به نام خدا

تکلیف شماره اول

درس:

طراحی سیستم های دیجیتال

استاد درس:

دکتر مروستی

اعضا:

محمدعلی مجتهدسلیمانی-992023039

حامد خسروی-992023010

**Q1**. Explain the basic blocks and interconnections of FPGA, CPLD, and PAL. Then, compare these chips in terms of area, performance, power, applications, and security.

* Field-Programmable Gate Arrays (FPGAs), Complex Programmable Logic Devices (CPLDs), and Programmable Array Logic (PAL) devices are all types of programmable logic devices, but they differ in their architecture, complexity, and capabilities.

1. FPGAs (Field-Programmable Gate Arrays):

* Basic Blocks:
* Logic Blocks (Configurable Logic Blocks or CLBs): These are the basic building blocks containing lookup tables (LUTs) and flip-flops for implementing logic functions.
* Routing Resources: FPGAs have a vast interconnection network of programmable switches and wires that allow the logic blocks to be connected in various configurations.
* Input/Output (I/O) Blocks: These blocks interface the FPGA with external components and provide I/O capabilities.
* Embedded Resources: Modern FPGAs may also include embedded memory blocks (Block RAMs), Digital Signal Processing (DSP) blocks, and even hard-wired processors or microcontrollers.
* Interconnections: FPGAs have a highly flexible and hierarchical interconnect structure, allowing logic blocks to be connected in various ways.

2. CPLDs (Complex Programmable Logic Devices):

* + Basic Blocks:
  + Macrocells: CPLDs are based on a sum-of-products (SOP) or product-of-sums (POS) architecture, where macrocells are used to implement logic functions.
  + Interconnect Matrix: CPLDs have a centralized interconnect matrix that connects the macrocells.
  + Interconnections: CPLDs have a simpler interconnect structure compared to FPGAs, with a centralized interconnect matrix.

3. PALs (Programmable Array Logic):

* + Basic Blocks:
  + AND Array: PALs consist of an AND array of programmable fuses that implement product terms (ANDs).
  + OR Array: The outputs of the AND array feed into an OR array, which combines the product terms to implement the desired logic functions.
  + Interconnections: PALs have a fixed interconnect structure, with the AND array feeding into the OR array.
* **Comparison:**
  + **Area**: FPGAs are the largest devices, occupying the most silicon area, followed by CPLDs, and then PALs, which are the smallest.
  + **Performance**: FPGAs offer the highest performance due to their flexible and optimized interconnect structure, followed by CPLDs and then PALs.
  + **Power**: PALs typically have the lowest power consumption, followed by CPLDs and then FPGAs, which consume the most power due to their complexity and high capacitance interconnects.
  + **Applications**:
  + FPGAs are used in a wide range of applications requiring high performance, flexibility, and parallel processing, such as digital signal processing, video processing, networking, and prototyping.
  + CPLDs are suitable for mid-range applications that require moderate complexity and performance, such as control systems, communications, and glue logic.
  + PALs are typically used in simpler and lower-density applications, such as address decoders, state machines, and glue logic.
  + Security: FPGAs and CPLDs offer various security features, such as bitstream encryption, secure boot, and tamper protection. PALs, being older devices, lack advanced security features found in modern FPGAs and CPLDs.

Q2.In digital systems it is often necessary to have circuits that can shift the bits of a vector by one or more bit positions to the left or right. One kind of shifter circuit shifts more bit positions at a time. If the bits that are shifted out are placed into the vacated positions on the left, then the circuit effectively rotates the bits of the input vector by a specified number of bit positions. Such a circuit is often called a barrel shifter. Write a VHDL code to implement a four-bit barrel shifter that rotates the bits by 0, 1, 2, or 3 bit positions as determined by the valuation of two control signals s1 and s0.

* Code:

library IEEE;

use IEEE.std\_logic\_1164.all;

entity HOME is

port (

data\_in : in std\_logic\_vector(3 downto 0);

control : in std\_logic\_vector(1 downto 0);

data\_out : out std\_logic\_vector(3 downto 0)

);

end entity HOME;

architecture rtl of HOME is

signal temp : std\_logic\_vector(3 downto 0);

begin

temp(0) <= data\_in(0) when control = "00" else

data\_in(1) when control = "01" else

data\_in(2) when control = "10" else

data\_in(3);

temp(1) <= data\_in(0) when control = "00" else

data\_in(2) when control = "01" else

data\_in(3) when control = "10" else

data\_in(1);

temp(2) <= data\_in(0) when control = "00" else

data\_in(3) when control = "01" else

data\_in(0) when control = "10" else

data\_in(2);

temp(3) <= data\_in(1) when control = "00" else

data\_in(3) when control = "01" else

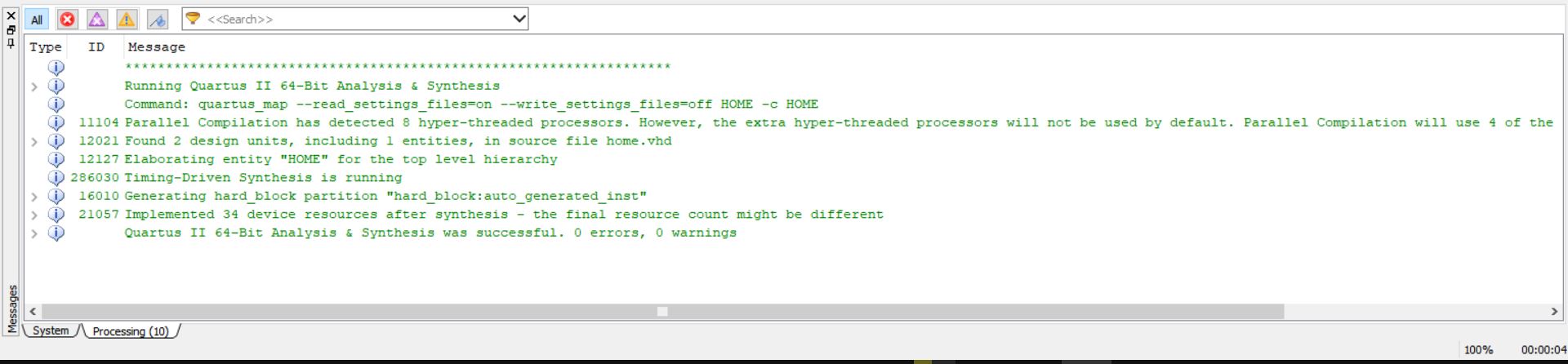
data\_in(1) when control = "10" else

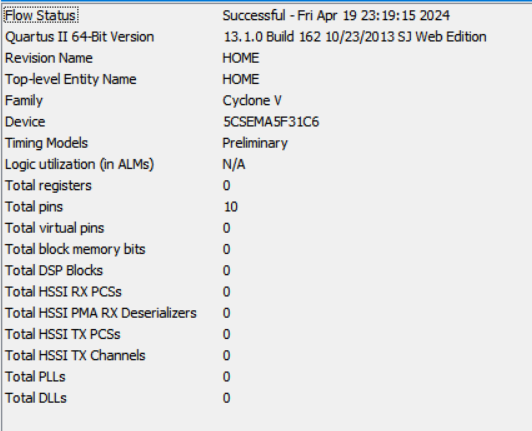
data\_in(0);

data\_out <= temp;

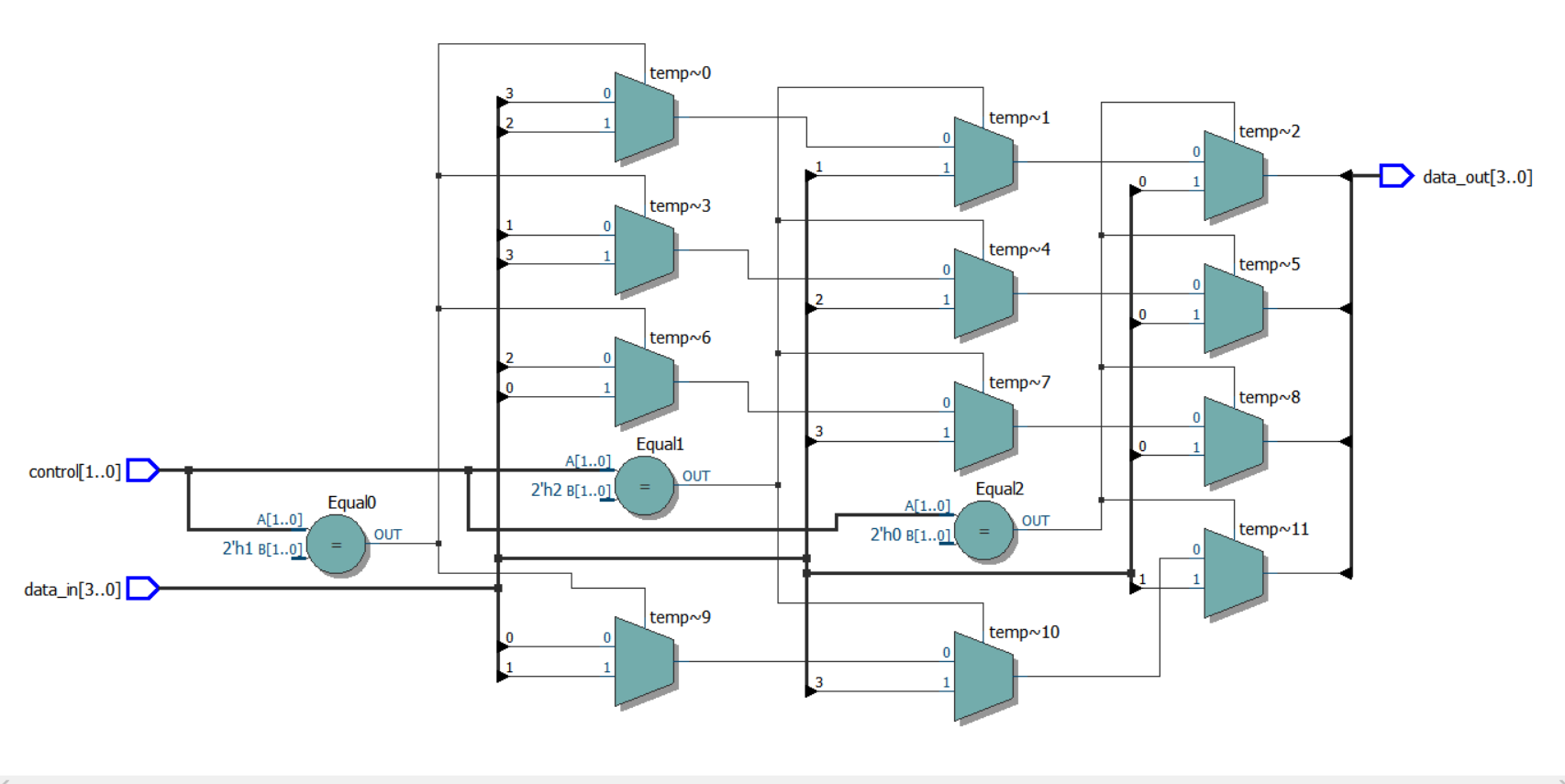
end architecture rtl;

* **output after compile**:

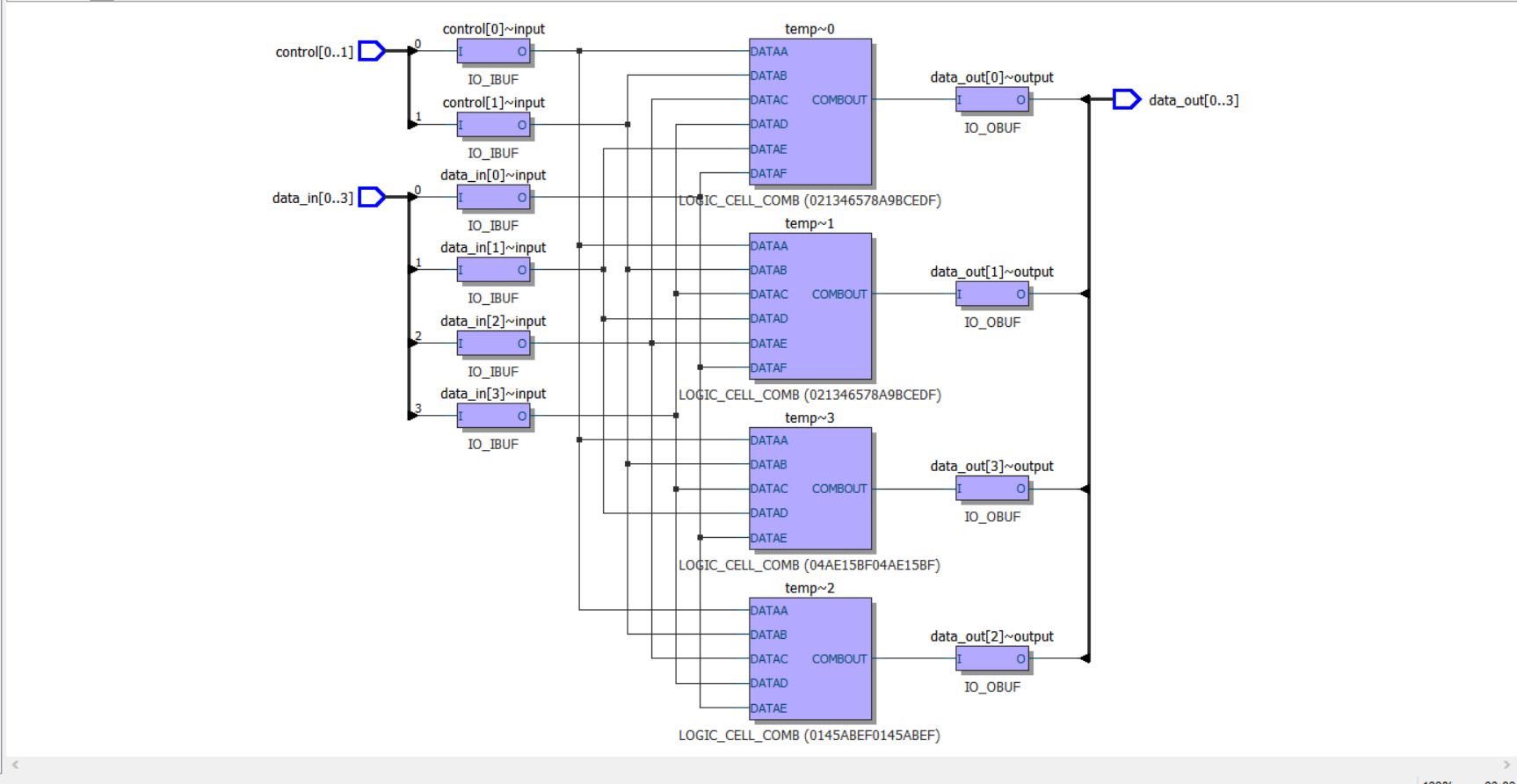




* **RTL VIEW**:



* **Post-mapping view**:



**Q3.**

* CODE

library IEEE;

use ieee.numeric\_std.all;

entity HOME is

port (

bcd\_in : in unsigned(3 downto 0); -- BCD input

seg\_out: out unsigned(6 downto 0) -- 7-segment output

);

end entity HOME;

-- Architecture definition

architecture rtl of HOME is

begin

-- Combinational logic for the decoder

process(bcd\_in)

begin

case bcd\_in is

when "0000" => seg\_out <= "0111000"; -- Display 0

when "0001" => seg\_out <= "0011000"; -- Display 1

when "0010" => seg\_out <= "1010100"; -- Display 2

when "0011" => seg\_out <= "1010010"; -- Display 3

when "0100" => seg\_out <= "1100000"; -- Display 4

when "0101" => seg\_out <= "1100100"; -- Display 5

when "0110" => seg\_out <= "1110000"; -- Display 6

when "0111" => seg\_out <= "0000000"; -- Display 7

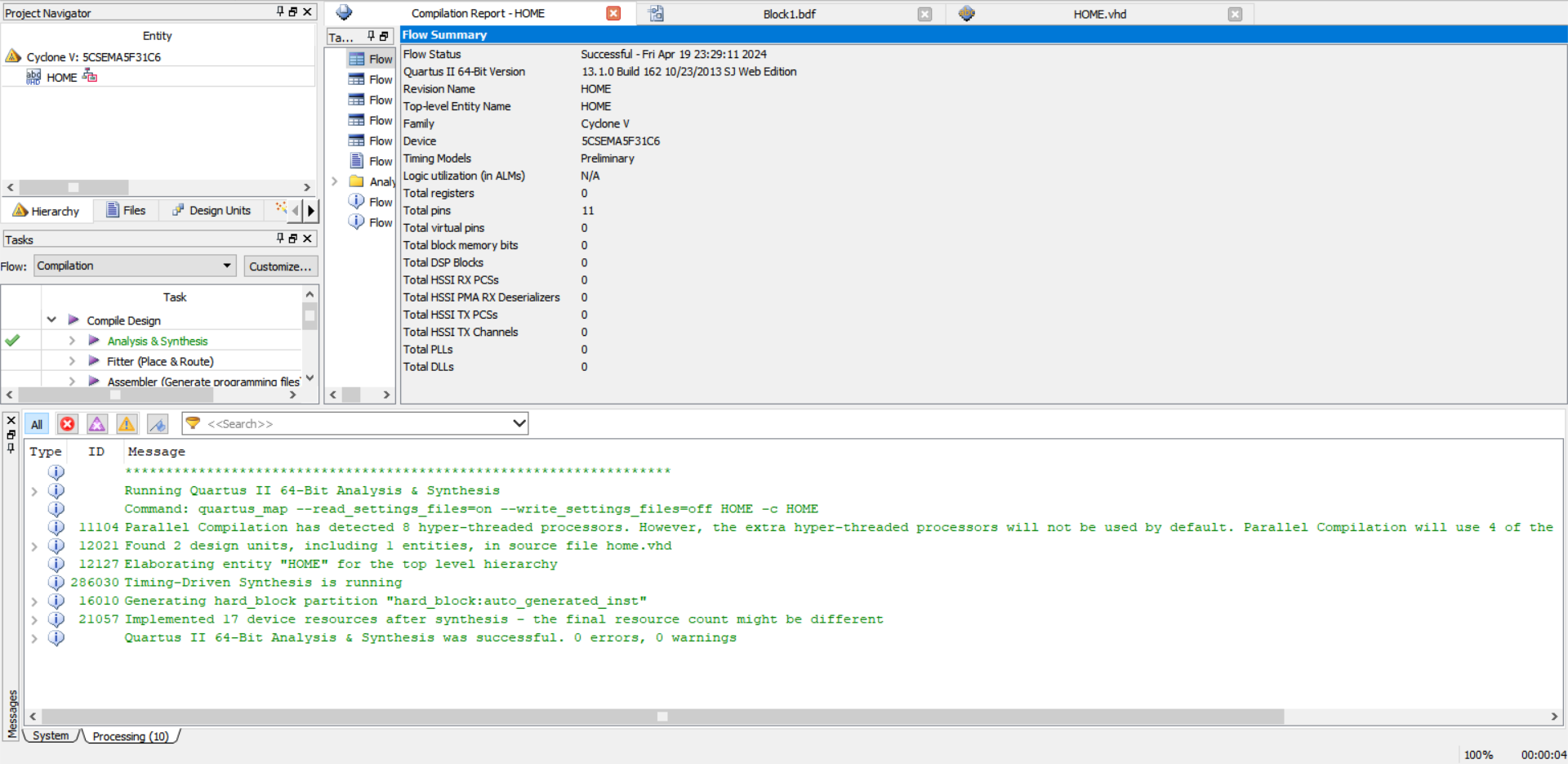
when others => seg\_out <= "1111111"; -- Display error

end case;

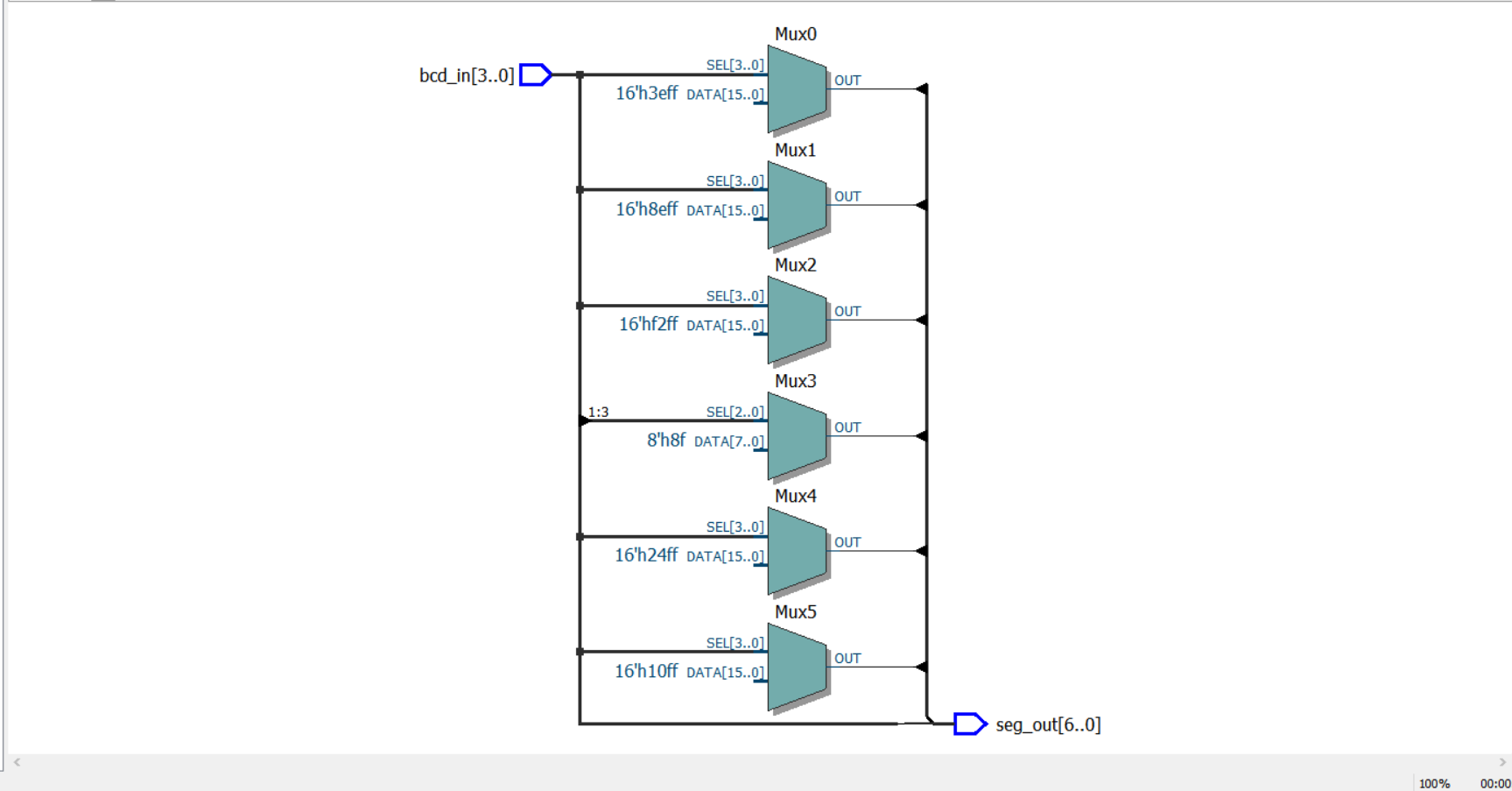
end process;

end architecture rtl;

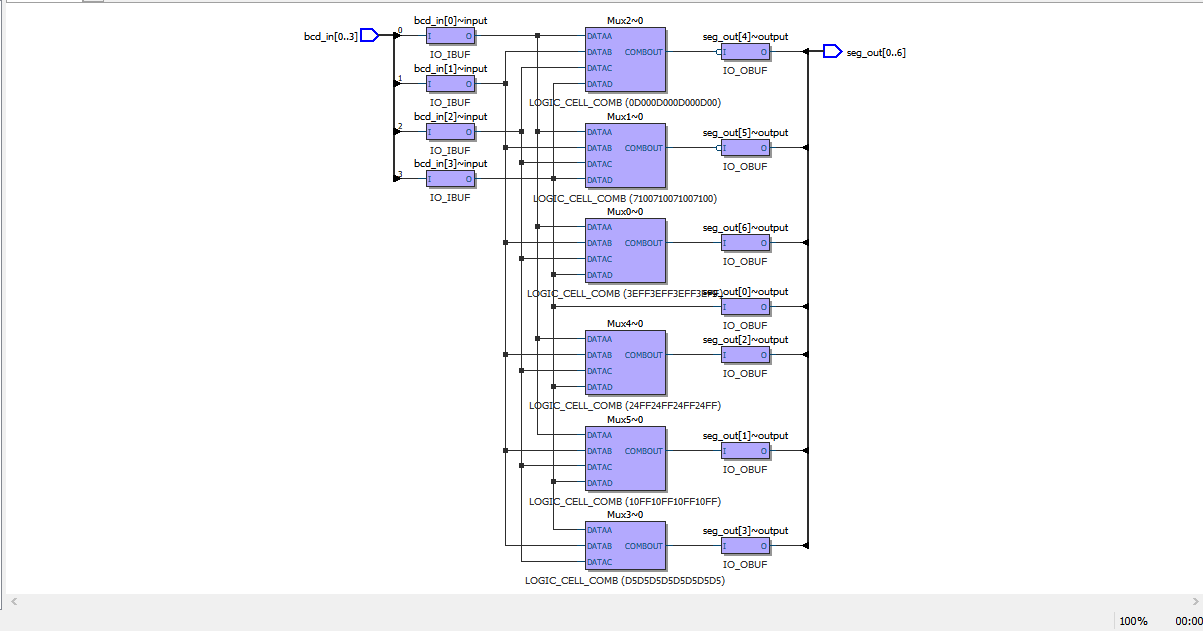
* output after compile:



* RTL VIEW:



* Post-mapping view:



**Q4.**

* Code:

-- Entity declaration for partial full adder

entity HOME is

port (

A : in bit;

B : in bit;

Cin: in bit;

P : out bit;

G : out bit

);

end entity HOME;

-- Architecture definition for partial full adder

architecture Behavioral of HOME is

begin

-- Calculate propagate (P)

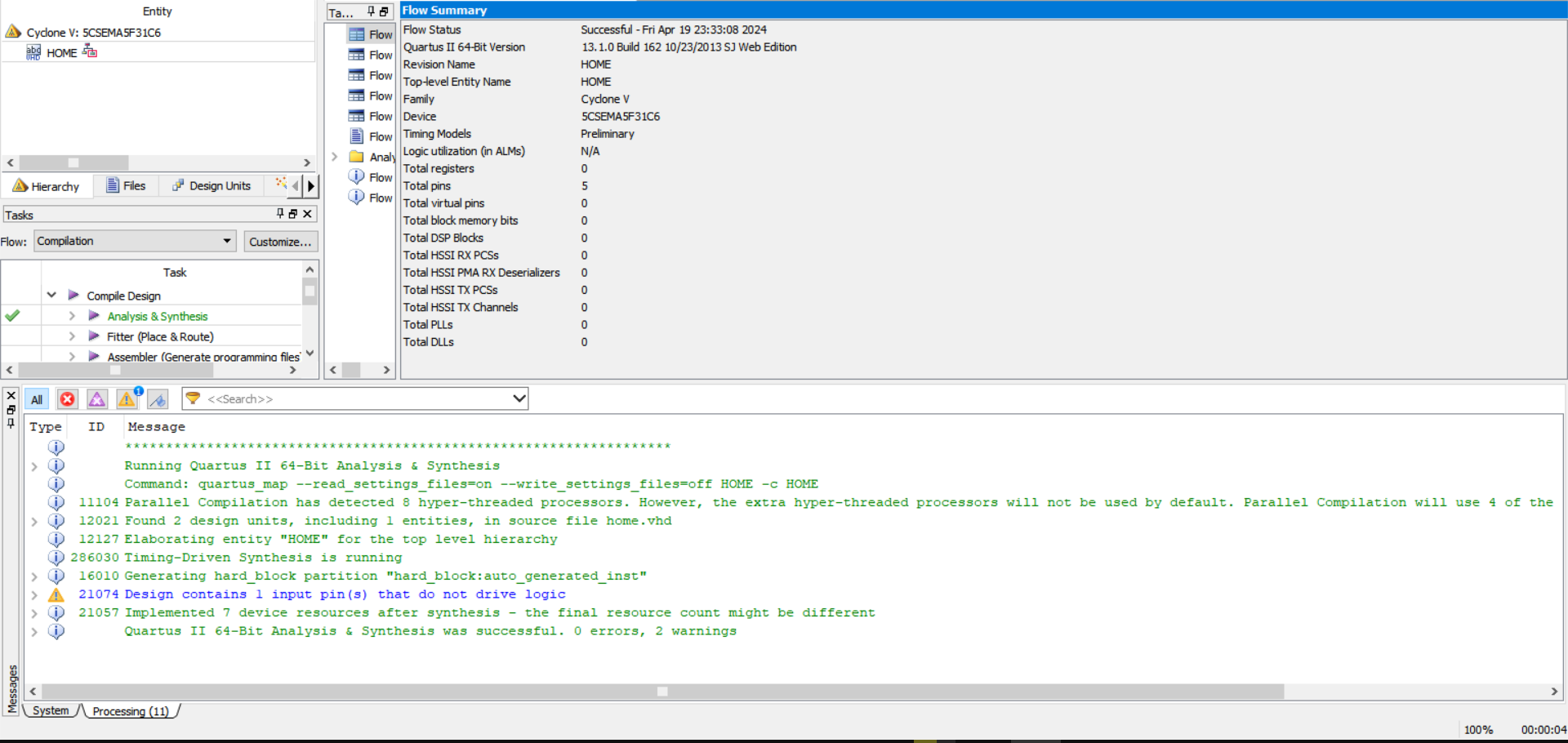
P <= A xor B;

-- Calculate generate (G)

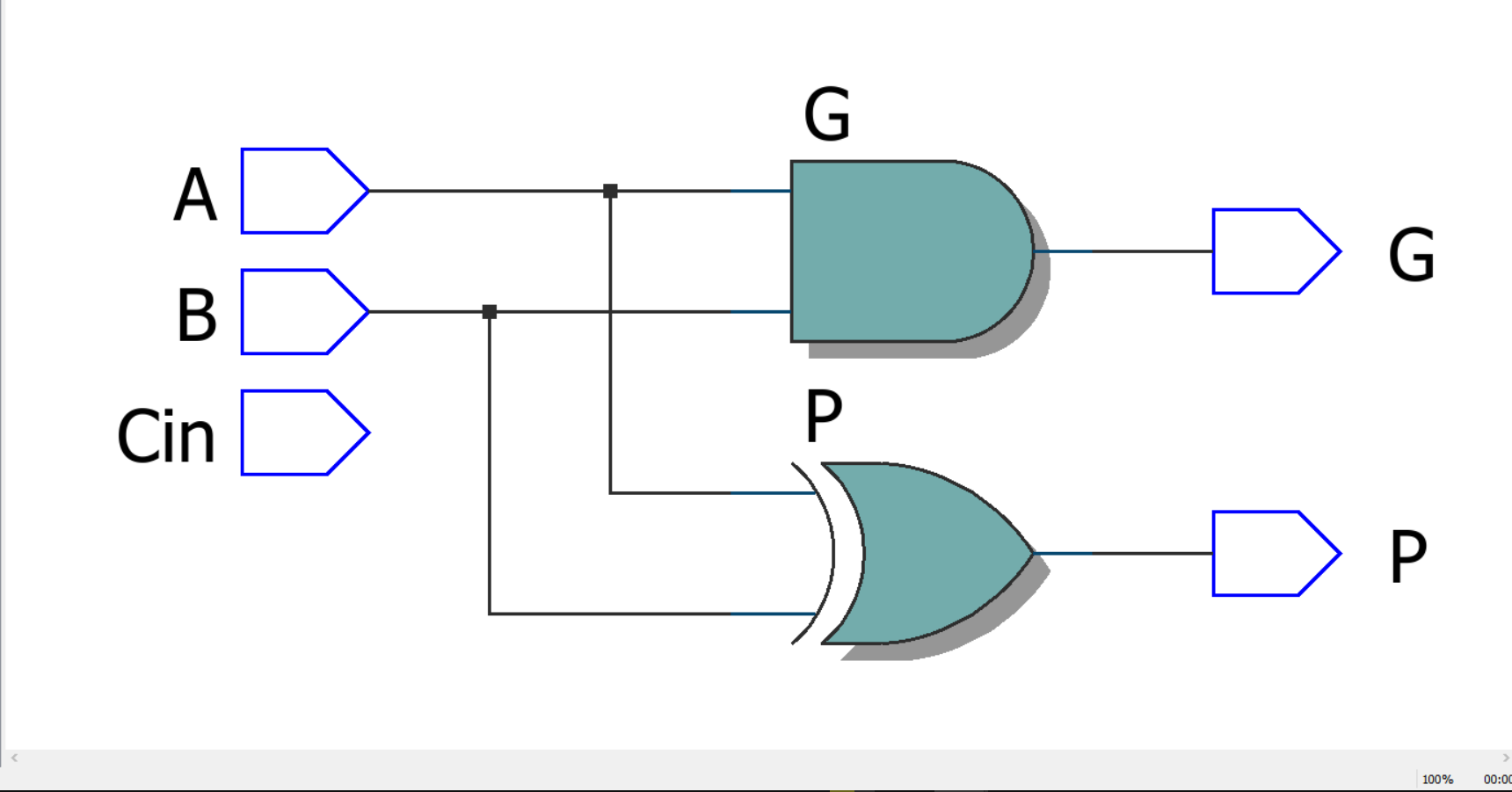
G <= A and B;

end architecture Behavioral;

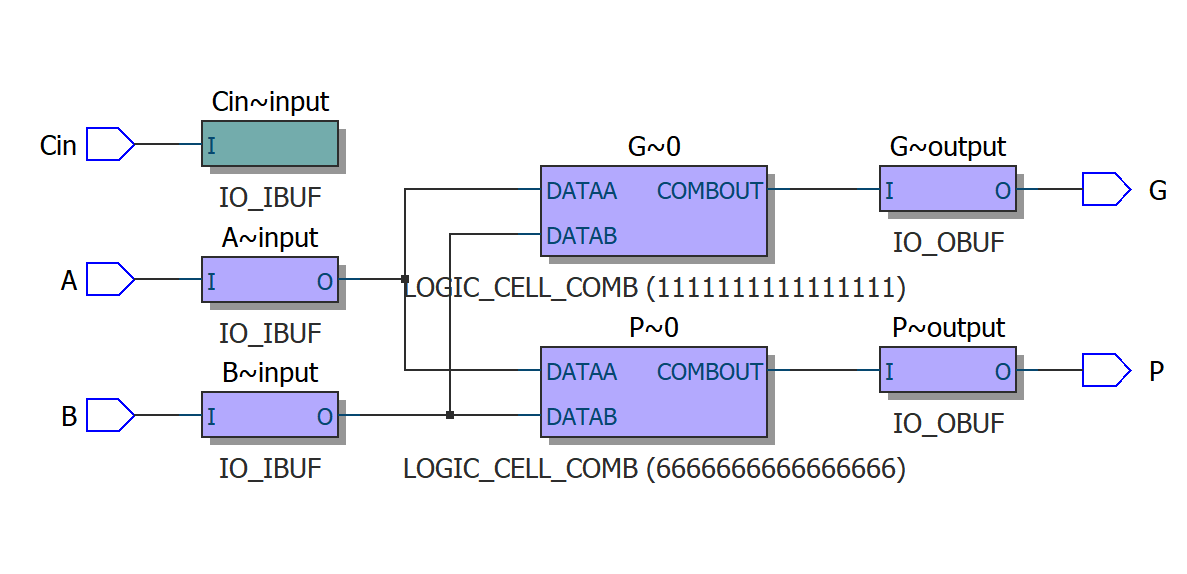
* output after compile:



* RTL VIEW:



* Post-mapping view:



**Q5.**

* Code

library ieee;

use ieee.std\_logic\_1164.all;

entity HOME is

port (

X, Y, C\_in : in std\_logic;

Sum, C\_out : out std\_logic

);

end entity HOME;

architecture rtl of HOME is

signal decoder\_out : std\_logic\_vector(7 downto 0);

begin

-- 3:8 decoder

decoder\_out <= "00000000" when X = '0' and Y = '0' and C\_in = '0' else

"00000001" when X = '0' and Y = '0' and C\_in = '1' else

"00000010" when X = '0' and Y = '1' and C\_in = '0' else

"00000011" when X = '0' and Y = '1' and C\_in = '1' else

"00000100" when X = '1' and Y = '0' and C\_in = '0' else

"00000101" when X = '1' and Y = '0' and C\_in = '1' else

"00000110" when X = '1' and Y = '1' and C\_in = '0' else

"00000111" when X = '1' and Y = '1' and C\_in = '1';

-- Sum output

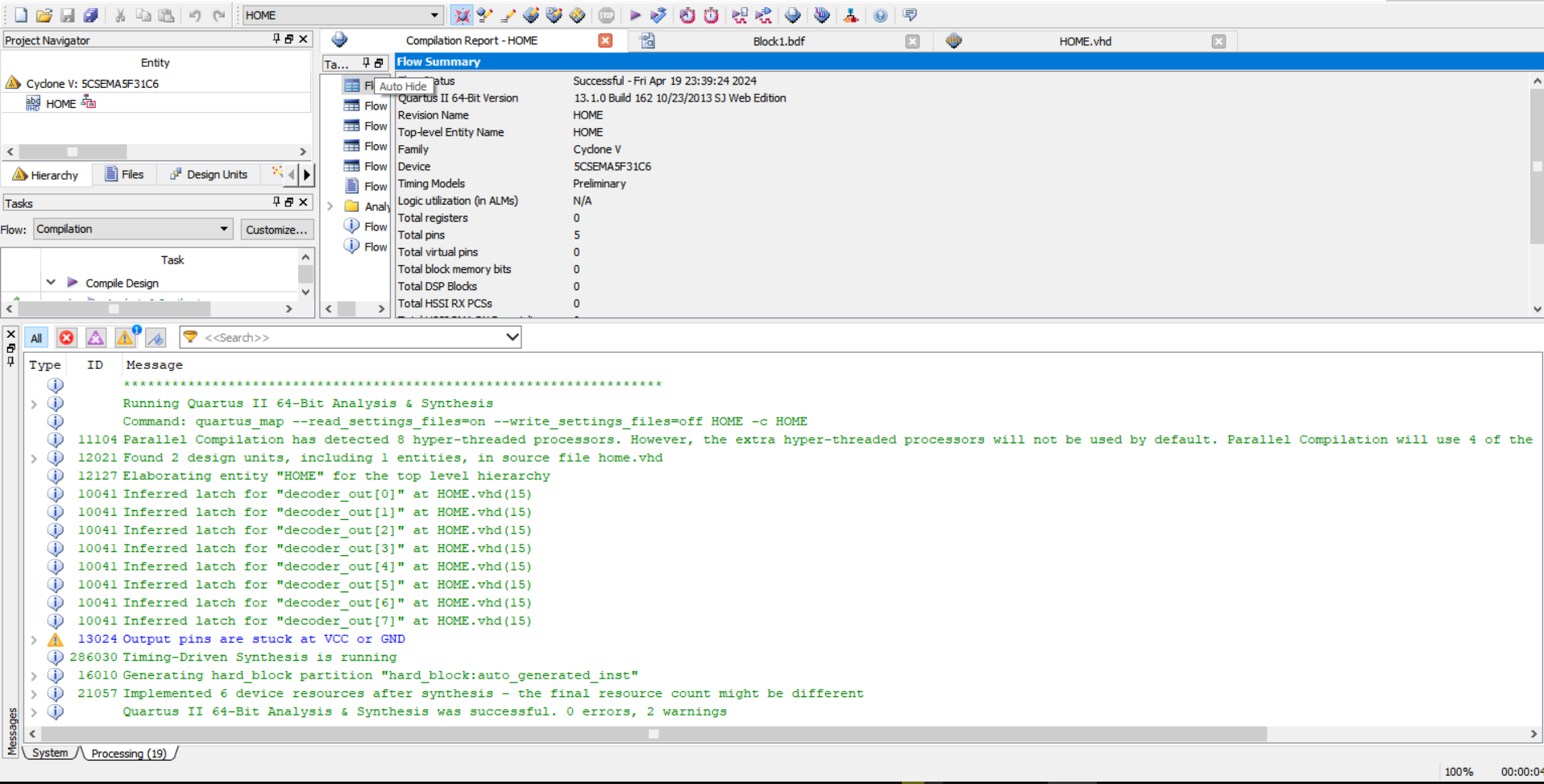
Sum <= decoder\_out(0) or decoder\_out(1) or decoder\_out(2) or decoder\_out(4) or decoder\_out(7);

-- C-out output

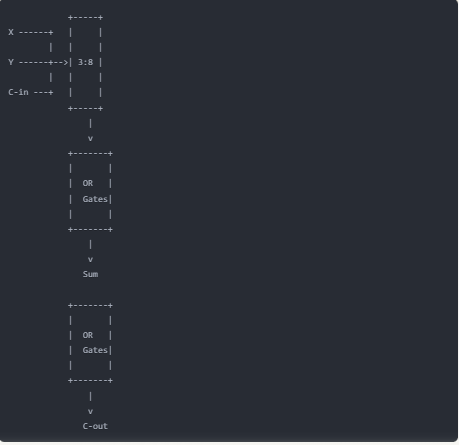
C\_out <= decoder\_out(3) or decoder\_out(5) or decoder\_out(6) or decoder\_out(7);

end architecture rtl;

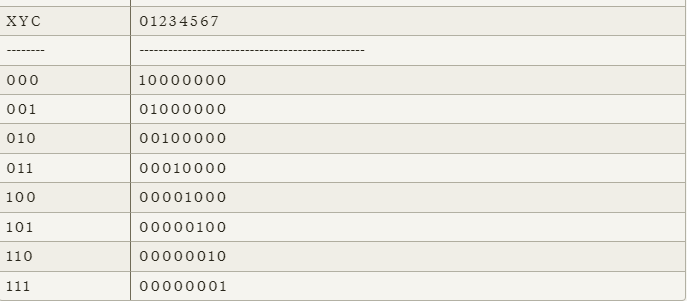
* output after compile:



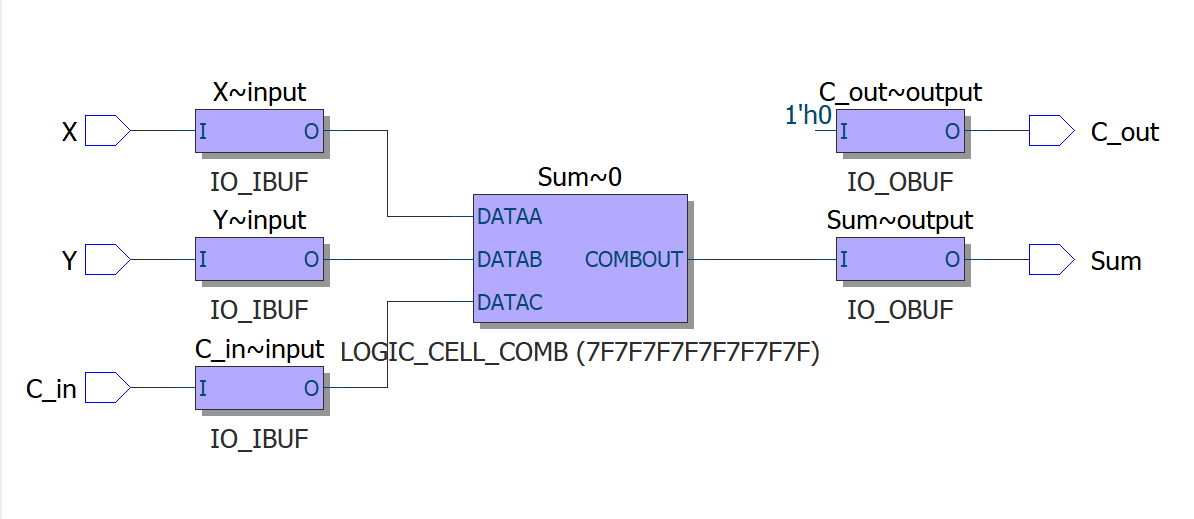
* A part: Architecture:



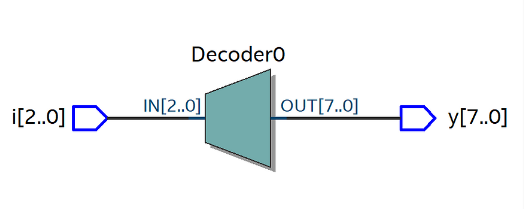
* Truth table:



* Post-mapping view:



* RTL view:



**Q6**.

* CODE

library ieee;

use ieee.std\_logic\_1164.all;

entity HOME is

port (

A, B, C, D : in std\_logic;

Z : out std\_logic

);

end entity HOME;

architecture rtl of HOME is

signal E, F : std\_logic;

begin

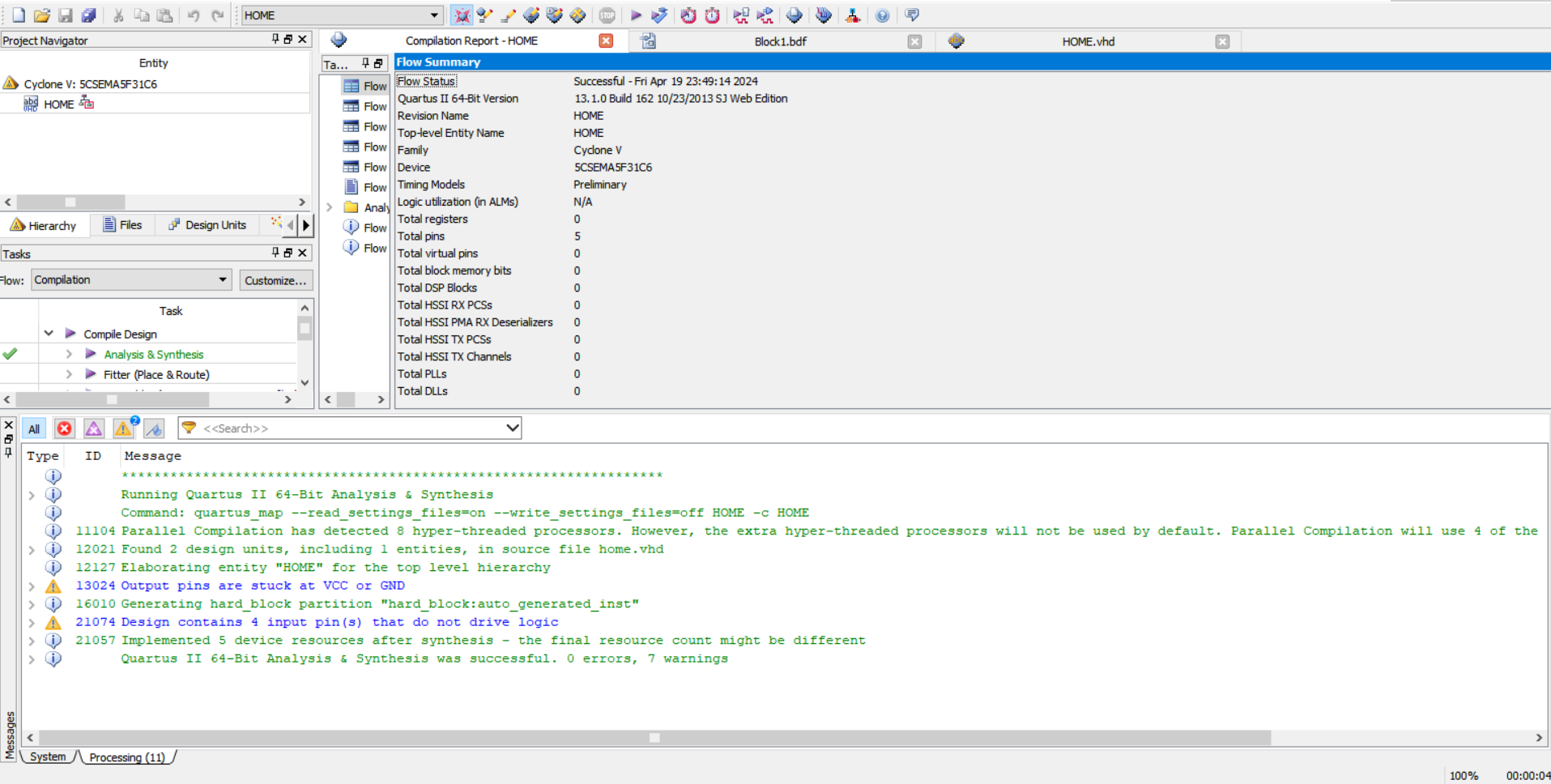
E <= (A and not B and not C and not D) or (B and C and D);

F <= (not A and B and not C) or (A and not B and C);

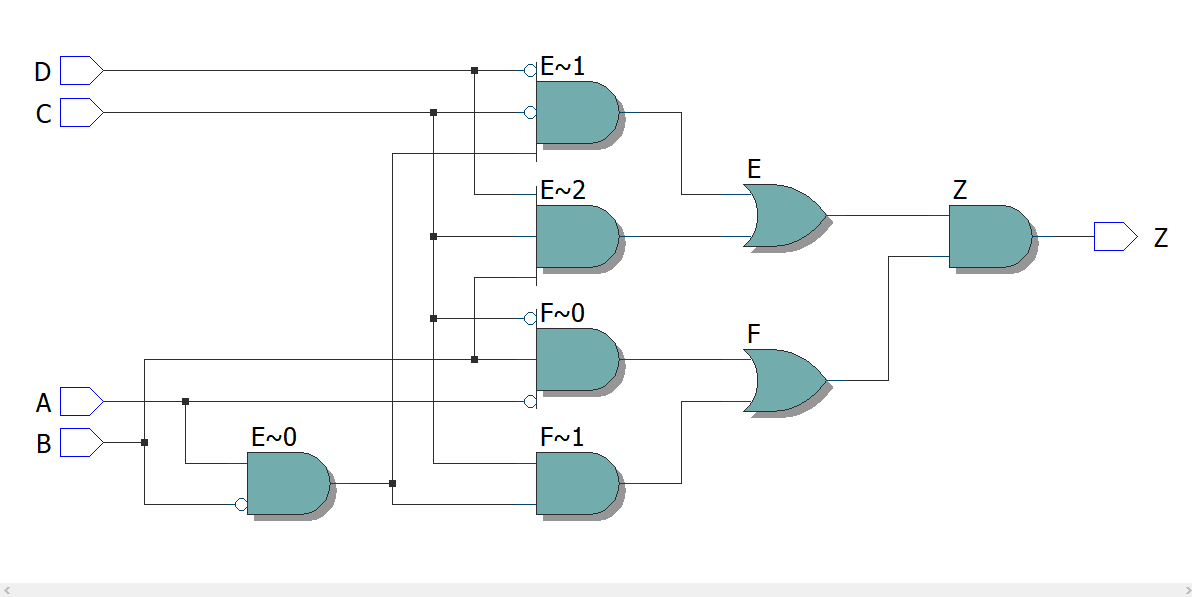
Z <= E and F;

end architecture rtl;

* output after compile:



* RTL view:



* Post-mapping view:

