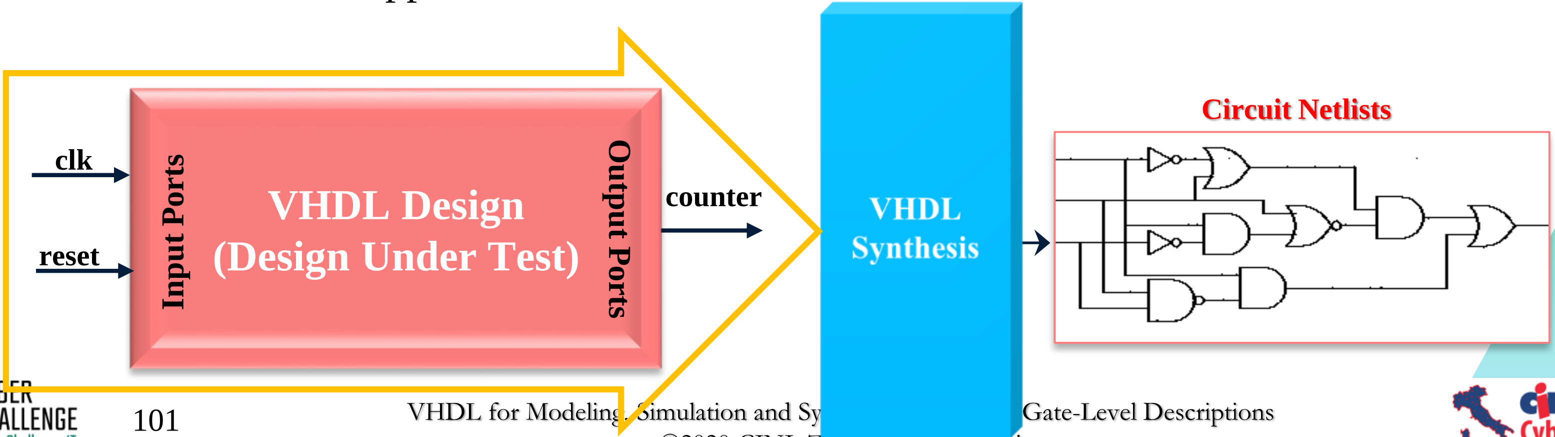
- Synthesis Concepts
- Combinational Rules
- Sequential Rules
- Combination of Sequential and Combinational Blocks
- ALU as an Example
- RT Level of ALU
- Circuit Netlists
- Technology Map of ALU

7- Synthesis Issues

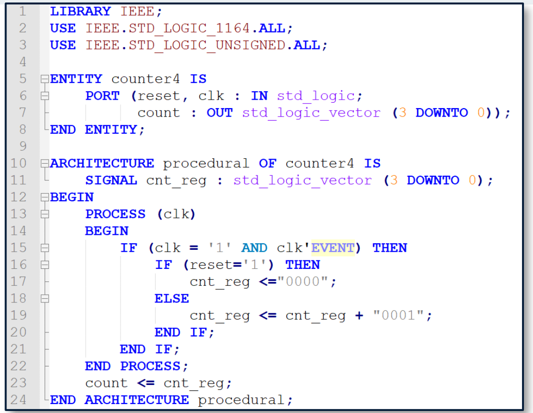
Synthesis Concepts

- Synthesis is the process of taking some higher level description down to an immediate next lower level description.
- For example - taking VHDL code and producing a netlist that can be mapped to an FPGA or an ASIC.

- Will discuss style of synthesis



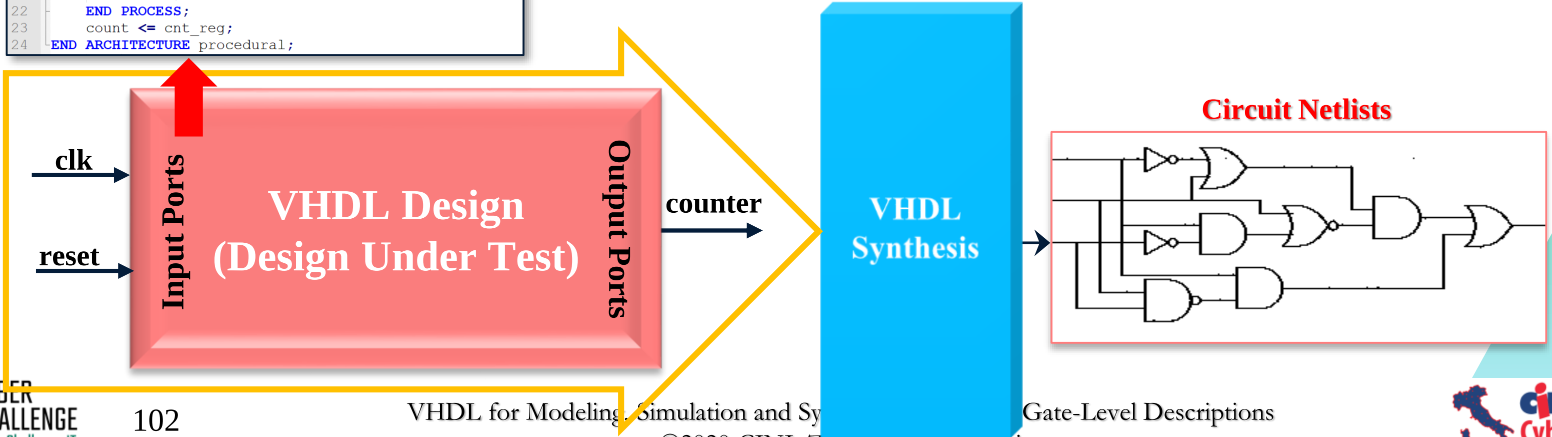
Synthesis Concepts



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY counter4 IS
    PORT (reset, clk : IN std_logic;
          count : OUT std_logic_vector (3 DOWNTO 0));
END ENTITY;

ARCHITECTURE procedural OF counter4 IS
    SIGNAL cnt_reg : std_logic_vector (3 DOWNTO 0);
BEGIN
    PROCESS (clk)
    BEGIN
        IF (clk = '1' AND clk'EVENT) THEN
            IF (reset='1') THEN
                cnt_reg <="0000";
            ELSE
                cnt_reg <= cnt_reg + "0001";
            END IF;
        END IF;
    END PROCESS;
    count <= cnt_reg;
END ARCHITECTURE procedural;
```

VHDL Description

- A synthesis tool requires a target library
- A target library can be as simple as a library of logic gates



Synthesis Issues

- A VHDL description may include combinational and sequential blocks.
- Combinational blocks will be synthesizable using process or concurrent statements.
- Sequential blocks will be synthesizable using process statements sensitive to clock.

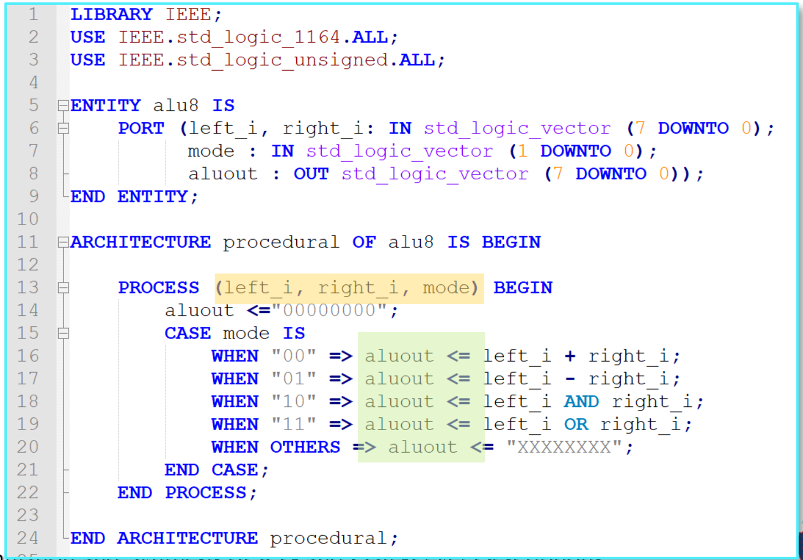
- Will discuss combinational and sequential rules
- Combinational circuit synthesis:
  - Signal assignments are easier for synthesis, not too many rules to follow
  - For process statements there are rules to follow

Combinational Rules

1. Input: All signals read must be in the sensitivity list.
2. Output: All outputs must receive a value in the process statement.

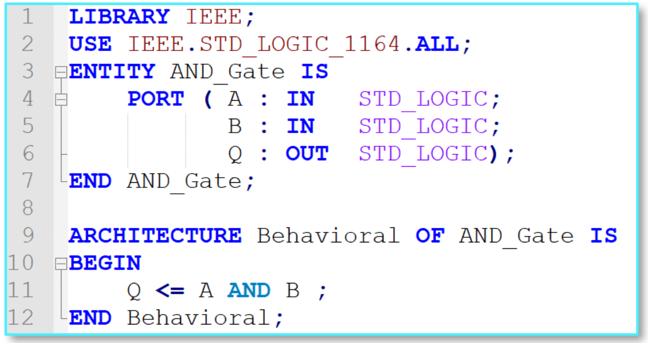
- Input / Output rules

- Procedural ALU



```
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
        aluout <="00000000";
        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;
```

- AND



```
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;
```

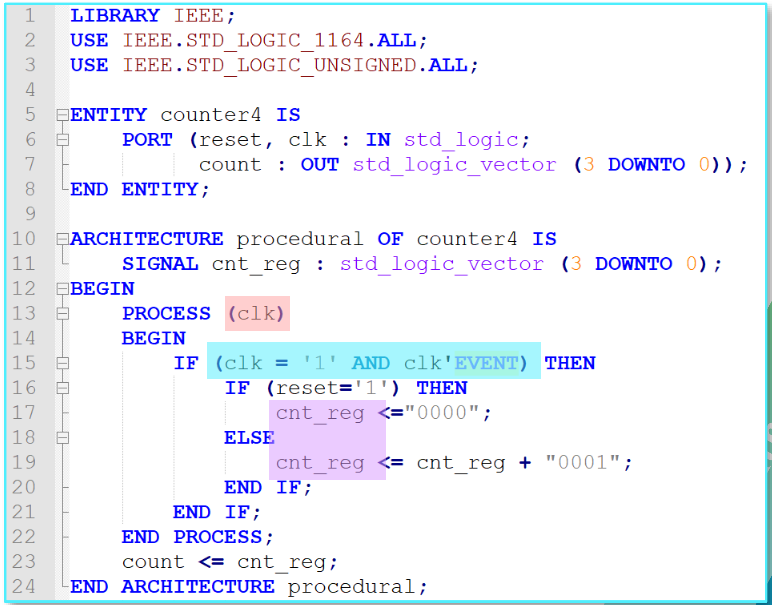
7- Synthesis Issues

Sequential Rules

- Sequential circuit synthesis:
  - Following simple rules keep you safe

1. A clocked process must have the clock and all asynchronous controls.
2. Synchronous register activities must be conditioned by a statement detecting clock edge.
3. For the beginners, limit the register left-hand-side to only one register.

- Counter

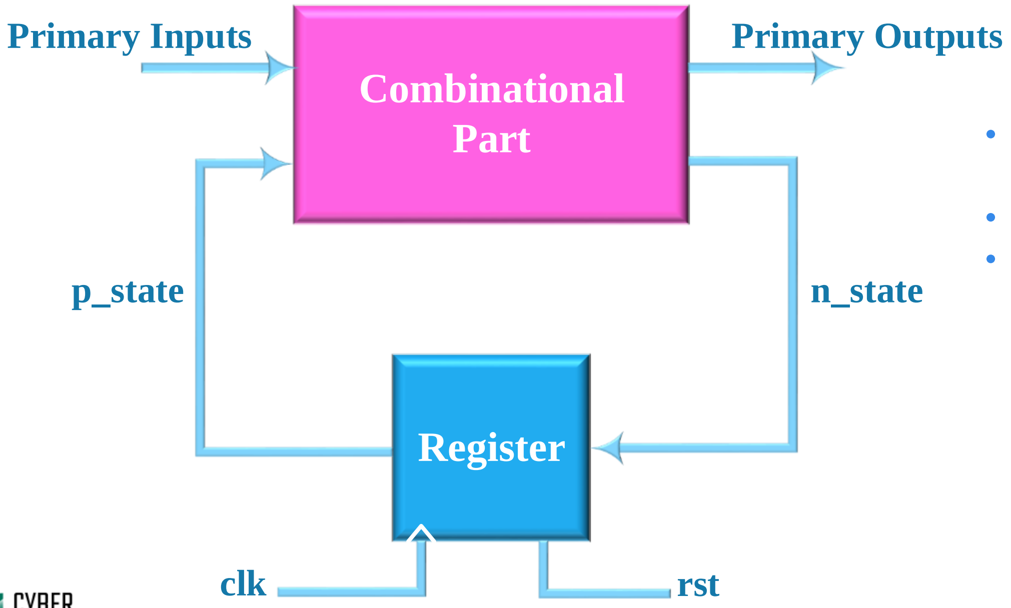


```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY counter4 IS
    PORT (reset, clk : IN std_logic;
          count : OUT std_logic_vector (3 DOWNTO 0));
END ENTITY;

ARCHITECTURE procedural OF counter4 IS
    SIGNAL cnt_reg : std_logic_vector (3 DOWNTO 0);
BEGIN
    PROCESS (clk)
    BEGIN
        IF (clk = '1' AND clk'EVENT) THEN
            IF (reset='1') THEN
                cnt_reg <="0000";
            ELSE
                cnt_reg <= cnt_reg + "0001";
            END IF;
        END IF;
    END PROCESS;
    count <= cnt_reg;
END ARCHITECTURE procedural;
```

- Sequential blocks will be synthesizable using process statements which are sensitive to clock.

Combination of Sequential and Combinational Blocks

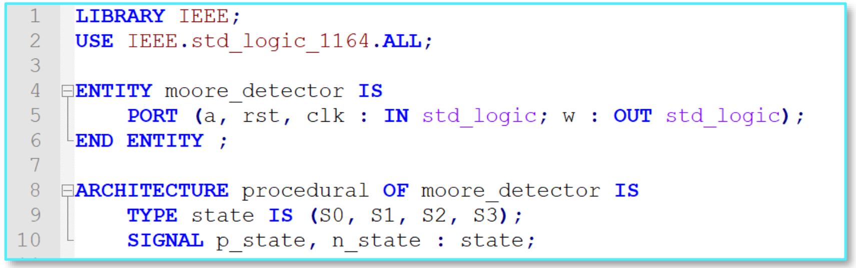
- Huffman Model



- Any complete digital circuit can be *split* into combinational part and register (sequential) part
- Use combinational rules for the combinational part
- Use sequential rules for the sequential part

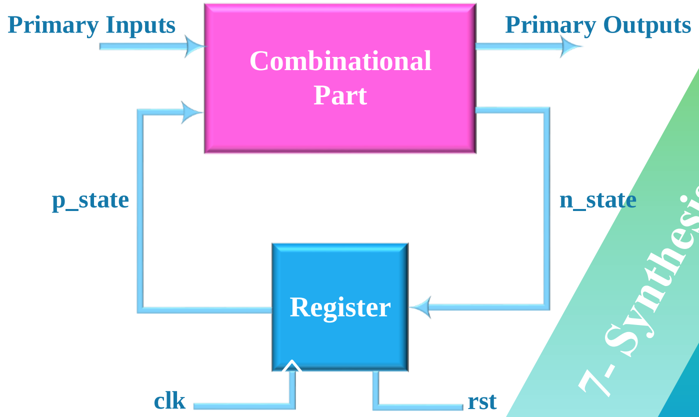
Combination of Sequential and Combinational Blocks

- Huffman Model



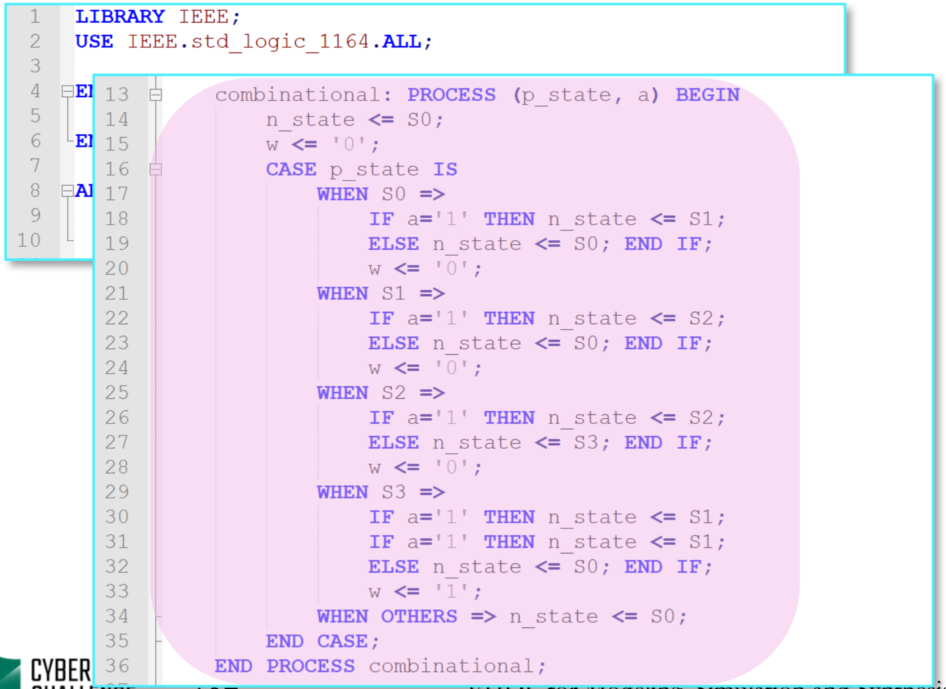
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY moore_detector IS
    PORT (a, rst, clk : IN std_logic; w : OUT std_logic);
END ENTITY ;

ARCHITECTURE procedural OF moore_detector IS
    TYPE state IS (S0, S1, S2, S3);
    SIGNAL p_state, n_state : state;
```

- Combinational and sequential parts are connected by p_state and n_state of type state
- Combinational part uses p_state assigned to n_state
- Sequential part clocks n_state to p_state
- A synthesizable description can contain any number of concurrent combinational and/or sequential processes



Combination of Sequential and Combinational Blocks

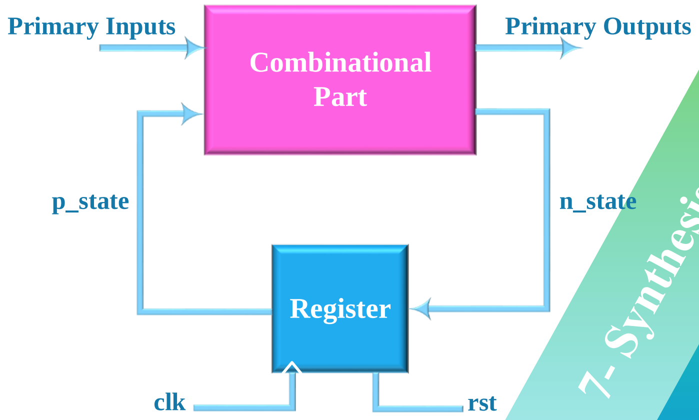
- Huffman Model



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

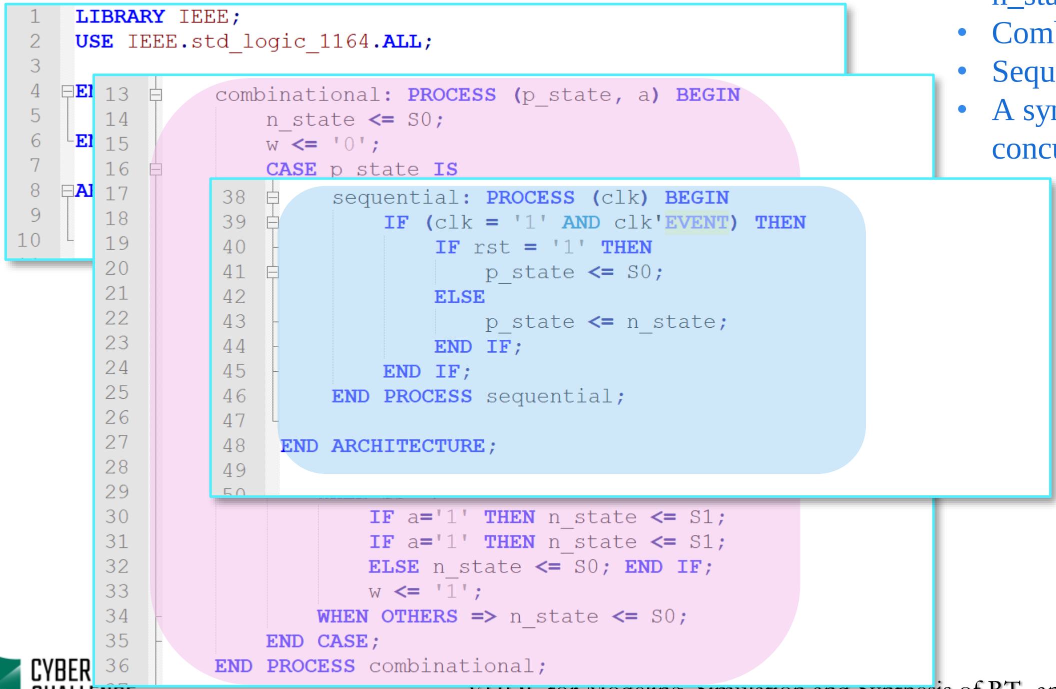
```vhdl
combinational: PROCESS (p_state, a) BEGIN
    n_state <= S0;
    w <= '0';
    CASE p_state IS
        WHEN S0 =>
            IF a='1' THEN n_state <= S1;
            ELSE n_state <= S0; END IF;
            w <= '0';
        WHEN S1 =>
            IF a='1' THEN n_state <= S2;
            ELSE n_state <= S0; END IF;
            w <= '0';
        WHEN S2 =>
            IF a='1' THEN n_state <= S2;
            ELSE n_state <= S3; END IF;
            w <= '0';
        WHEN S3 =>
            IF a='1' THEN n_state <= S1;
            IF a='1' THEN n_state <= S1;
            ELSE n_state <= S0; END IF;
            w <= '1';
        WHEN OTHERS => n_state <= S0;
    END CASE;
END PROCESS combinational;
```

- Combinational and sequential parts are connected by p_state and n_state of type state
- Combinational part uses p_state assigned to n_state
- Sequential part clocks n_state to p_state
- A synthesizable description can contain any number of concurrent combinational and/or sequential processes



Combination of Sequential and Combinational Blocks

- Huffman Model



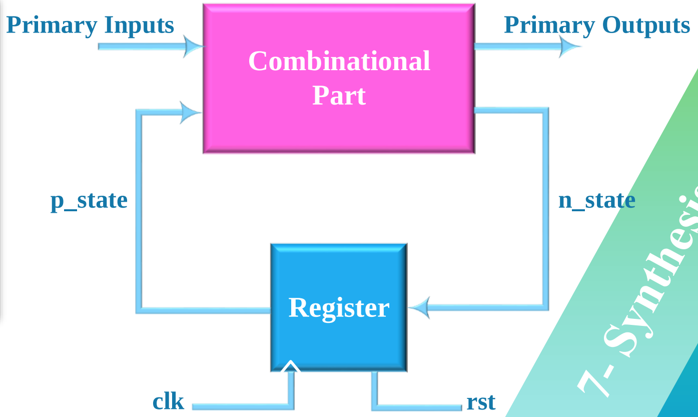
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

```
    combinational: PROCESS (p_state, a) BEGIN
        n_state <= S0;
        w <= '0';
        CASE p_state IS
```

```
            IF a='1' THEN n_state <= S1;
            IF a='1' THEN n_state <= S1;
            ELSE n_state <= S0; END IF;
            w <= '1';
        WHEN OTHERS => n_state <= S0;
      END CASE;
    END PROCESS combinational;
```

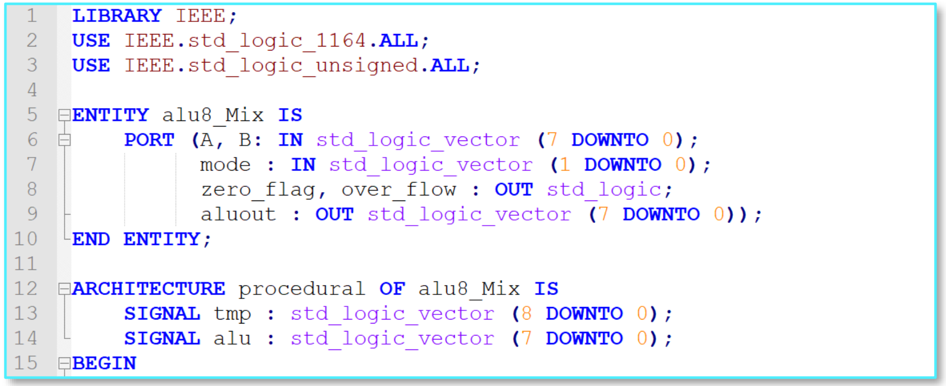
```
    sequential: PROCESS (clk) BEGIN
        IF (clk = '1' AND clk'EVENT) THEN
            IF rst = '1' THEN
                p_state <= S0;
            ELSE
                p_state <= n_state;
            END IF;
        END IF;
    END PROCESS sequential;

END ARCHITECTURE;
```

- Combinational and sequential parts are connected by p_state and n_state of type state
- Combinational part uses p_state assigned to n_state
- Sequential part clocks n_state to p_state
- A synthesizable description can contain any number of concurrent combinational and/or sequential processes



ALU as a synthesis example

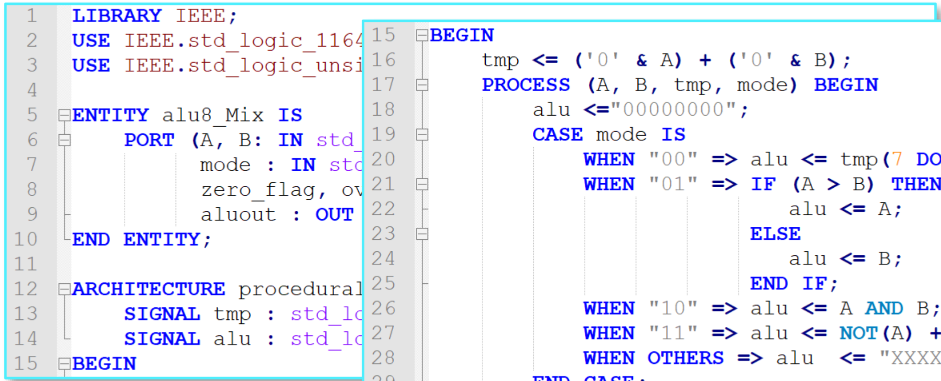
- Multiple Concurrent Statements



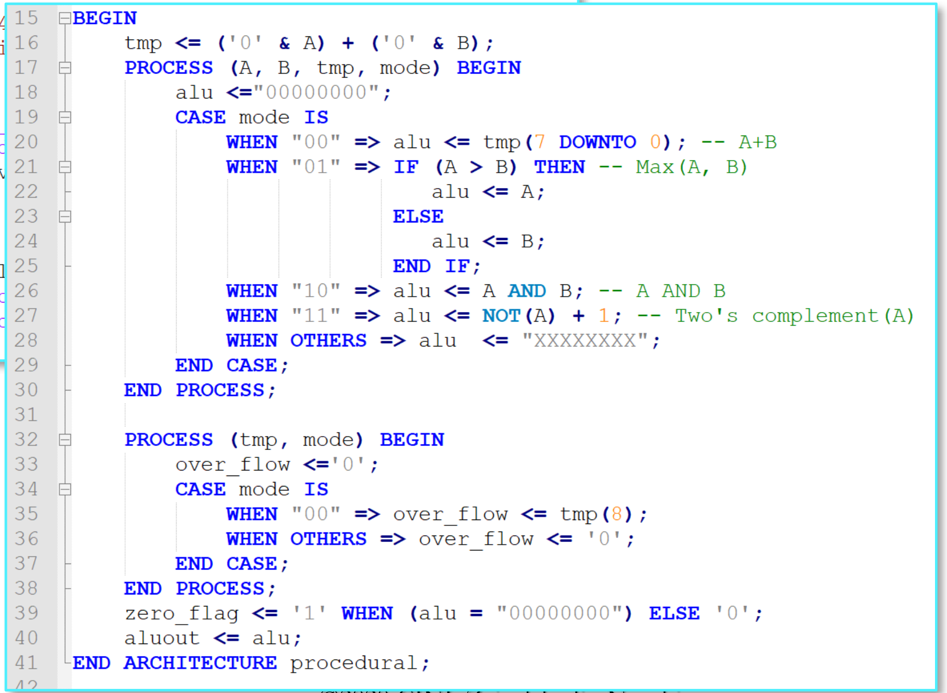
```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

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);
BEGIN
```

ALU as a synthesis example

- Multiple Concurrent Statements



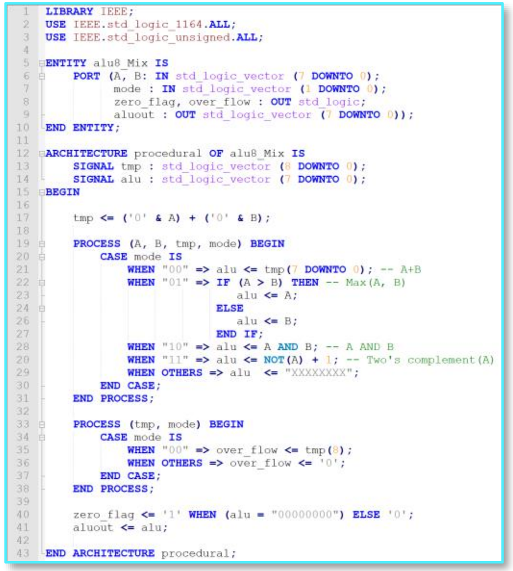
```
LIBRARY IEEE;
USE IEEE.std_logic_1164
USE IEEE.std_logic_unsi

ENTITY alu8_Mix IS
    PORT (A, B: IN std_
           mode : IN std
           zero_flag, ov
           aluout : OUT
END ENTITY;

ARCHITECTURE procedural
    SIGNAL tmp : std_lo
    SIGNAL alu : std_lo
BEGIN
```



```
BEGIN
    tmp <= ('0' & A) + ('0' & B);
    PROCESS (A, B, tmp, mode) BEGIN
        alu <="00000000";
        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
        over_flow <='0';
        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;
```

Circuit Netlists

VHDL Description

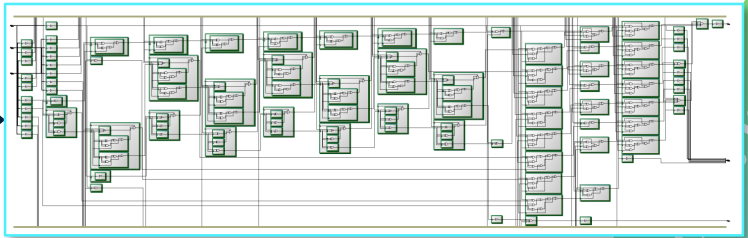


```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

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);
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;
```

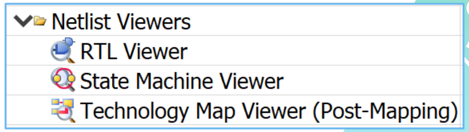
VHDL Synthesis

- Synthesizing ALU
- Use Quartus
- Target library: Cyclone IV GX FPGA
- The synthesis output can be viewed in various forms
- RTL view with abstract symbol models

Circuit Netlists

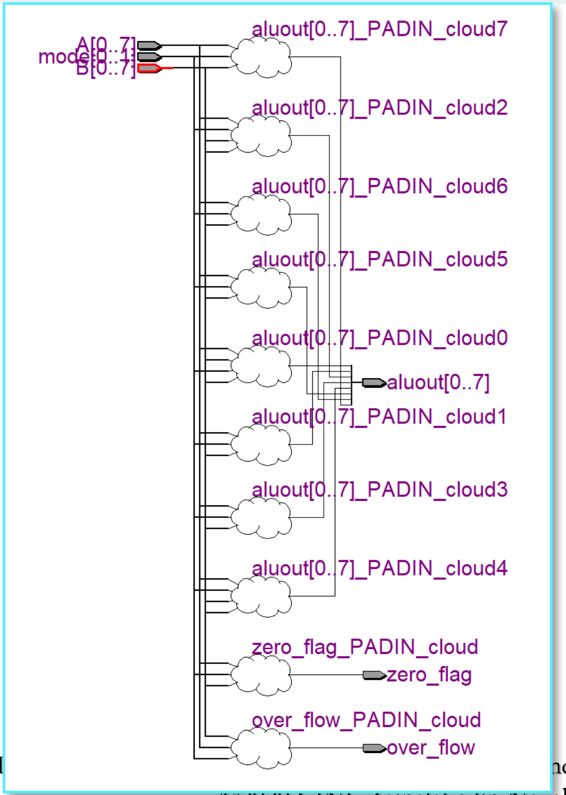


Outputs



ALU:

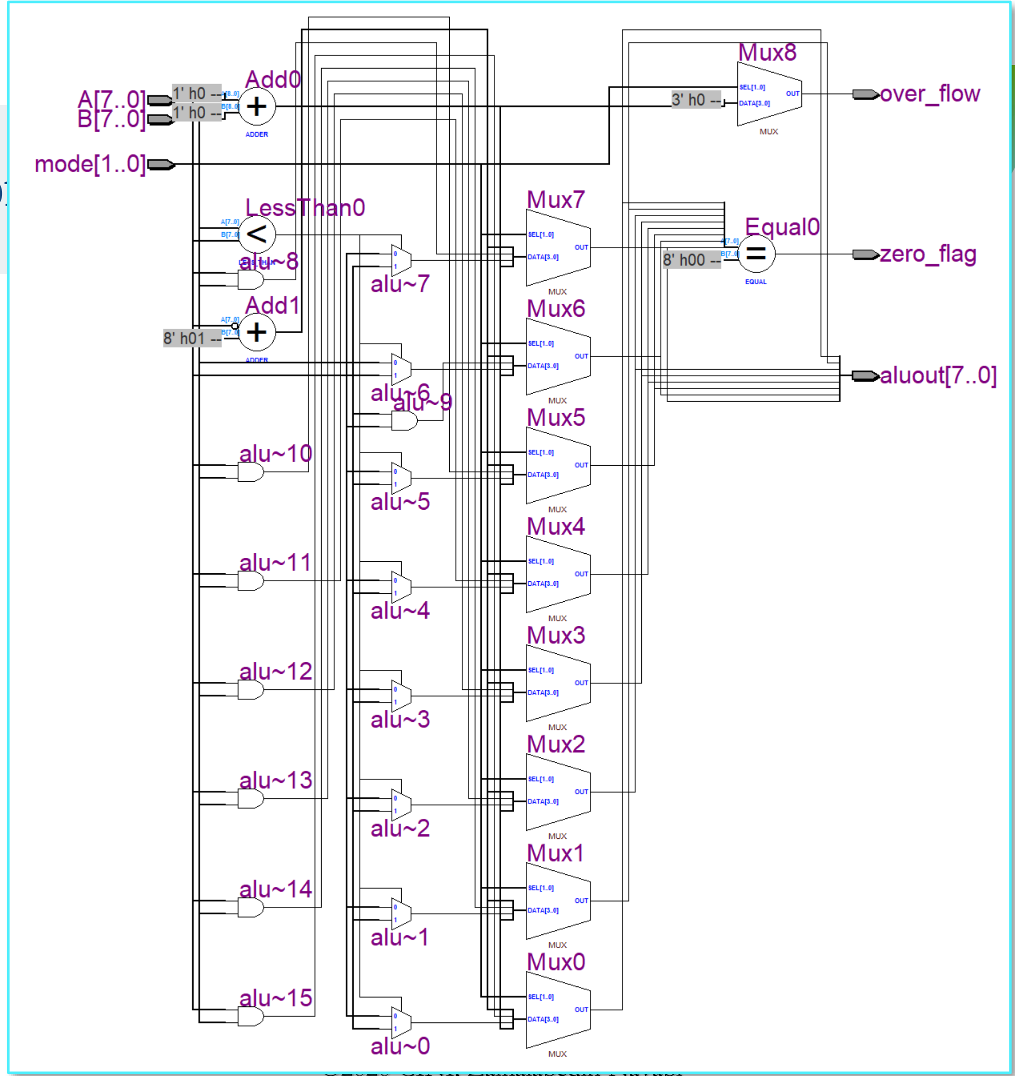
- Combinational Logics


A[0..7]
mode[0..1]
B[0..7]
aluout[0..7]_PADIN_cloud7
aluout[0..7]_PADIN_cloud2
aluout[0..7]_PADIN_cloud6
aluout[0..7]_PADIN_cloud5
aluout[0..7]_PADIN_cloud0
aluout[0..7]
aluout[0..7]_PADIN_cloud1
aluout[0..7]_PADIN_cloud3
aluout[0..7]_PADIN_cloud4
zero_flag_PADIN_cloud
zero_flag
over_flow_PADIN_cloud
over_flow

- Overall view of layout
- Layout uses combinational clouds

RT Level o

- Multiple Concurrent Statements



- RTL view of the synthesis output details of abstract symbols are shown here

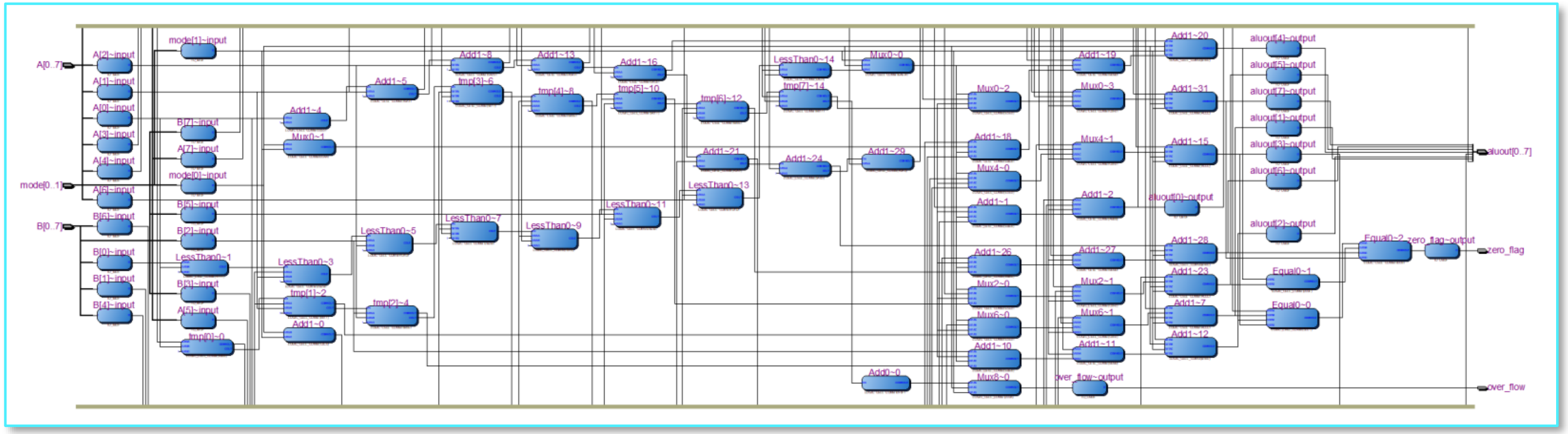
CYBER CHALLENGE CyberChallenge.IT

cini Cybersecurity National Lab

Technology Map of ALU:

- Multiple Concurrent Statements

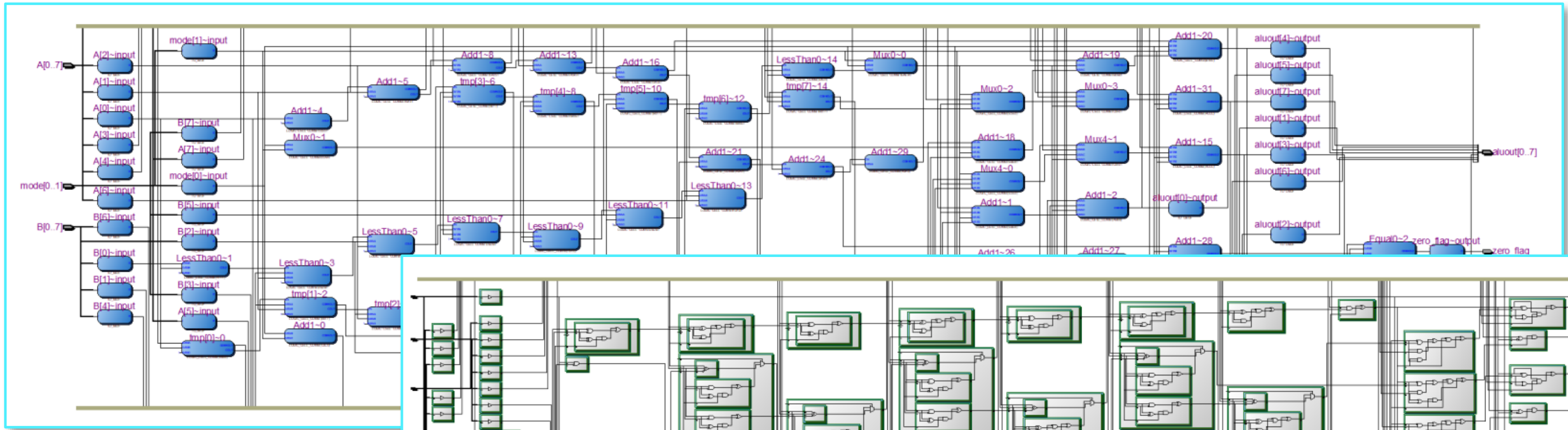
- Synthesis result of ALU
- Technology map view of ALU with equivalent gates with details

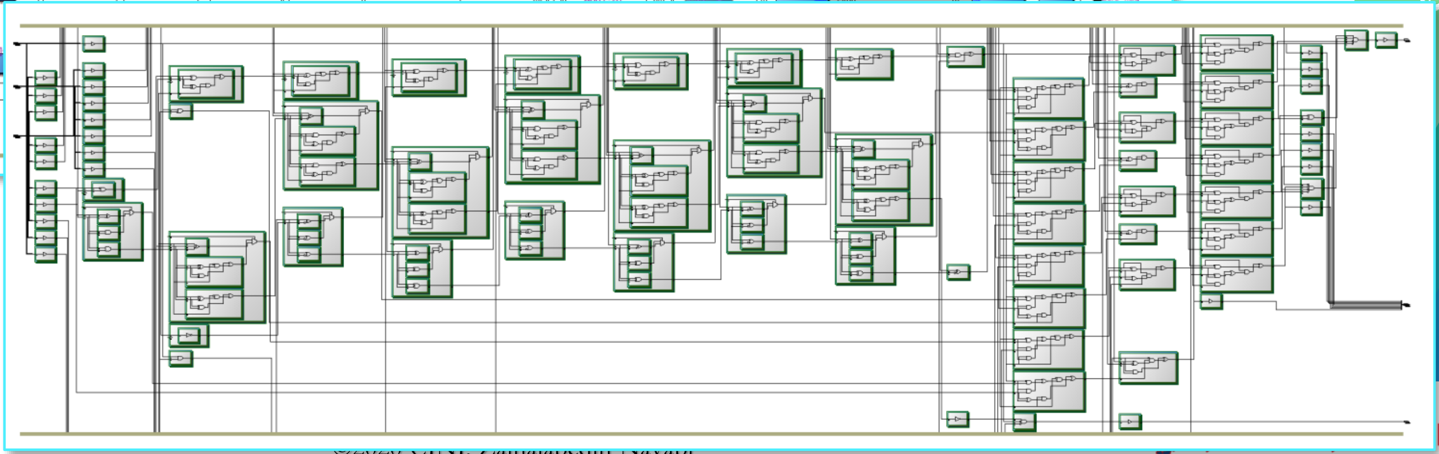


Technology Map of ALU:

- Multiple Concurrent Statements

- Synthesis result of ALU
- Technology map view of ALU with equivalent gates with details

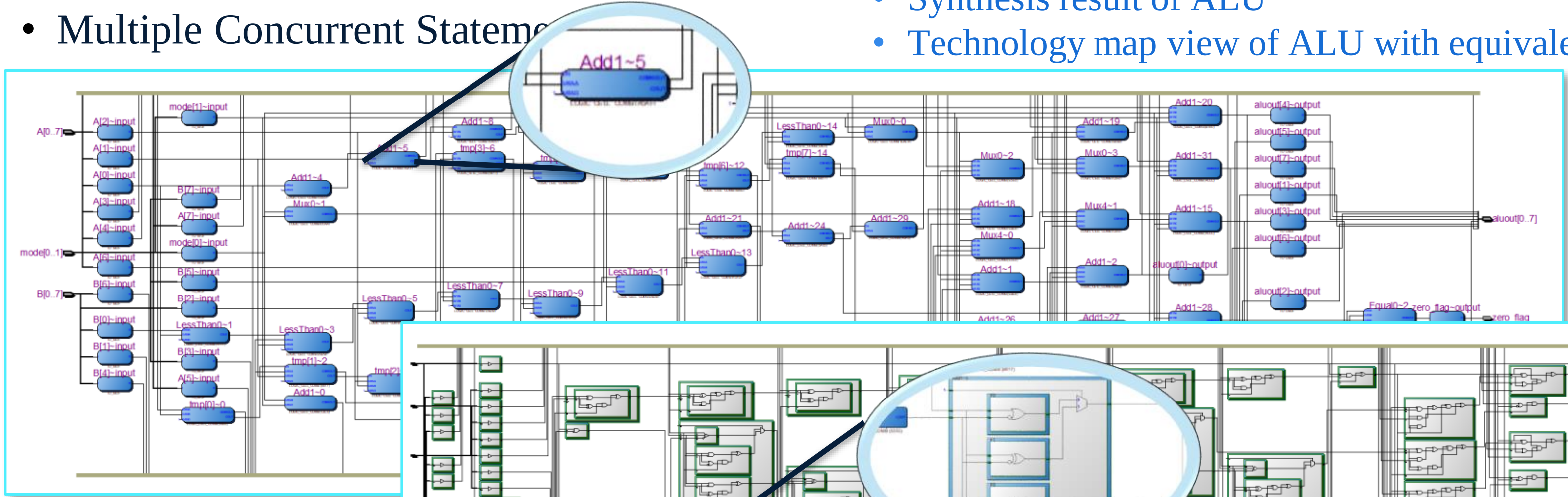


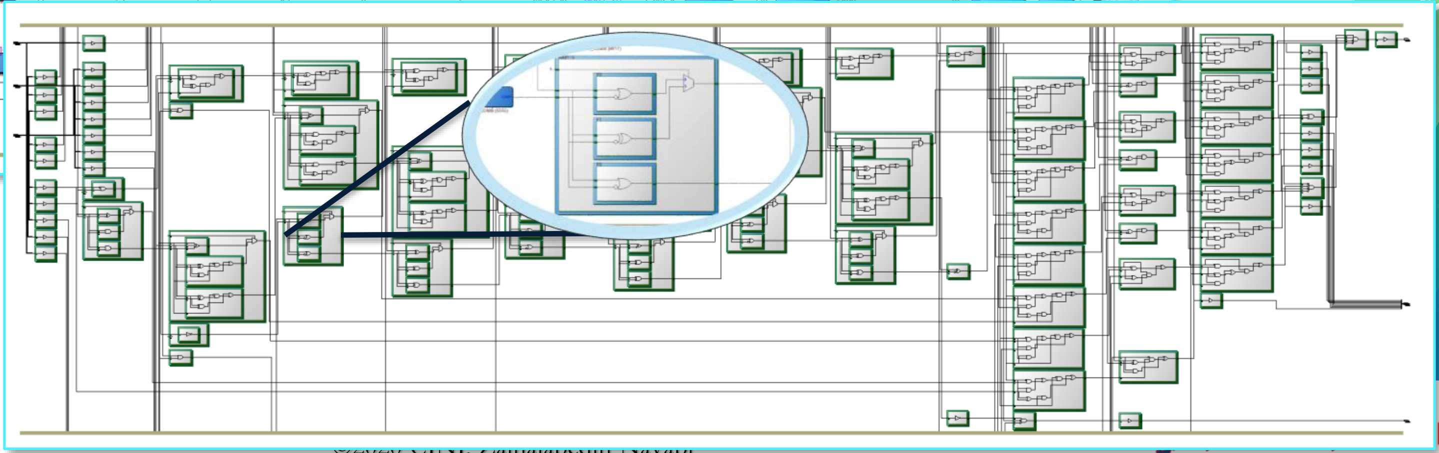


Technology Map of ALU:

- Multiple Concurrent Stateme

- Synthesis result of ALU
- Technology map view of ALU with equivalent gates with details

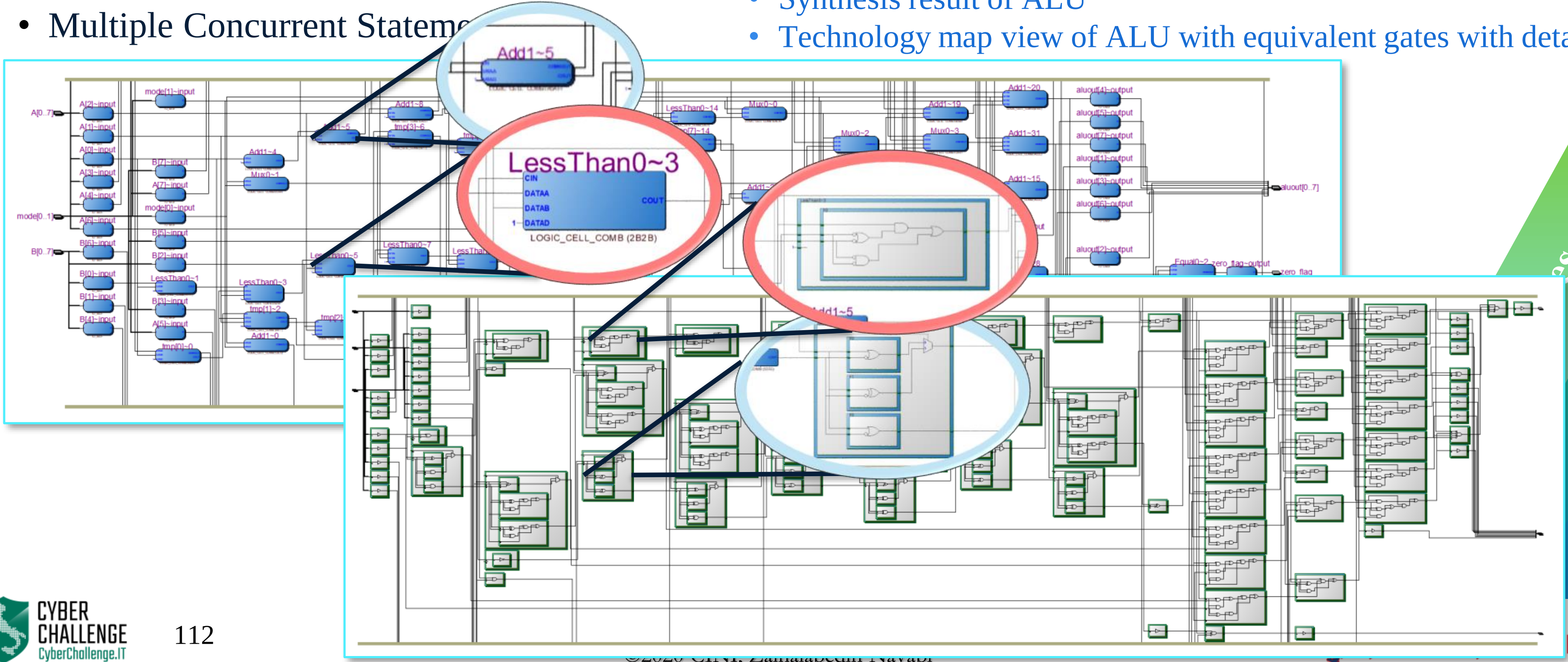




Technology Map of ALU:

- Multiple Concurrent Stateme

- Synthesis result of ALU
- Technology map view of ALU with equivalent gates with details



Add1~5
LessThan0~3
CIN
DATAA
DATAB
DATAD
COUT
LOGIC_CELL_COMB (2B2B)
Add1~5
A[0..7]
mode[0..1]
B[0..7]
aluout[0..7]
zero_flag