CPE 462
VHDL: Simulation and Synthesis

Topic #05 - c) BLOCK statements

Nuno Alves (nalves@wne.edu), College of Engineering

BLOCK statement

- The BLOCK statement, in its simple form, represents only a way of locally partitioning the code.
- It allows a set of concurrent statements to be clustered into a BLOCK, with the purpose of turning the overall code more readable and more manageable

```
label: BLOCK
  [declarative part]
BEGIN
  (concurrent statements)
END BLOCK label;
```

“Blocked” code

Therefore, the overall aspect of a “blocked” code is the following:

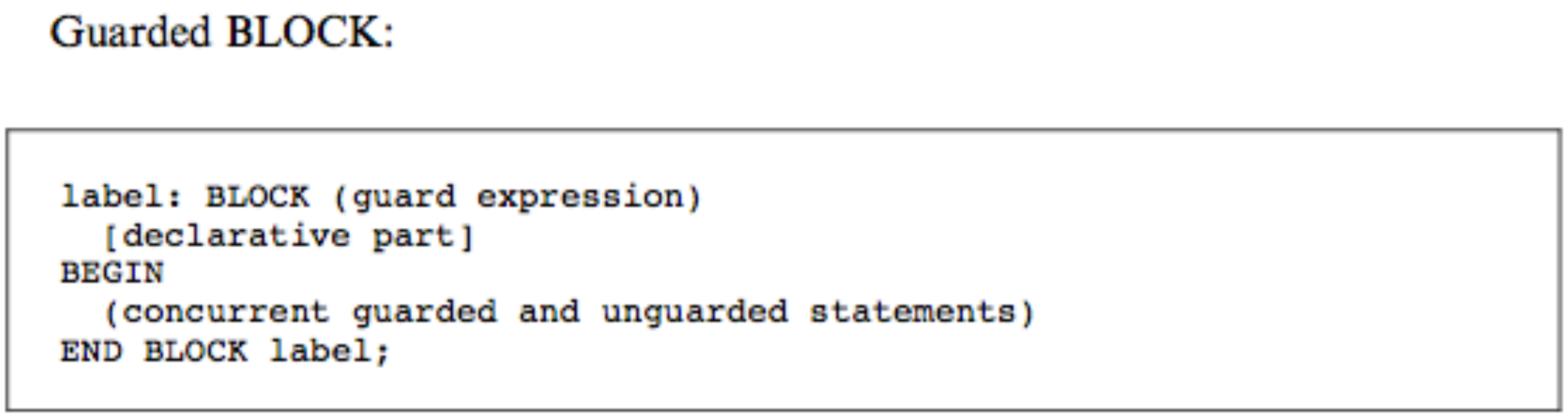
```
------------------------
ARCHITECTURE example ...
BEGIN
   ...
   block1: BLOCK
   BEGIN
      ...
   END BLOCK block1
   ...
   block2: BLOCK
   BEGIN
      ...
   END BLOCK block2;
   ...
END example;
------------------------
```

Example:

```
b1: BLOCK
   SIGNAL a: STD_LOGIC;
BEGIN
   a <= input_sig   WHEN ena='1' ELSE 'Z';
END BLOCK b1;
```

Guarded BLOCK

- A guarded BLOCK is a special kind of BLOCK, which includes an additional expression, called guard expression.
- A guarded statement in a guarded BLOCK is executed only when the guard expression is TRUE.
- Note: Only concurrent statements can be written within a BLOCK. However, with a guarded BLOCK even sequential circuits can be constructed.


Guarded BLOCK:

```
label: BLOCK (guard expression)
  [declarative part]
BEGIN
  (concurrent guarded and unguarded statements)
END BLOCK label;
```

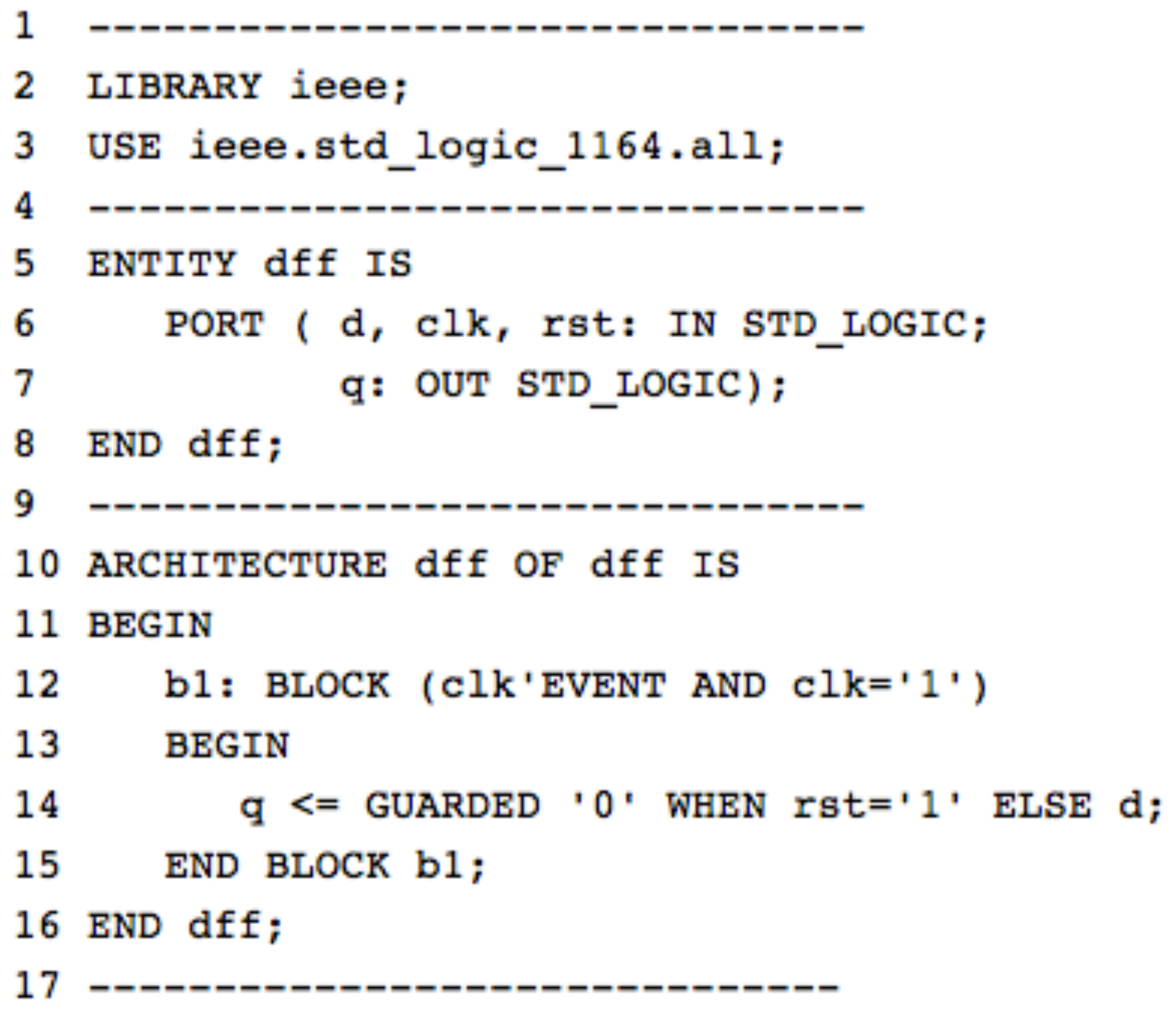
Latch example with BLOCK

- The example presented implements a transparent latch.
- In it, clk='1' (line 12) is the guard expression, while q<=GUARDED d (line 14) is a guarded statement.
- Therefore, q<=d will only occur if clk='1'.

```
1  ------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------
5  ENTITY latch IS
6     PORT (d, clk: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END latch;
9  ------------------------------
10 ARCHITECTURE latch OF latch IS
11 BEGIN
12    b1: BLOCK (clk='1')
13    BEGIN
14       q <= GUARDED d;
15    END BLOCK b1;
16 END latch;
17 ------------------------------
```

DFF Implemented with a Guarded BLOCK

- Positive-edge sensitive D-type flip-flop, with synchronous reset, is designed.
- The clk'EVENT AND clk='1' (line 12) is the guard expression, while q <= GUARDED '0' WHEN rst='1' (line 14) is a guarded statement.
- Therefore, q<='0' will occur when the guard expression is true and rst is '1'.



```
1  -------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -------------------------------
5  ENTITY dff IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8  END dff;
9  -------------------------------
10 ARCHITECTURE dff OF dff IS
11 BEGIN
12    b1: BLOCK (clk'EVENT AND clk='1')
13    BEGIN
14       q <= GUARDED '0' WHEN rst='1' ELSE d;
15    END BLOCK b1;
16 END dff;
17 ------------------------------
```

Small (but fundamental) detour

Variable Scope in C

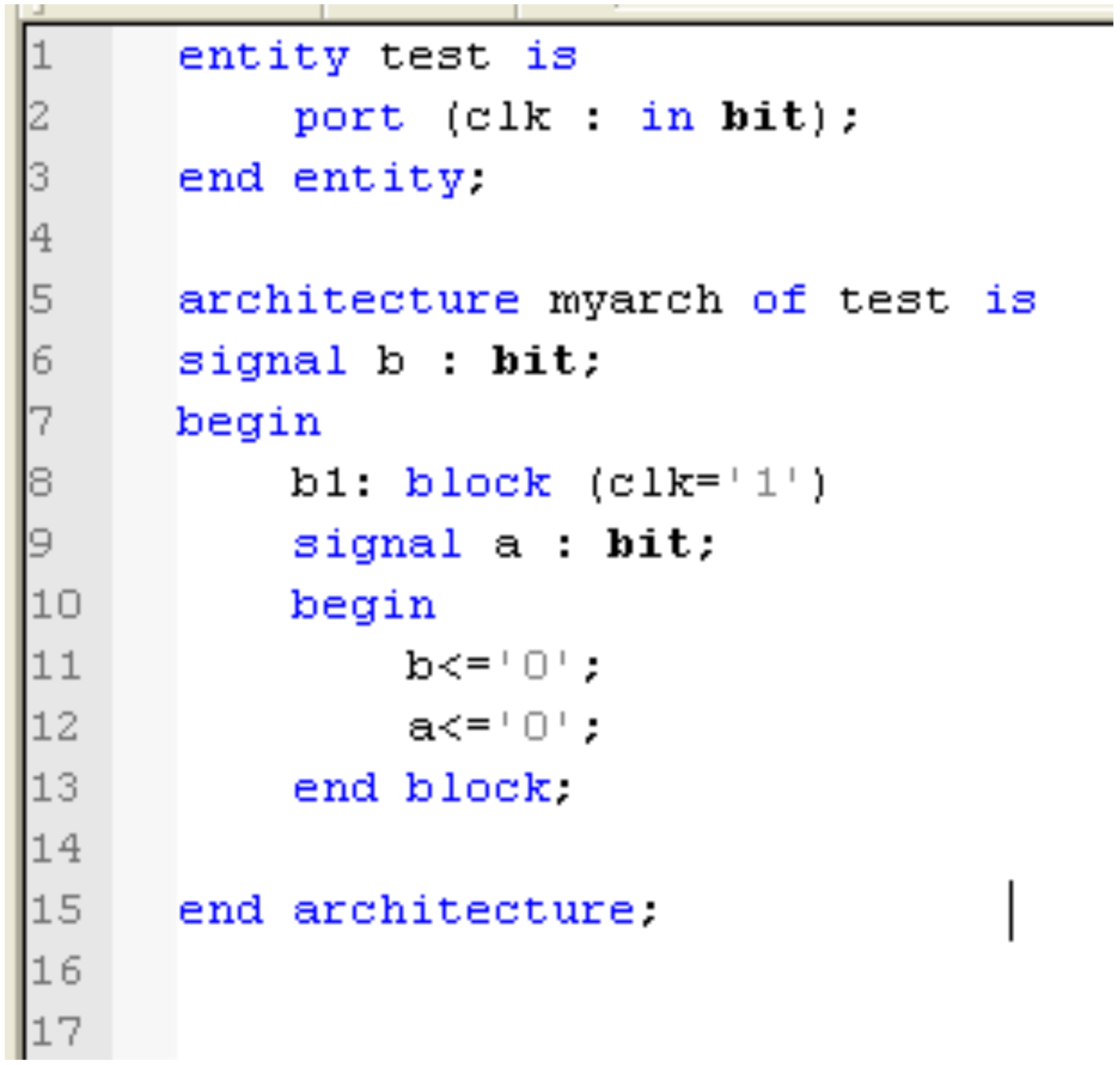
```
float valA=12.5; float valB=16.1;

void print_variable_scope()
{
  int intVariable=10;
  Serial.println(intVariable);

  Serial.println(valA); //valA is global

  //ERROR! Variable res is out of Scope
  Serial.println(res);
}

void loop()
{
  print_variable_scope();
  float res = 12.5;
  Serial.println(res);
  //ERROR! intVariable is out of Scope
  Serial.println(intVariable);
}
```

- The scope of a variable is where it can be used in a program
- Normally variables are local in scope - this means they can only be used in the function where they are declared (main is a function)
- We can also declare global variables.
- If we declare a variable outside a function it can be used in any function beneath where it is declared
- Global variables are A BAD THING... Try to minimize their use.

Signal Scope

- Signal a is only active inside block b1
- In the declaration part of “blocks”, “processes” or “architectures” you can declare signals that are only active **INSIDE** that section of code

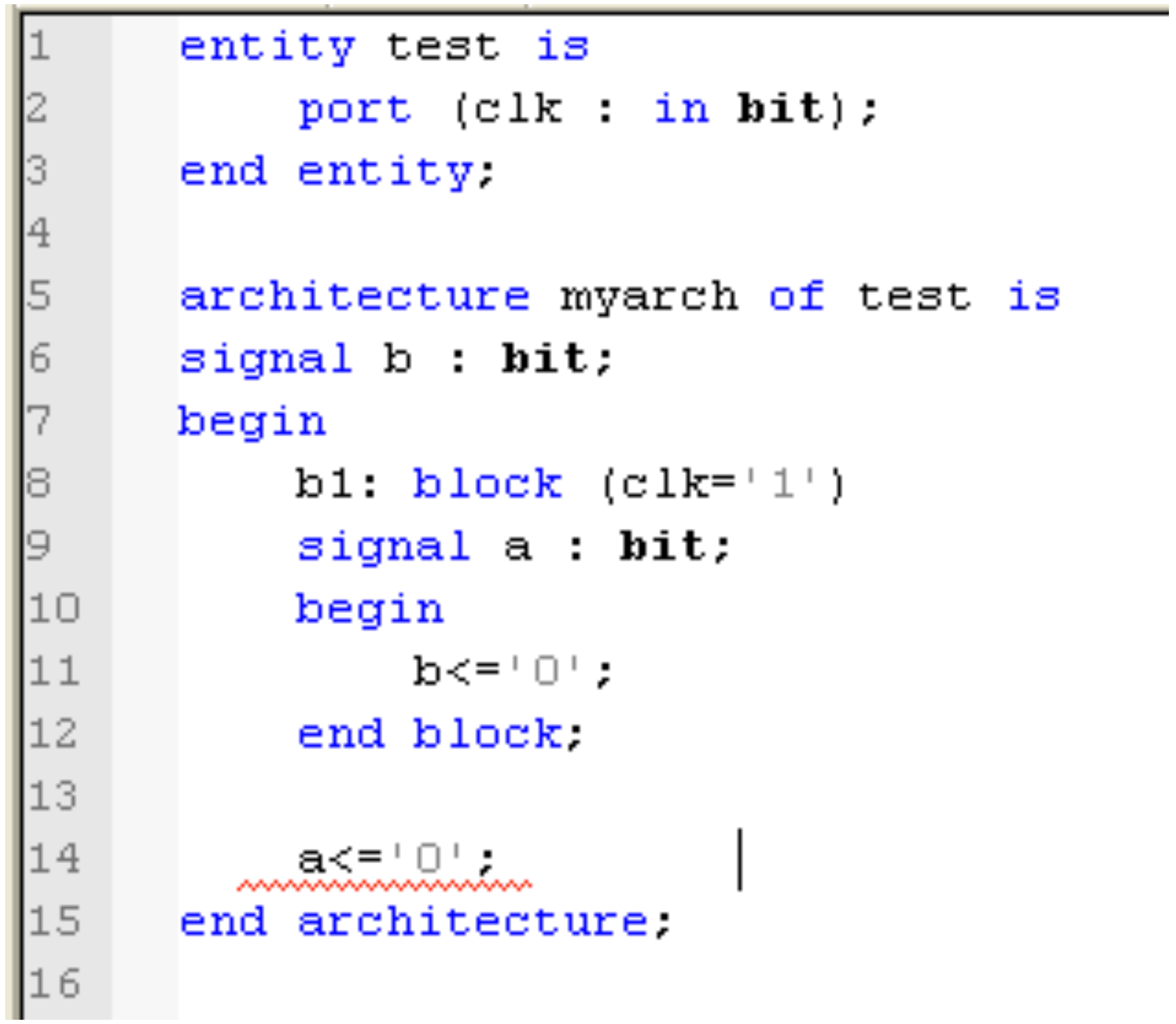
```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```



```
entity test is
    port (clk : in bit);
end entity;

architecture myarch of test is
signal b : bit;
begin
    b1: block (clk='1')
    signal a : bit;
    begin
        b<='0';
        a<='0';
    end block;

end architecture;
```

Signal Scope

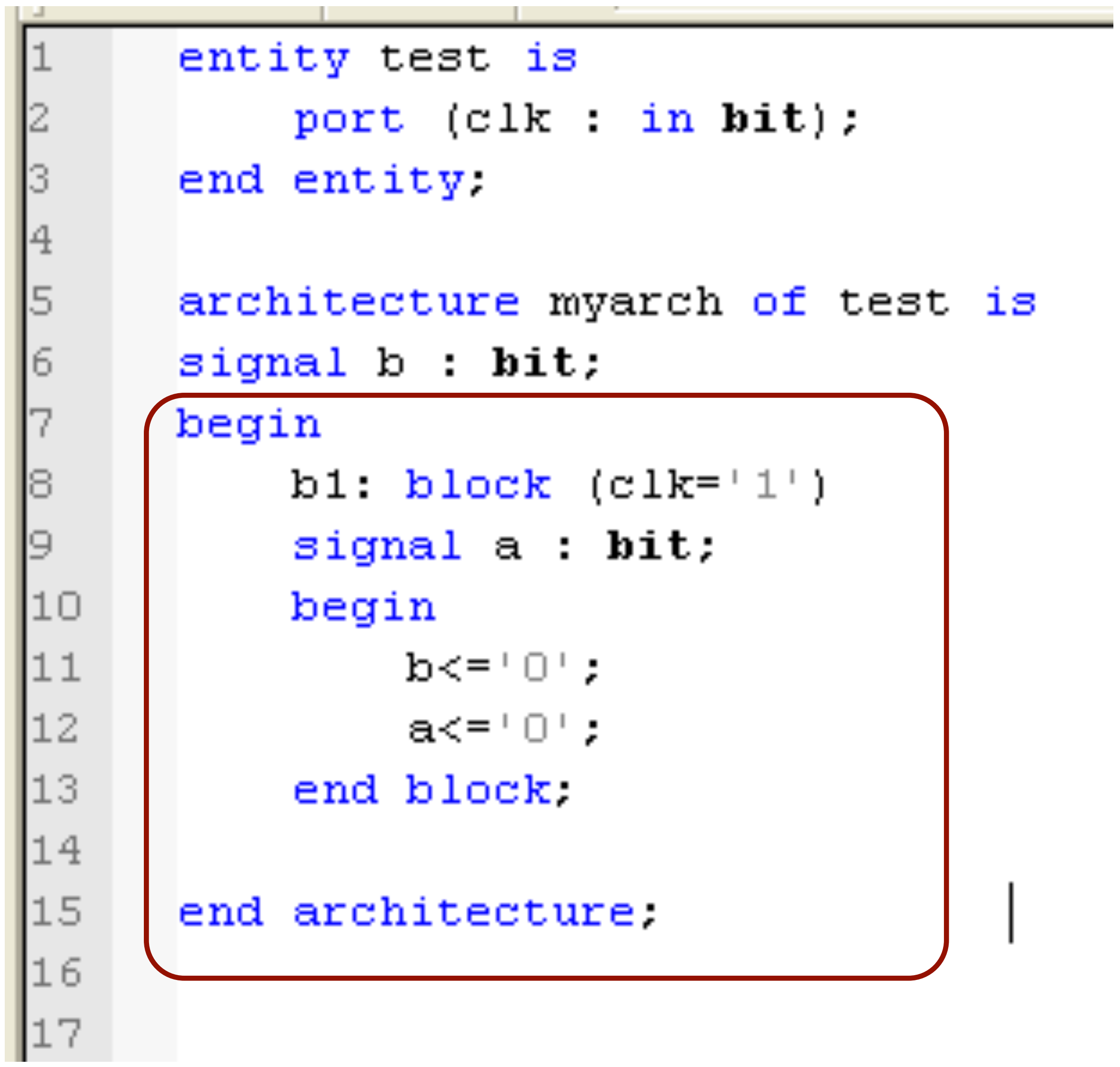
- This is not OK!
- The signal a is not available outside the block b1



```
entity test is
    port (clk : in bit);
end entity;

architecture myarch of test is
signal b : bit;
begin
    b1: block (clk='1')
    signal a : bit;
    begin
        b<='0';
    end block;

    a<='0';
end architecture;

```

Signal Scope

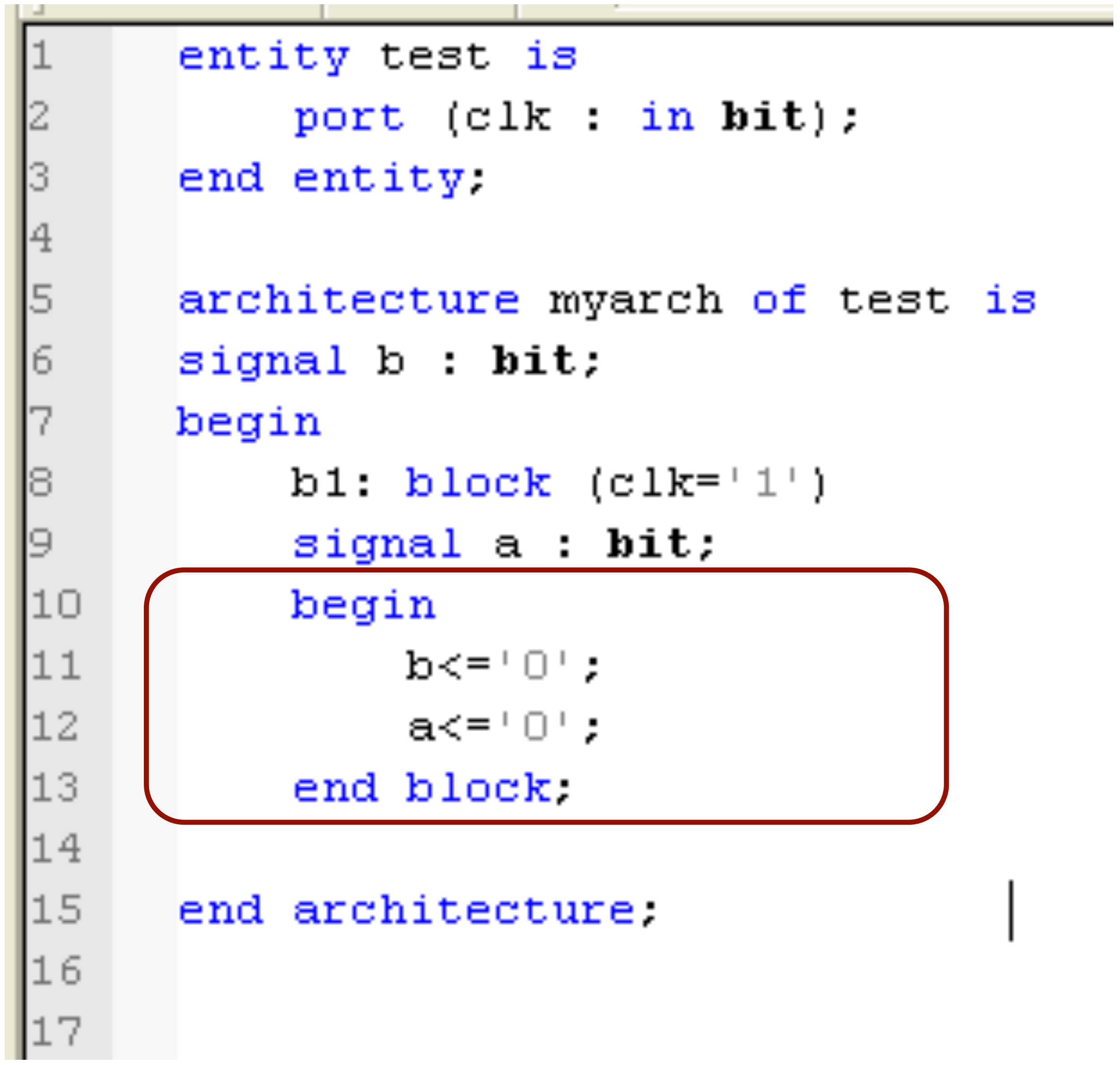
- This is the scope of signal b



```
entity test is
    port (clk : in bit);
end entity;

architecture myarch of test is
signal b : bit;
begin
    b1: block (clk='1')
    signal a : bit;
    begin
        b<='0';
        a<='0';
    end block;

end architecture;
```

Signal Scope

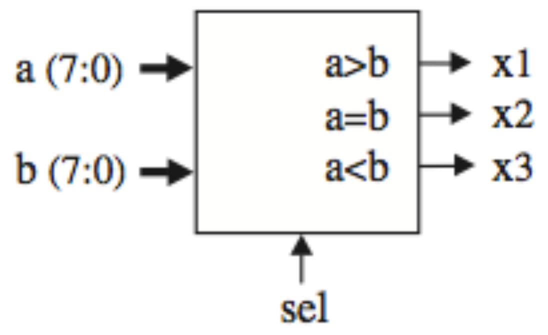
- This is the scope of signal a



```
entity test is
    port (clk : in bit);
end entity;

architecture myarch of test is
signal b : bit;
begin
    b1: block (clk='1')
    signal a : bit;
    begin
        b<='0';
        a<='0';
    end block;

end architecture;
```

Practice Exercises

Comparator



Construct a circuit capable of comparing two 8-bit vectors, a and b. A selection pin (sel) should determine whether the comparison is signed (sel ='1') or unsigned (sel='0'). The circuit must have three outputs, x1, x2, and x3, corresponding to a > b, a=b, and a < b, respectively .