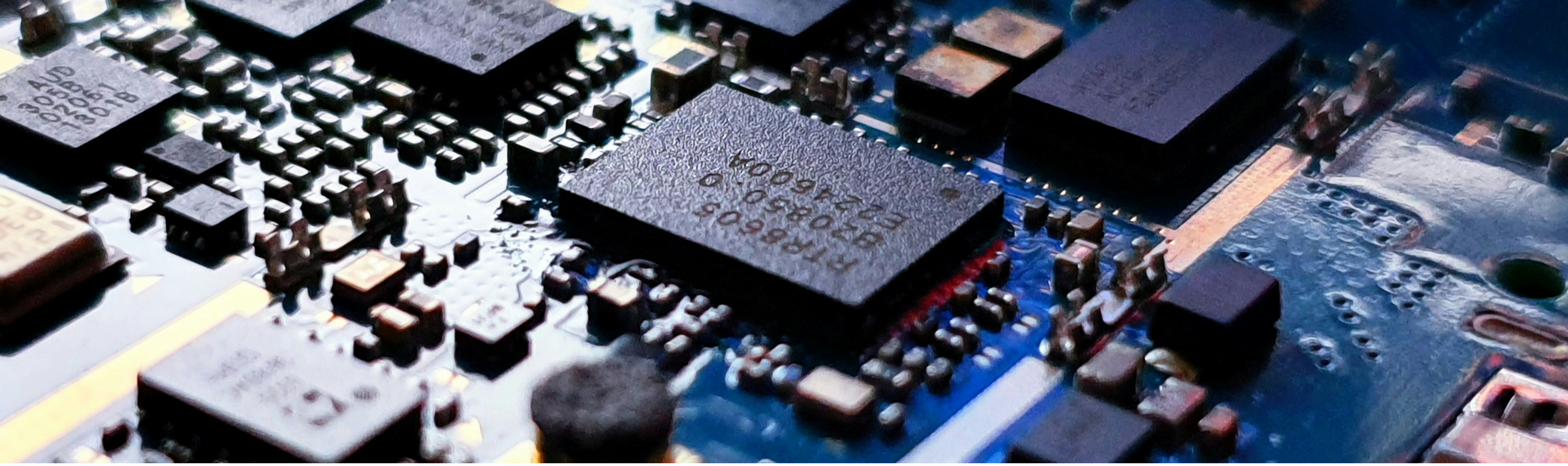


Laboratory 4

Structural & Sequential Systems

Computer Architecture

A.Y. 2023/24

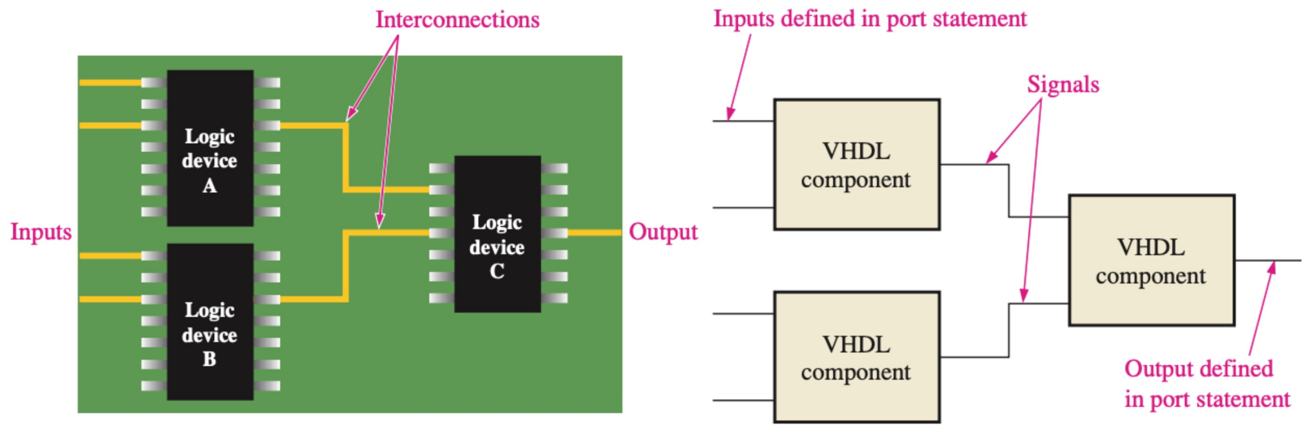
Laboratory goals

- Understand hierarchical and structural approaches to VHDL
- Implement several logic functions in VHDL
- Create a VHDL program to describe flip-flops

Hierarchy & structural analogy

VHDL's structural design is analogous to interconnecting logic devices with wires.

- Top-level entity: can be seen as the whole circuit board.
- VHDL component: represents a reusable logic function
- Signal: wired connection between components.



(a) Hardware implementation with fixed-function logic (b) VHDL structural implementation

Hierarchical design pattern

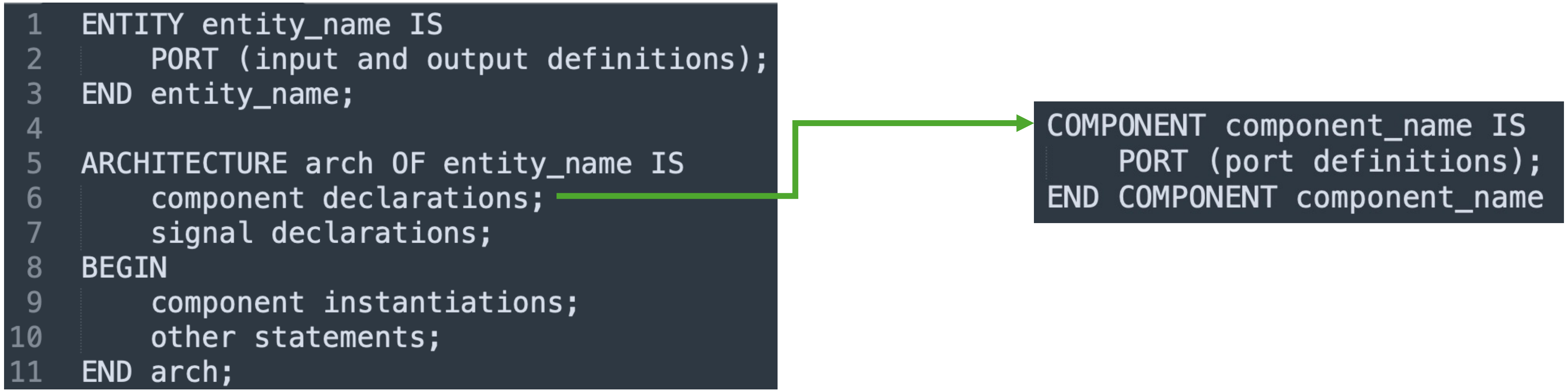
Starting from top-level entity, components can have lower-level components inside them.

- When a component is included in a higher-level component, an instance of it is created.
- Allows for design reutilization and nesting.
- Facilitates group work by having each person work on a different subsystem.

Structural synthesis

In the ARCHITECTURE part of an entity, we can create instances of other components.

Syntax:



```
ENTITY entity_name IS
    PORT (input and output definitions);
END entity_name;

ARCHITECTURE arch OF entity_name IS
    component declarations;
    signal declarations;
BEGIN
    component instantiations;
    other statements;
END arch;
```

```
COMPONENT component_name IS
    PORT (port definitions);
END COMPONENT component_name
```

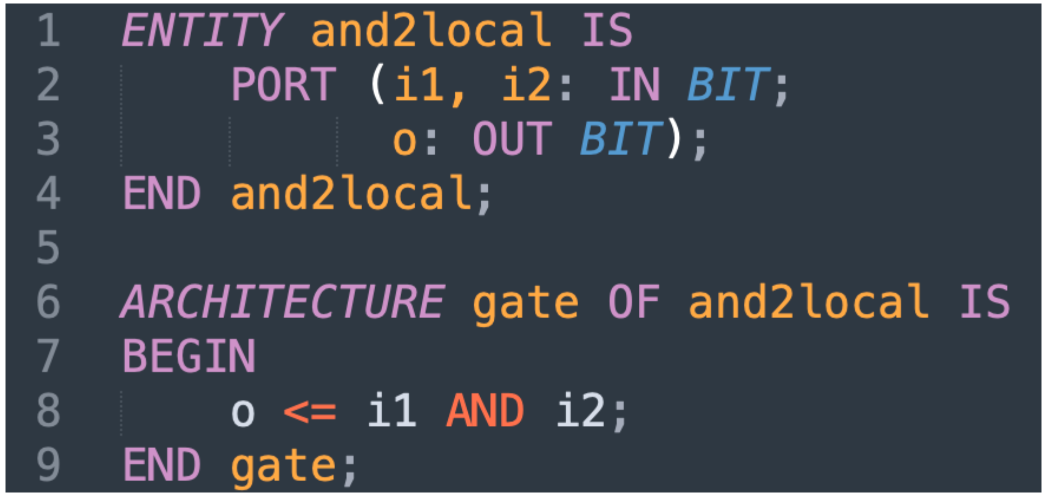
Structural synthesis

A VHDL file can use another VHDL file as a component.

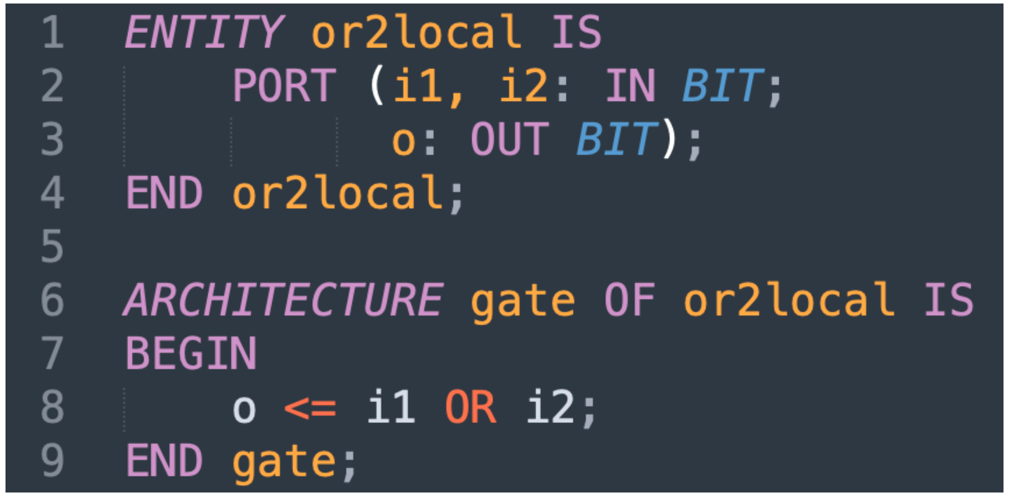
- Component declarations resemble entity declarations.
- Each instance of a component requires an instantiation statement.

Structural synthesis - Example

Suppose we have the following AND and OR gates defined as VHDL entities:



```vhdl
ENTITY and2local IS
    PORT (i1, i2: IN BIT;
          o: OUT BIT);
END and2local;

ARCHITECTURE gate OF and2local IS
BEGIN
    o <= i1 AND i2;
END gate;
```

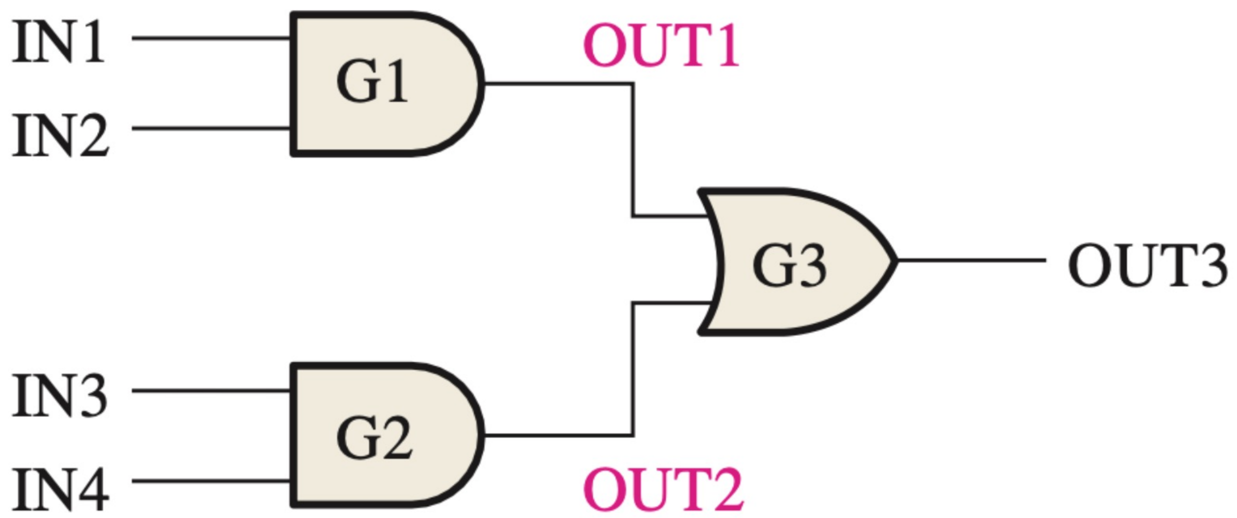


```vhdl
ENTITY or2local IS
    PORT (i1, i2: IN BIT;
          o: OUT BIT);
END or2local;

ARCHITECTURE gate OF or2local IS
BEGIN
    o <= i1 OR i2;
END gate;
```

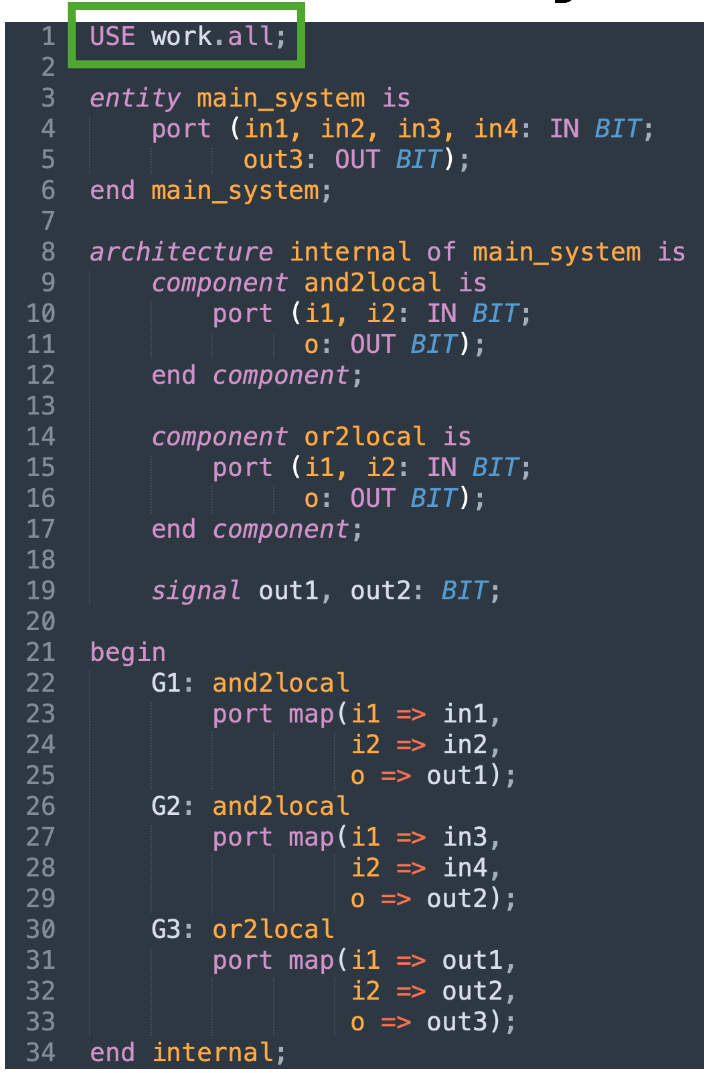
Note: these are for simplicity's sake; you don't need to create entities for primitive logic functions in VHDL!

Structural synthesis – Example 1

We want to reuse these components to create the following circuit:



Structural synthesis – Example 1



```vhdl
USE work.all;

entity main_system is
    port (in1, in2, in3, in4: IN BIT;
          out3: OUT BIT);
end main_system;

architecture internal of main_system is
    component and2local is
        port (i1, i2: IN BIT;
              o: OUT BIT);
    end component;

    component or2local is
        port (i1, i2: IN BIT;
              o: OUT BIT);
    end component;

    signal out1, out2: BIT;

begin
    G1: and2local
        port map(i1 => in1,
                 i2 => in2,
                 o => out1);
    G2: and2local
        port map(i1 => in3,
                 i2 => in4,
                 o => out2);
    G3: or2local
        port map(i1 => out1,
                 i2 => out2,
                 o => out3);
end internal;
```

`USE work.all;` → Use the work library, which allows for component reutilization in VHDL projects.

Component declarations

Component instantiations

Structural synthesis – Example 2

```vhdl
ENTITY FUNCTION1 IS
    PORT( X: IN BIT_VECTOR (2 downto 0);
            z: OUT BIT);
END FUNCTION1;

ARCHITECTURE AND_OR of FUNCTION1 IS

    COMPONENT AND2LOCAL
        PORT (I1, I2: IN BIT;
                O: OUT BIT);
    END COMPONENT;

    COMPONENT OR3LOCAL
        PORT (I1, I2, I3: IN BIT;
                O: OUT BIT);
    END COMPONENT;

    SIGNAL A1, A2, A3: BIT;

    BEGIN

    l1:     AND2LOCAL
            port map(X(0), X(1), A1);

    l2:     AND2LOCAL
            port map(X(0), X(2), A2);

    l3:     AND2LOCAL
            port map(X(1), X(2), A3);

    l4:     OR3LOCAL
            port map(A1, A2, A3, Z);
END AND_OR;
```

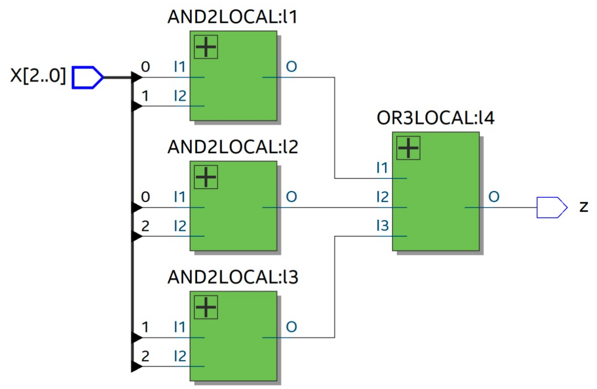
Component declarations

(they refer to these separate files!)

```vhdl
ENTITY AND2LOCAL IS
    PORT(I1, I2: IN BIT;
            O:  OUT BIT);
END AND2LOCAL;
ARCHITECTURE gate OF AND2LOCAL IS
BEGIN
    O <= I1 and I2;
END gate;
```

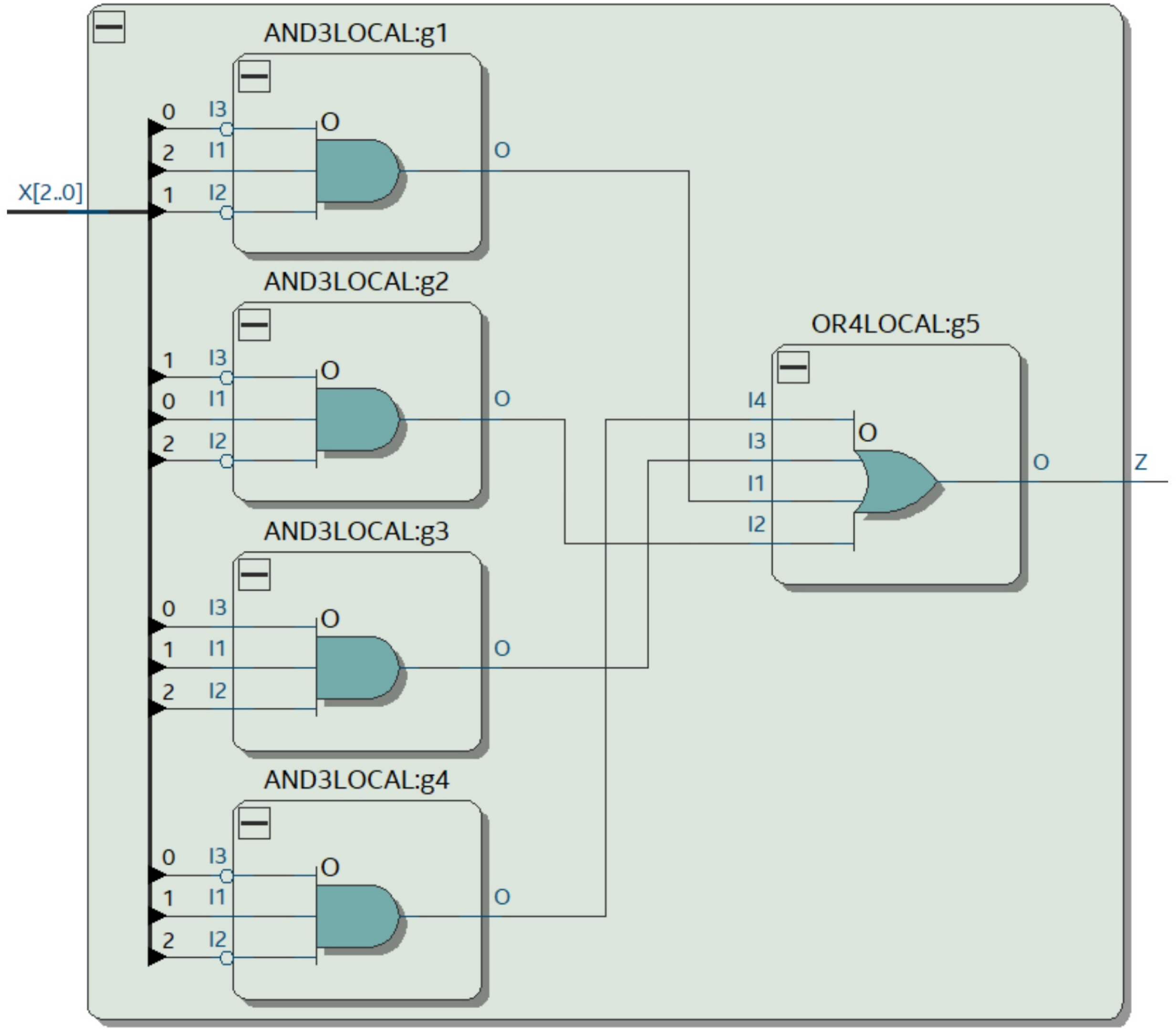
```vhdl
ENTITY OR3LOCAL IS
    PORT(I1, I2, I3: IN BIT;
            O:  OUT BIT);
END OR3LOCAL;
ARCHITECTURE gate OF OR3LOCAL IS
BEGIN
    O <= I1 or I2 or I3;
END gate;
```

Component instantiations



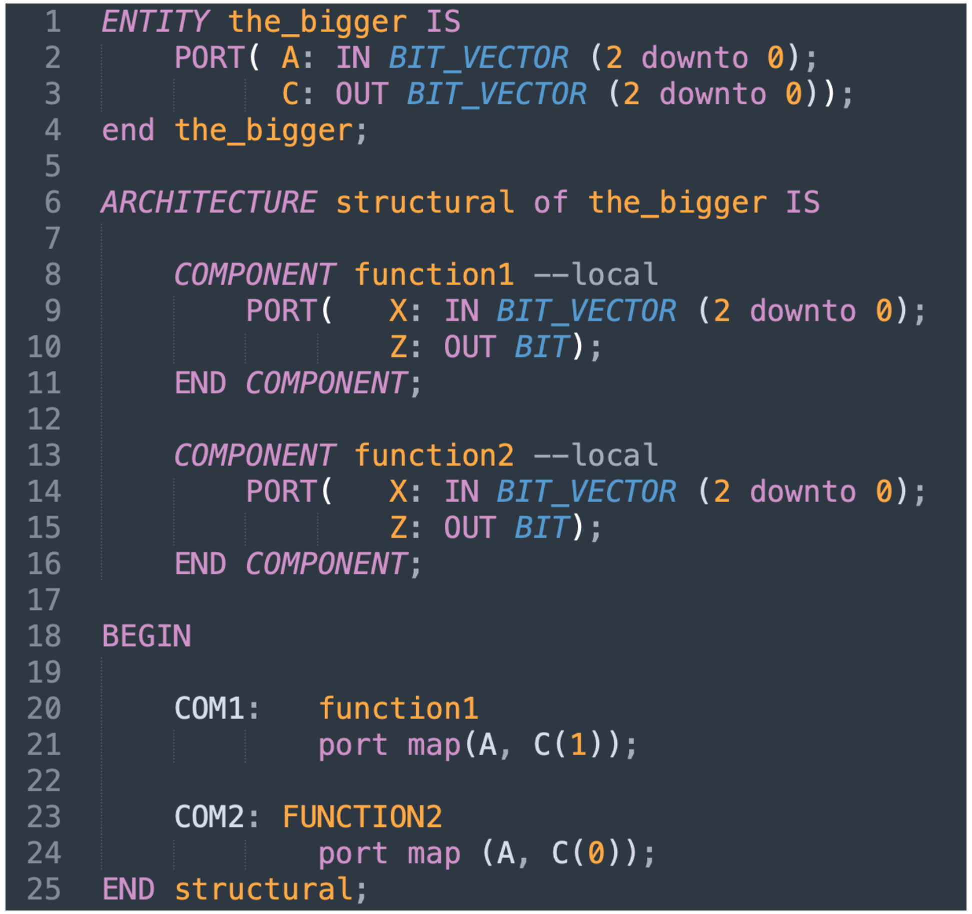
Exercise 1

- Create a new project and add the VHDL files from lab4_struc_examples.zip
- Based on Example 2, implement the component *"function2"* following this schematic;
- Next, use components *function1* and *function2* component to compile the VHDL file "the_bigger.vhd"

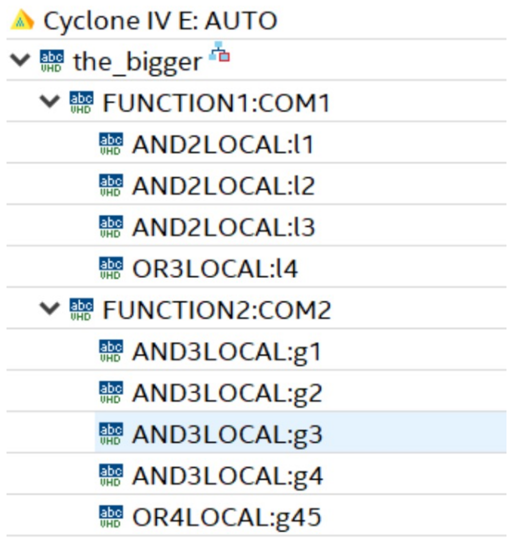


Exercise 1

the_bigger.vhd:



```vhdl
ENTITY the_bigger IS
    PORT( A: IN BIT_VECTOR (2 downto 0);
          C: OUT BIT_VECTOR (2 downto 0));
end the_bigger;

ARCHITECTURE structural of the_bigger IS

    COMPONENT function1 --local
        PORT(   X: IN BIT_VECTOR (2 downto 0);
                Z: OUT BIT);
    END COMPONENT;

    COMPONENT function2 --local
        PORT(   X: IN BIT_VECTOR (2 downto 0);
                Z: OUT BIT);
    END COMPONENT;

BEGIN

    COM1:   function1
            port map(A, C(1));

    COM2: FUNCTION2
            port map (A, C(0));
END structural;
```



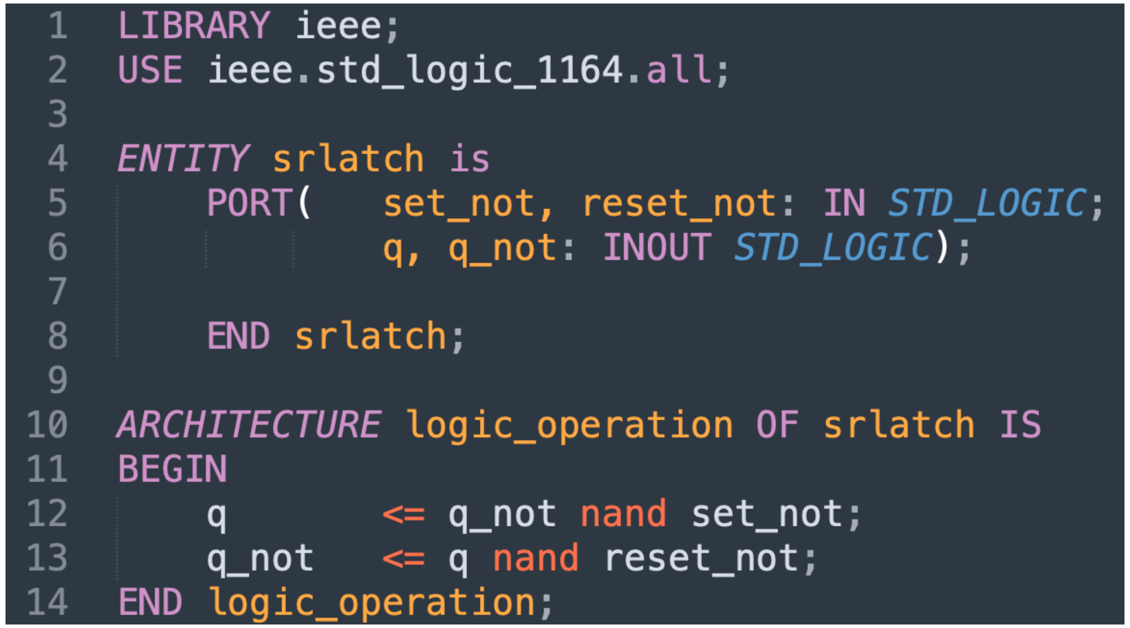
```
Cyclone IV E: AUTO
  the_bigger
    FUNCTION1:COM1
      AND2LOCAL:l1
      AND2LOCAL:l2
      AND2LOCAL:l3
      OR3LOCAL:l4
    FUNCTION2:COM2
      AND3LOCAL:g1
      AND3LOCAL:g2
      AND3LOCAL:g3
      AND3LOCAL:g4
      OR4LOCAL:g45
```

Part 2

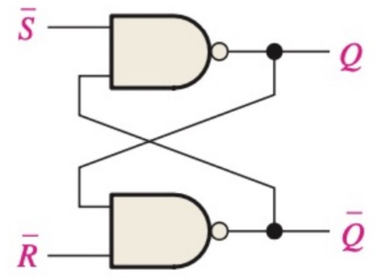
Sequential systems in VHDL

Sequential circuits in VHDL

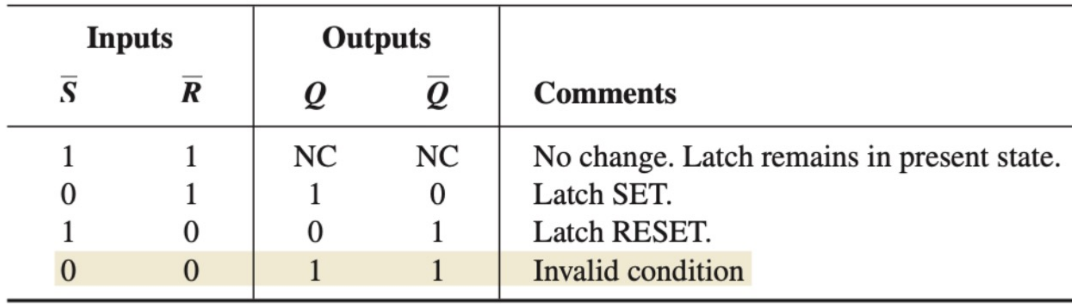
- LATCH ($\overline{SET} - \overline{RESET}$)



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY srlatch is
	PORT(	set_not, reset_not: IN STD_LOGIC;
			q, q_not: INOUT STD_LOGIC);

	END srlatch;

ARCHITECTURE logic_operation OF srlatch IS
BEGIN
	q		<= q_not nand set_not;
	q_not	<= q nand reset_not;
END logic_operation;
```



Truth table for an active-LOW input $\overline{S}$-$\overline{R}$ latch.



Inputs		Outputs		
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	Comments
1	1	NC	NC	No change. Latch remains in present state.
0	1	1	0	Latch SET.
1	0	0	1	Latch RESET.
0	0	1	1	Invalid condition

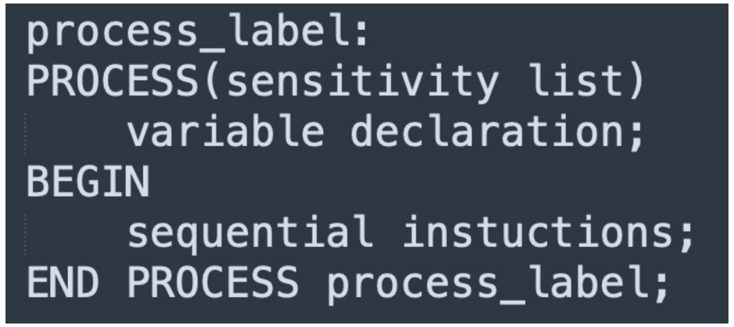
VHDL Processes

A process consists of a set of instructions analyzed sequentially. It behaves as a single instruction, and it is used to perform algorithmic or behavioral descriptions.

It has a declaration section for variables, types, constants and subprograms. The sequential instructions include *if, case, loop, next, assert, wait* statements.

Sensitivity lists: trigger the process.

Syntax:



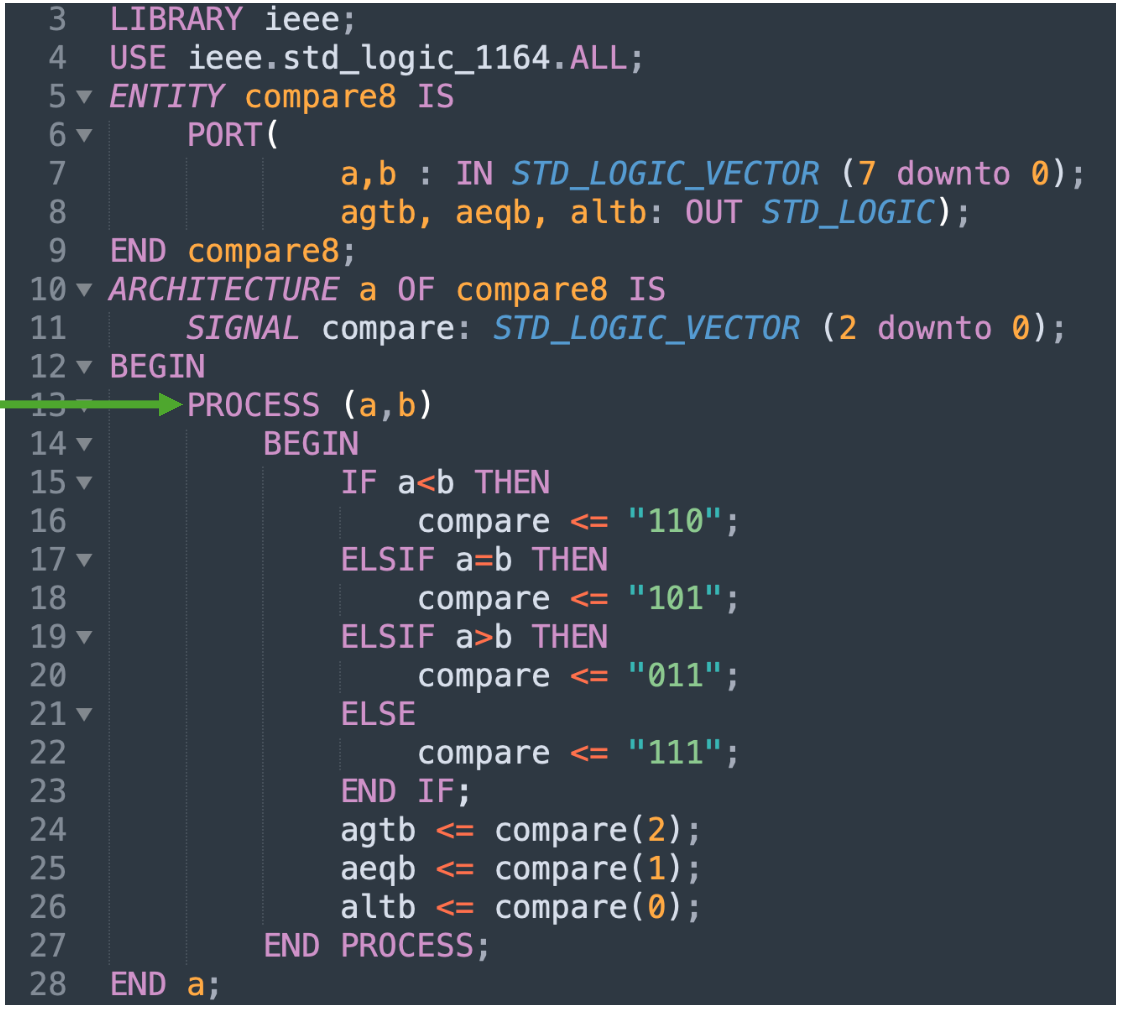
```
process_label:
PROCESS(sensitivity list)
    variable declaration;
BEGIN
    sequential instuctions;
END PROCESS process_label;
```

VHDL Processes

- Sensitivity lists:

They are made up of defined signals that trigger process execution upon change.

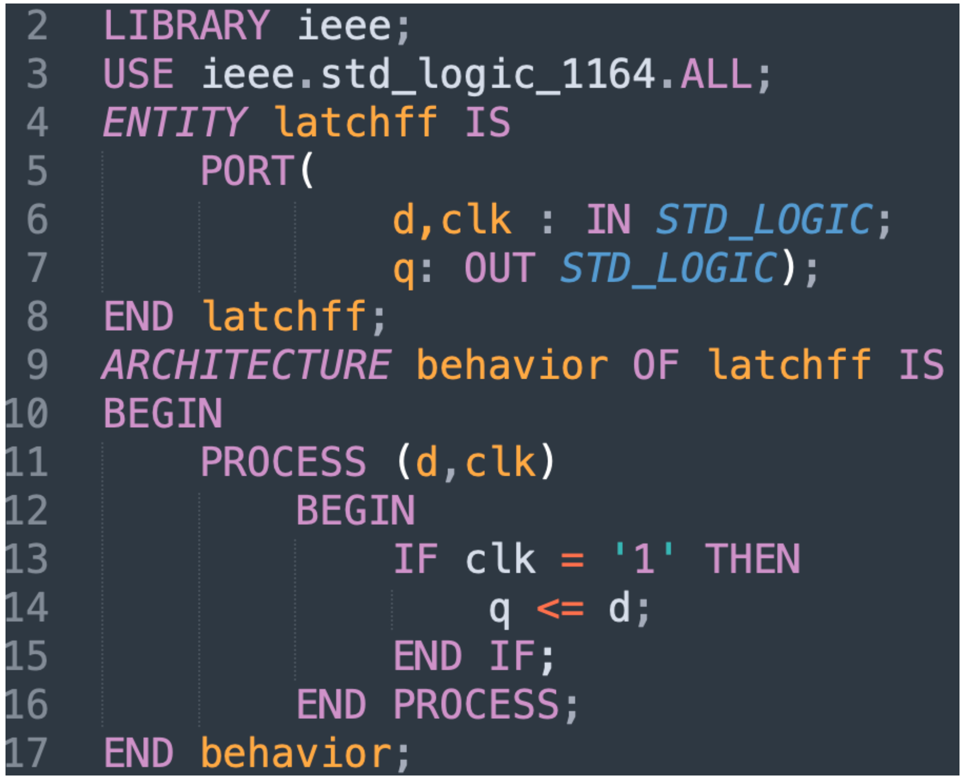
Process will trigger when a or b change.



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY compare8 IS
    PORT(
            a,b : IN STD_LOGIC_VECTOR (7 downto 0);
            agtb, aeqb, altb: OUT STD_LOGIC);
END compare8;
ARCHITECTURE a OF compare8 IS
    SIGNAL compare: STD_LOGIC_VECTOR (2 downto 0);
BEGIN
    PROCESS (a,b)
        BEGIN
            IF a<b THEN
                compare <= "110";
            ELSIF a=b THEN
                compare <= "101";
            ELSIF a>b THEN
                compare <= "011";
            ELSE
                compare <= "111";
            END IF;
            agtb <= compare(2);
            aeqb <= compare(1);
            altb <= compare(0);
        END PROCESS;
END a;
```

Sequential circuits in VHDL

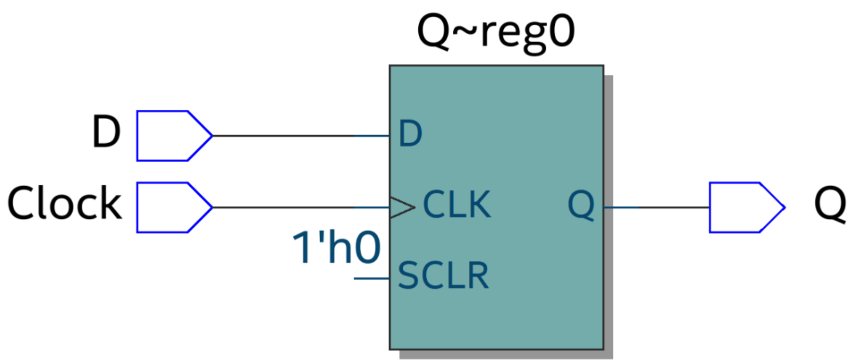
- LATCH-D

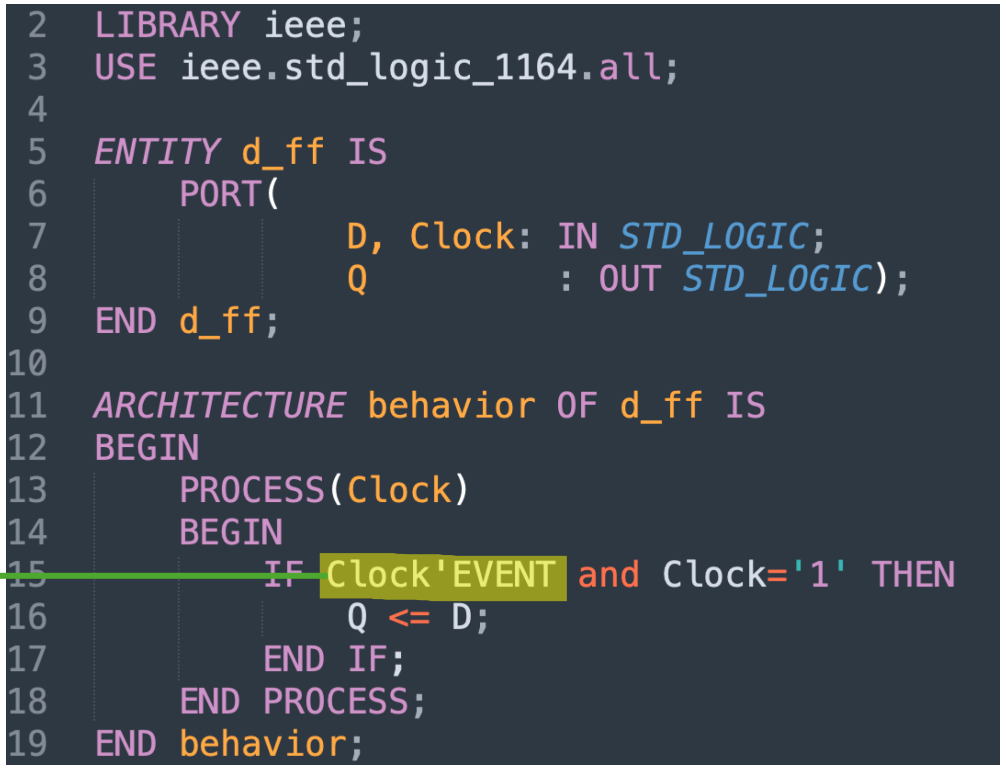


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY latchff IS
    PORT(
            d,clk : IN STD_LOGIC;
            q: OUT STD_LOGIC);
END latchff;
ARCHITECTURE behavior OF latchff IS
BEGIN
    PROCESS (d,clk)
        BEGIN
            IF clk = '1' THEN
                q <= d;
            END IF;
        END PROCESS;
END behavior;
```

Sequential circuits in VHDL

- D-Flipflop (rising edge).





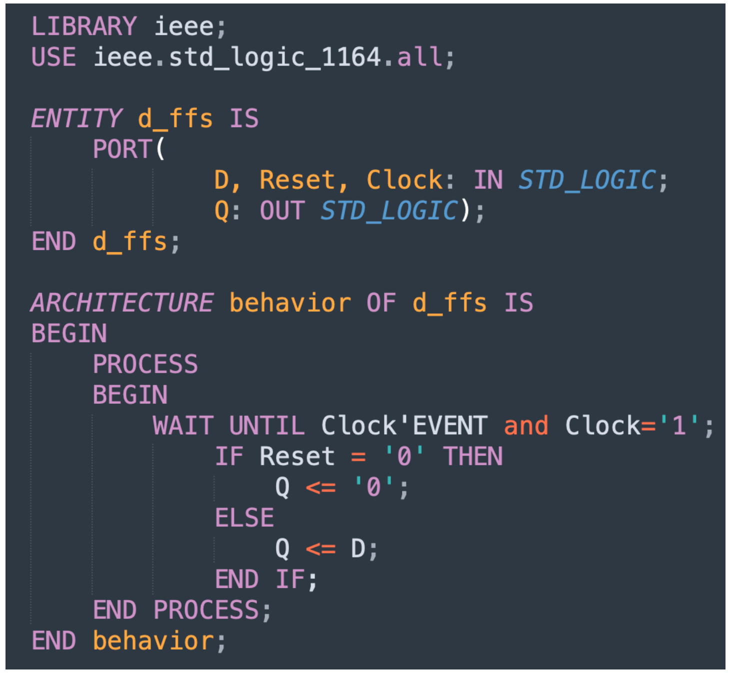
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ff IS
    PORT(
            D, Clock: IN STD_LOGIC;
            Q        : OUT STD_LOGIC);
END d_ff;

ARCHITECTURE behavior OF d_ff IS
BEGIN
    PROCESS(Clock)
    BEGIN
        IF Clock'EVENT and Clock='1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END behavior;
```

Tells the compiler to check on the rising edge of the clock signal.

Sequential circuits in VHDL

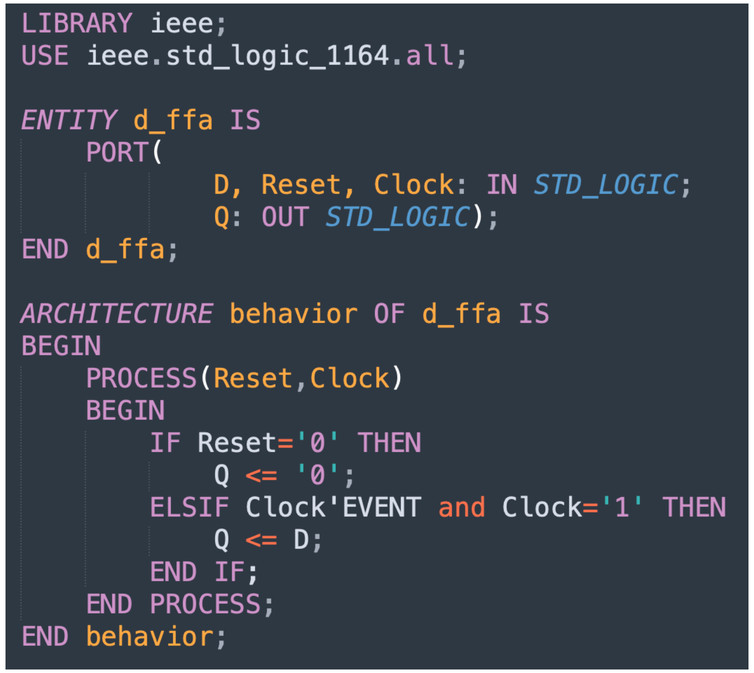
- D-Flipflop with reset (rising edge).

Synchronous



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ffs IS
    PORT(
            D, Reset, Clock: IN STD_LOGIC;
            Q: OUT STD_LOGIC);
END d_ffs;

ARCHITECTURE behavior OF d_ffs IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT and Clock='1';
            IF Reset = '0' THEN
                Q <= '0';
            ELSE
                Q <= D;
            END IF;
    END PROCESS;
END behavior;
```

Asynchronous



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_ffa IS
    PORT(
            D, Reset, Clock: IN STD_LOGIC;
            Q: OUT STD_LOGIC);
END d_ffa;

ARCHITECTURE behavior OF d_ffa IS
BEGIN
    PROCESS(Reset,Clock)
    BEGIN
        IF Reset='0' THEN
            Q <= '0';
        ELSIF Clock'EVENT and Clock='1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END behavior;
```

VHDL CASE statement

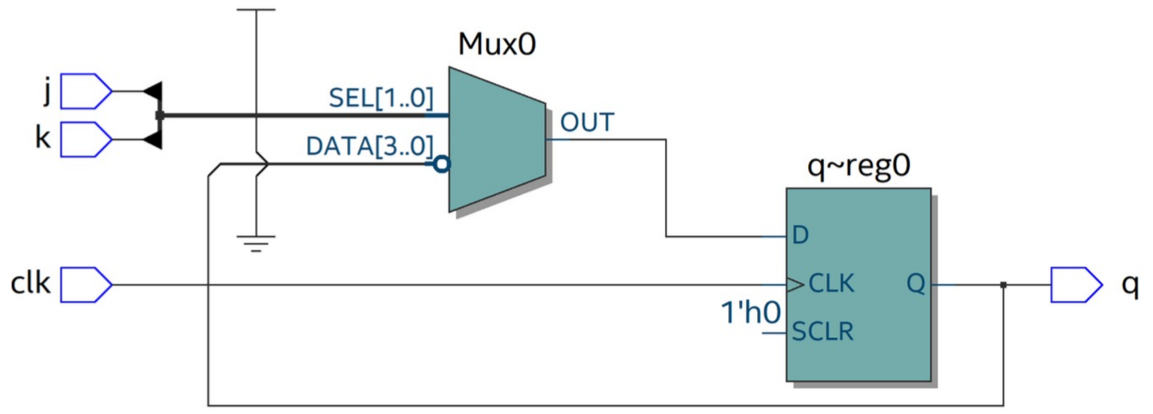
The CASE statement is used to execute one of several instruction, based on a signal’s value.

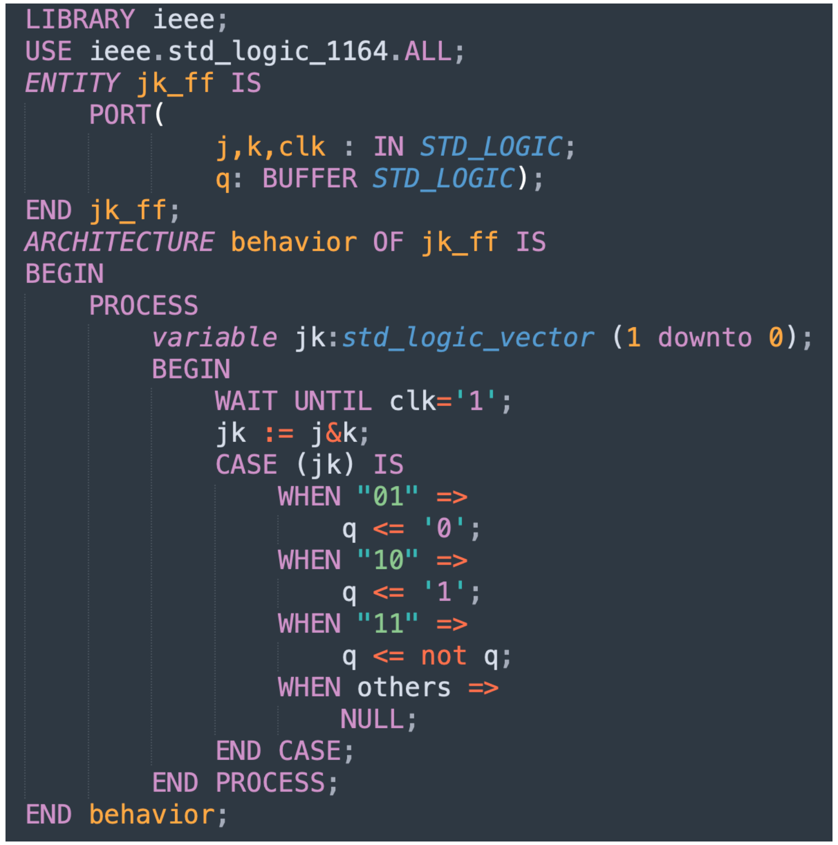
```
CASE __expression IS
WHEN __constant_value =>
__statement;
__statement;
WHEN __constant_value =>
__statement;
__statement;
WHEN OTHERS =>
__statement;
__statement;
END CASE;
```

```
CASE s IS
WHEN “00” =>
y <= “0001”;
x <= “1110”;
WHEN “01” =>
y => “0010”;
x => “1101”;
WHEN “10” =>
y <= “0100”;
x <= “1011”;
WHEN “11” =>
y <= “1000”;
x <= “0111”;
WHEN others =>
y <= “0000”;
x <= “1111”;
END CASE;
```

Sequential circuits in VHDL

- J-K Flipflop (rising edge)



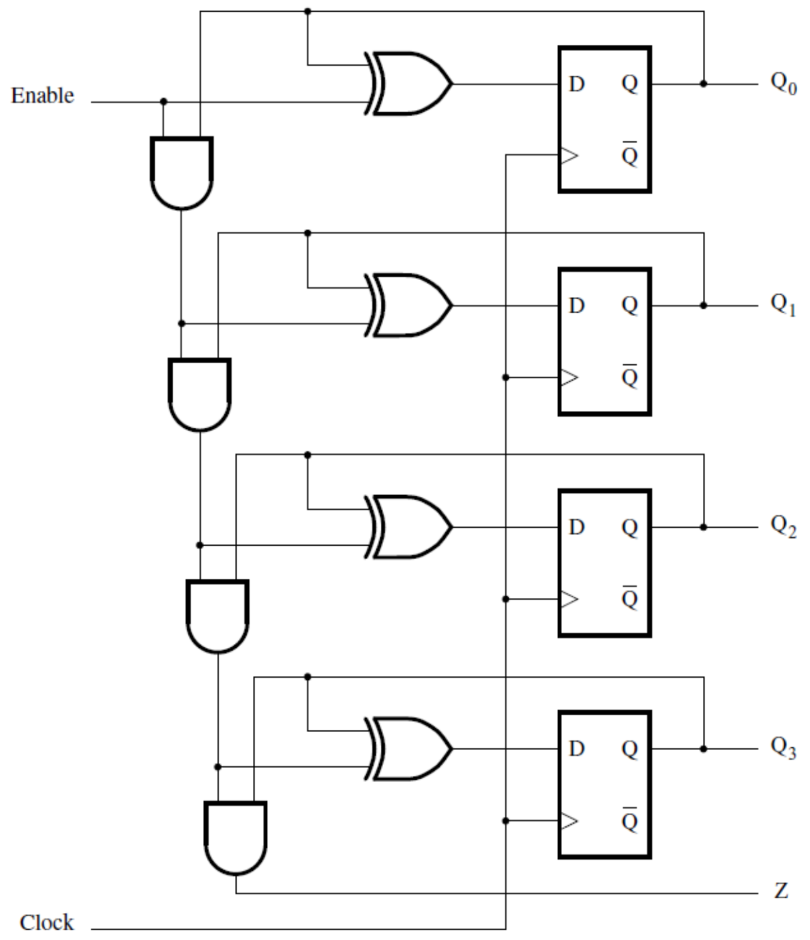


```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY jk_ff IS
    PORT(
            j,k,clk : IN STD_LOGIC;
            q: BUFFER STD_LOGIC);
END jk_ff;
ARCHITECTURE behavior OF jk_ff IS
BEGIN
    PROCESS
        variable jk:std_logic_vector (1 downto 0);
        BEGIN
            WAIT UNTIL clk='1';
            jk := j&k;
            CASE (jk) IS
                WHEN "01" =>
                    q <= '0';
                WHEN "10" =>
                    q <= '1';
                WHEN "11" =>
                    q <= not q;
                WHEN others =>
                    NULL;
            END CASE;
        END PROCESS;
END behavior;
```

Exercise 2

- Create a new project and add the VHDL files from lab4_seq_examples.zip
- Create a **4-bit Up-Counter**. Use flip-flops as components, and only use PROCESS constructs.
- You are free to use any flip-flop from the example files.

Exercise 2 – Sample using D-flipflops



Source: https://home.engineering.iastate.edu/alexs/classes/2018_Fall_281/slides_PDF/31_Solved_Problems.pdf

References

- Tokheim, R., *Digital Electronics: Principles And Applications*.
- Floyd, Thomas L., *Digital Fundamentals*.
- Mano, Morris. *Digital Design*.
- Intel QUARTUS Help Manuals
- Tocci, Ronald. *Digital Systems Principles and Applications*.