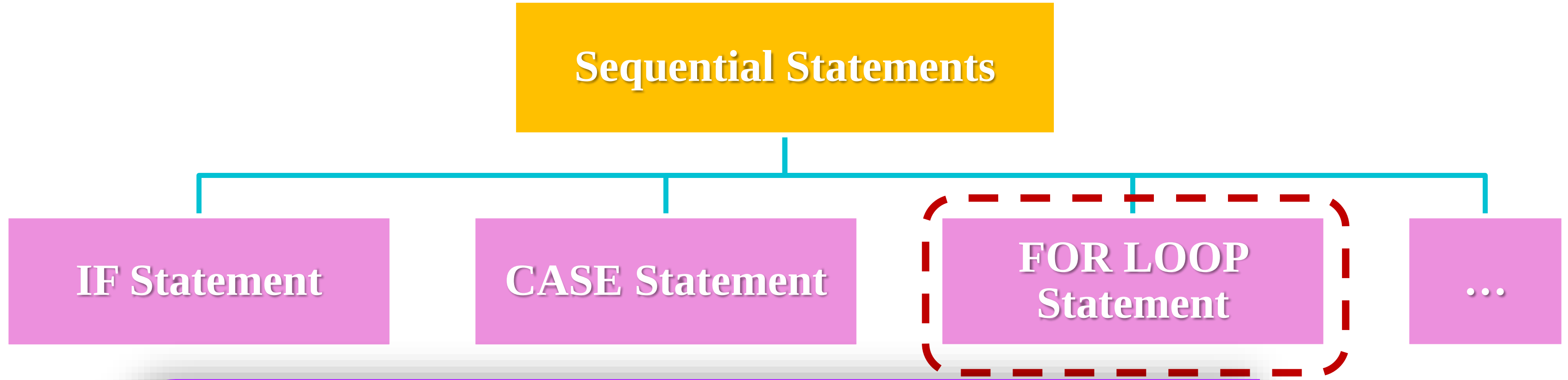
```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY alu8 IS
    PORT (left_i, right_i: IN std_logic_vector (7 DOWNTO 0);
          mode : IN std_logic_vector (1 DOWNTO 0);
          aluout : OUT std_logic_vector (7 DOWNTO 0));
END ENTITY;

ARCHITECTURE procedural OF alu8 IS BEGIN

    PROCESS (left_i, right_i, mode) BEGIN
        CASE mode IS
            WHEN "00" => aluout <= left_i + right_i;
            WHEN "01" => aluout <= left_i - right_i;
            WHEN "10" => aluout <= left_i AND right_i;
            WHEN "11" => aluout <= left_i OR right_i;
            WHEN OTHERS => aluout <= "XXXXXXXX";
        END CASE;
    END PROCESS;

END ARCHITECTURE procedural;
```

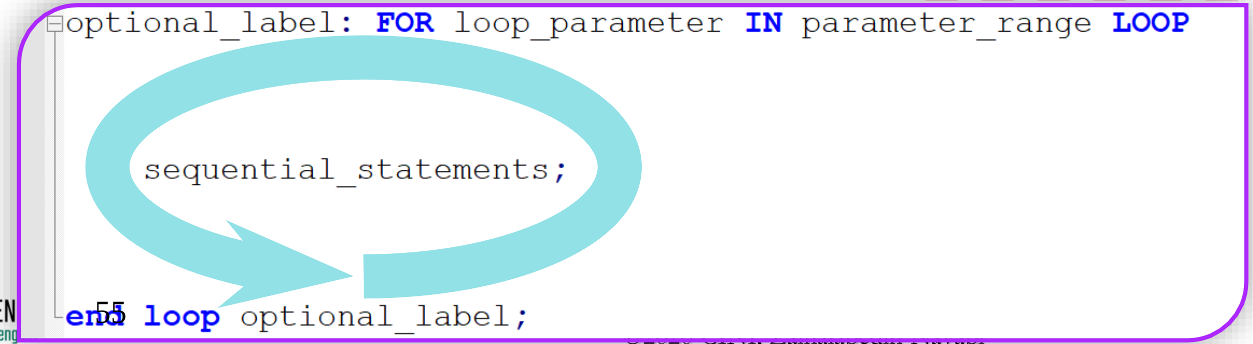


```vhdl
CASE expression IS
    WHEN choice => sequential statements;
    WHEN choice => sequential statements;
    . . .
    END CASE;
```

Sequential Statements

- Another sequential statement is a for loop statement

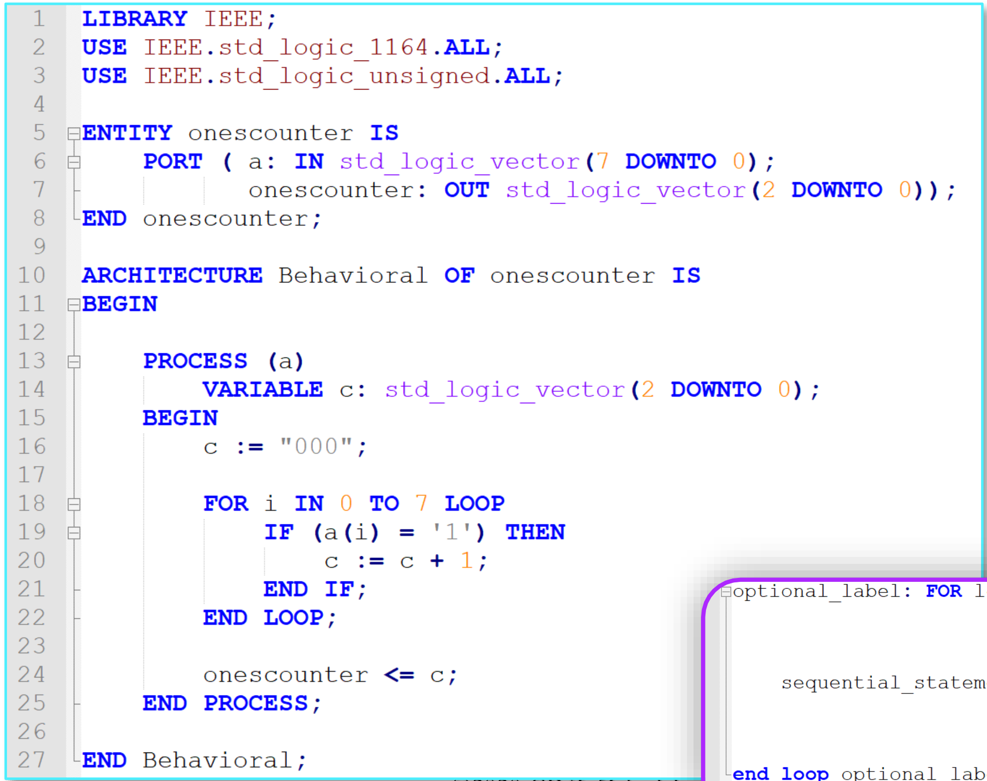




```
optional_label: FOR loop_parameter IN parameter_range LOOP

    sequential_statements;

end loop optional_label;
```

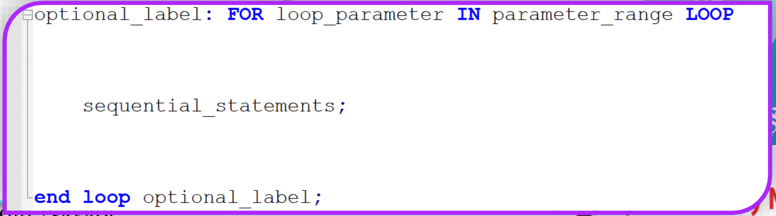
FOR LOOP Statement

- Ones counter



```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY onescounter IS
    PORT ( a: IN std_logic_vector(7 DOWNTO 0);
           onescounter: OUT std_logic_vector(2 DOWNTO 0));
END onescounter;

ARCHITECTURE Behavioral OF onescounter IS
BEGIN

    PROCESS (a)
        VARIABLE c: std_logic_vector(2 DOWNTO 0);
    BEGIN
        c := "000";

        FOR i IN 0 TO 7 LOOP
            IF (a(i) = '1') THEN
                c := c + 1;
            END IF;
        END LOOP;

        onescounter <= c;
    END PROCESS;

END Behavioral;
```

- A process statement with signal *a* in its sensitivity list; variable *c* is declared that is only available within the process statement
- There are a total 5 sequential statements here

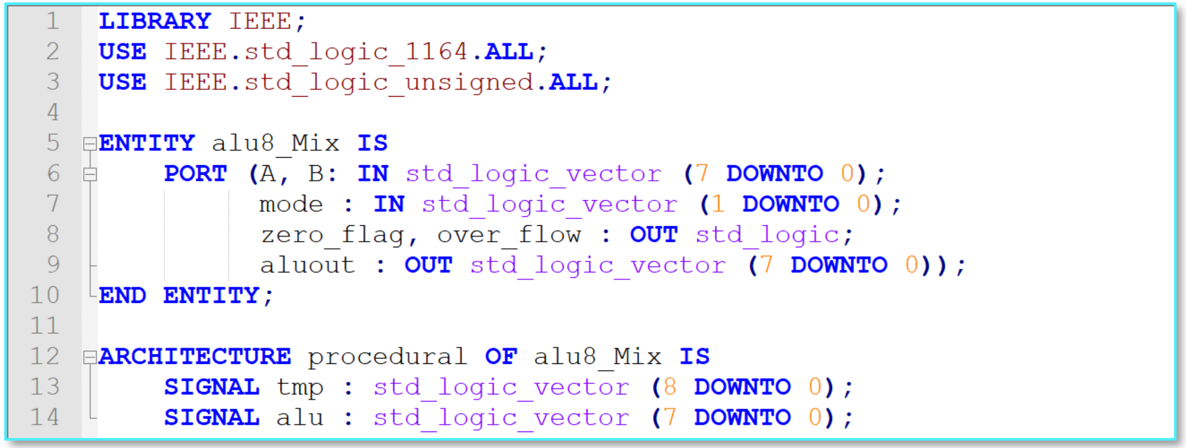


```vhdl
optional_label: FOR loop_parameter IN parameter_range LOOP

    sequential_statements;

end loop optional_label;
```

National Lab

Examples:

- Multiple Concurrent Statements

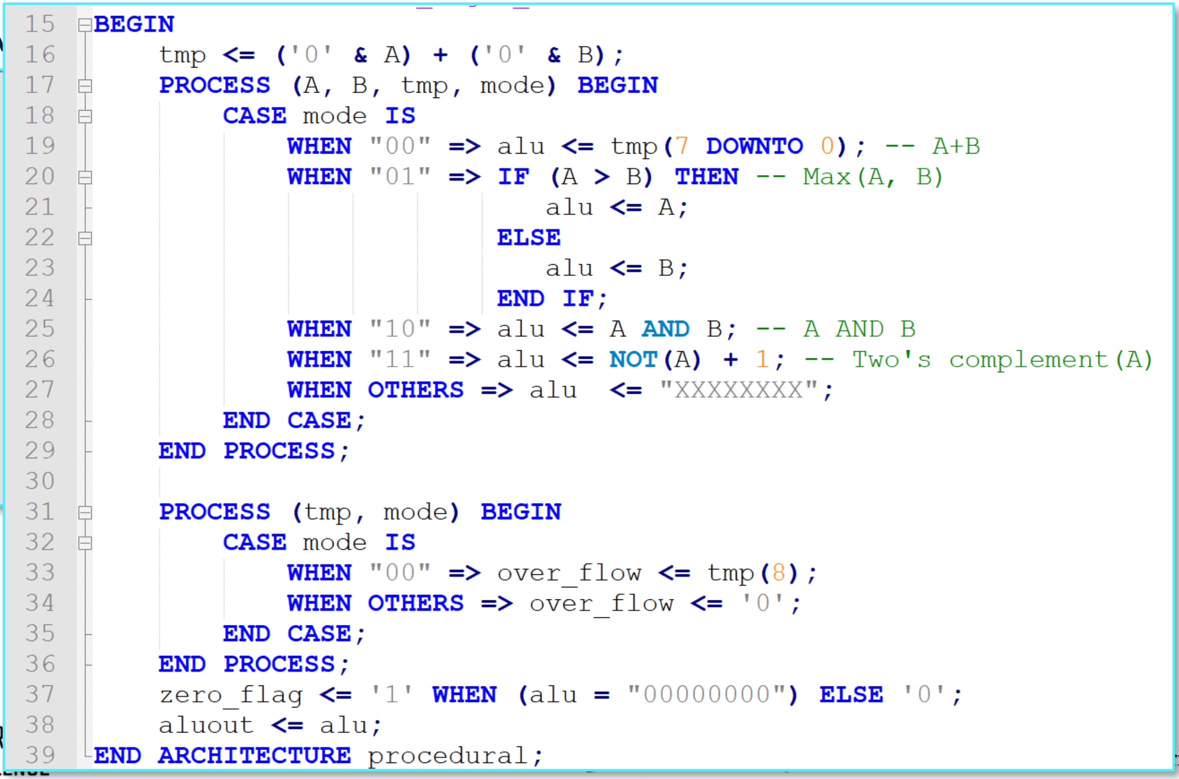


```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY alu8_Mix IS
    PORT (A, B: IN std_logic_vector (7 DOWNTO 0);
          mode : IN std_logic_vector (1 DOWNTO 0);
          zero_flag, over_flow : OUT std_logic;
          aluout : OUT std_logic_vector (7 DOWNTO 0));
END ENTITY;

ARCHITECTURE procedural OF alu8_Mix IS
    SIGNAL tmp : std_logic_vector (8 DOWNTO 0);
    SIGNAL alu : std_logic_vector (7 DOWNTO 0);
```

- An architecture can include signal assignments
- An architecture can have any number of concurrent statements
- There are a total of 5 concurrent statements here

Examples:

- N



```
BEGIN
    tmp <= ('0' & A) + ('0' & B);
    PROCESS (A, B, tmp, mode) BEGIN
        CASE mode IS
            WHEN "00" => alu <= tmp(7 DOWNTO 0); -- A+B
            WHEN "01" => IF (A > B) THEN -- Max(A, B)
                            alu <= A;
                         ELSE
                            alu <= B;
                         END IF;
            WHEN "10" => alu <= A AND B; -- A AND B
            WHEN "11" => alu <= NOT(A) + 1; -- Two's complement(A)
            WHEN OTHERS => alu  <= "XXXXXXXX";
        END CASE;
    END PROCESS;

    PROCESS (tmp, mode) BEGIN
        CASE mode IS
            WHEN "00" => over_flow <= tmp(8);
            WHEN OTHERS => over_flow <= '0';
        END CASE;
    END PROCESS;
    zero_flag <= '1' WHEN (alu = "00000000") ELSE '0';
    aluout <= alu;
END ARCHITECTURE procedural;
```

- An architecture can include signal assignments
- An architecture can have any number of concurrent statements
- There are a total of 5 concurrent statements here