Digital Systems Design II

COEN 313 (FJ-X)

Experiment 3

Andre Hei Wang Law

4017 5600

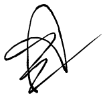
Performed on October 18, 2021

Due on November 1, 2021

“I certify that this submission is my original work and meets the Faculty’s Expectations of

Originality”.

Saturday, November 1, 2021

4017 5600

**1) Modelsim Simulation Results**

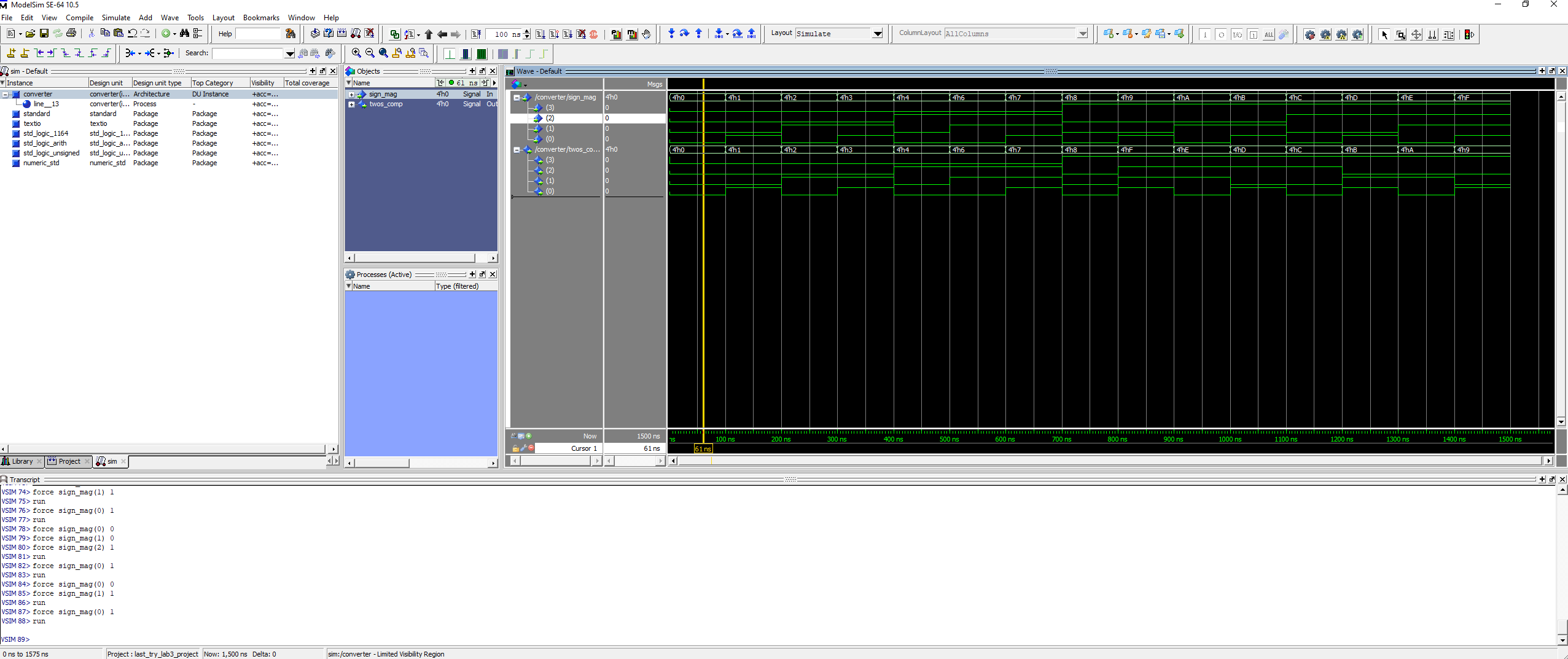


Figure 1.1. Modelsim Simulation Results for All Possible Inputs

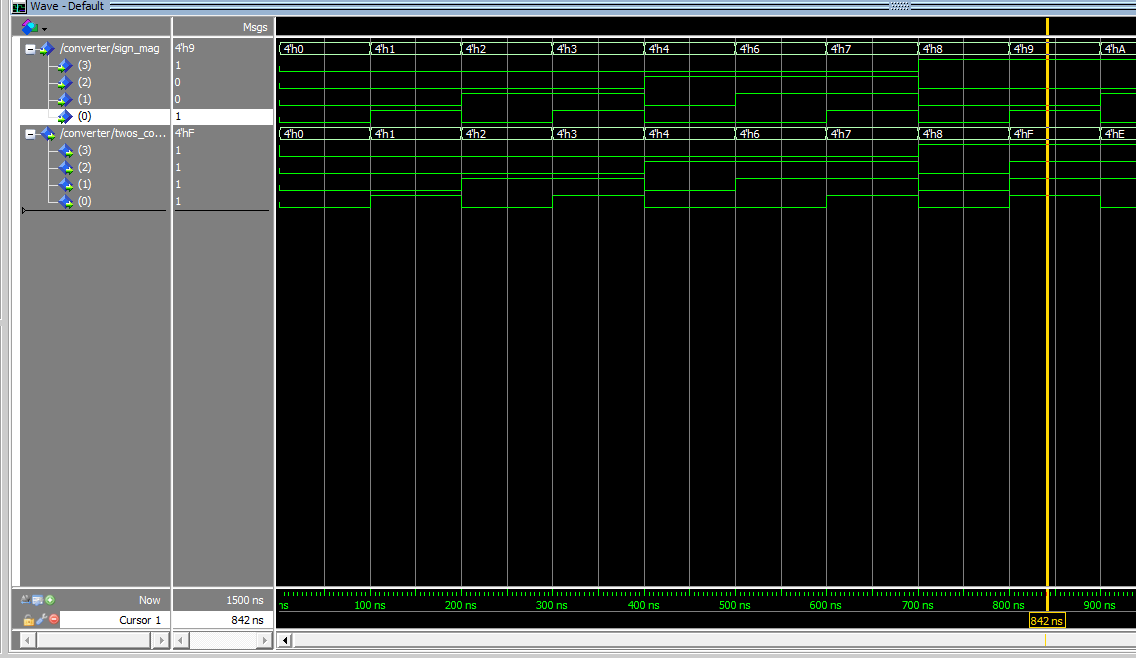


Figure 1.2. Simulation Example with Inputs of “1001” and Outputs “1111”



Figure 1.3. Simulation Example with Inputs of “1100” and Outputs “1100”

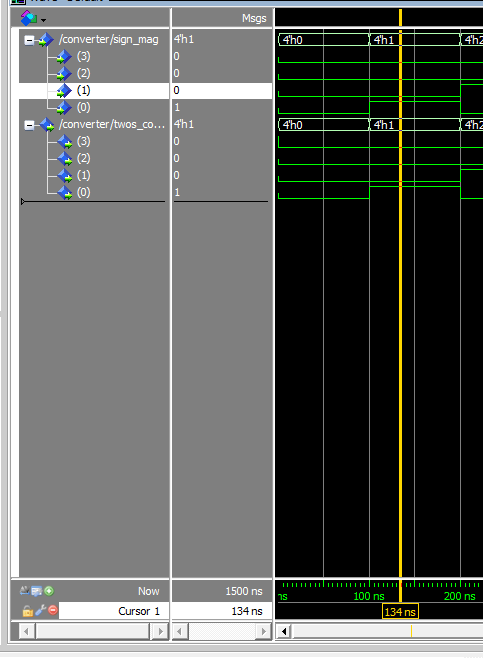
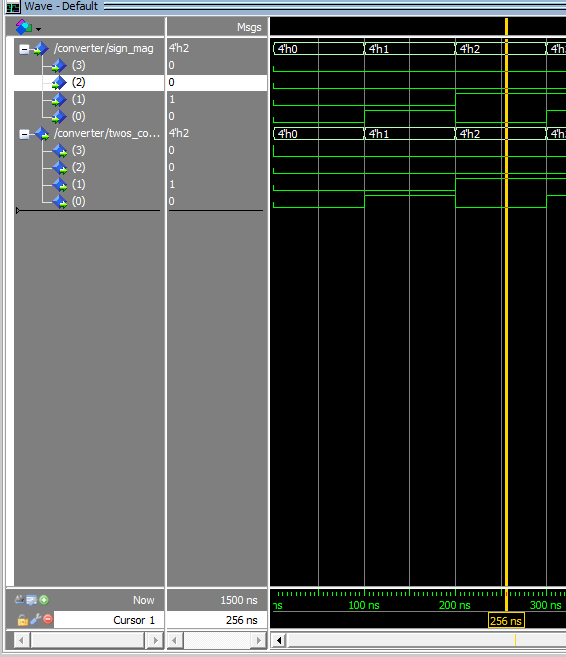
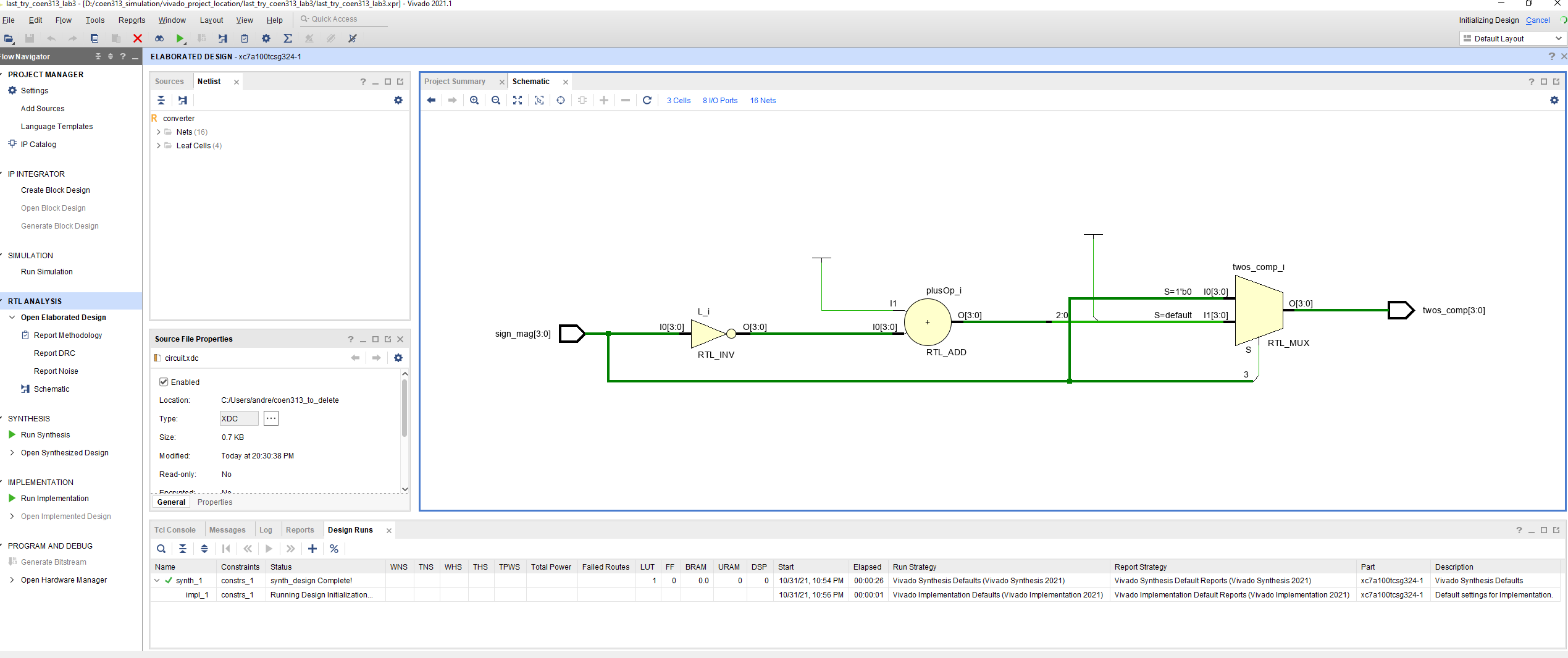


Figure 1.3. Simulation Example with Inputs of “1100” and Outputs “1100”

**2) RTL Schematic Diagrams of the Elaborated and Implemented Circuit**



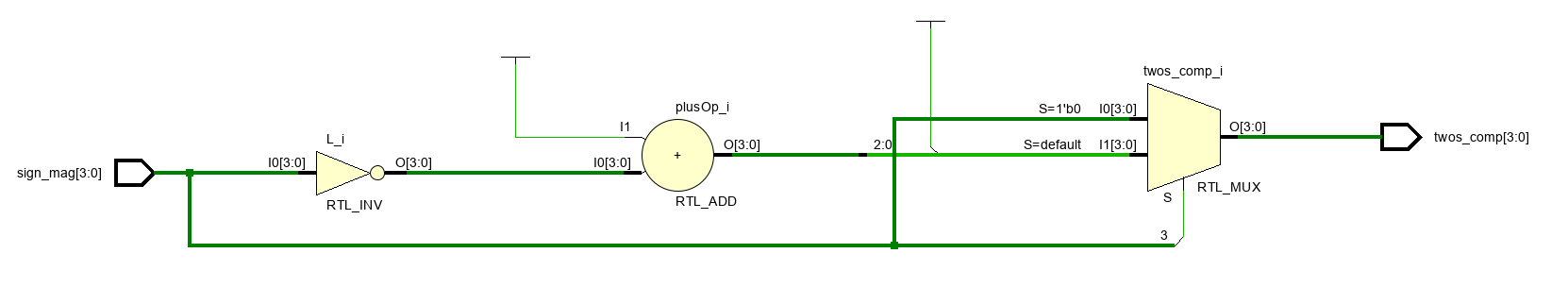


Figure 2.1. Elaborated Schematic of the Circuit

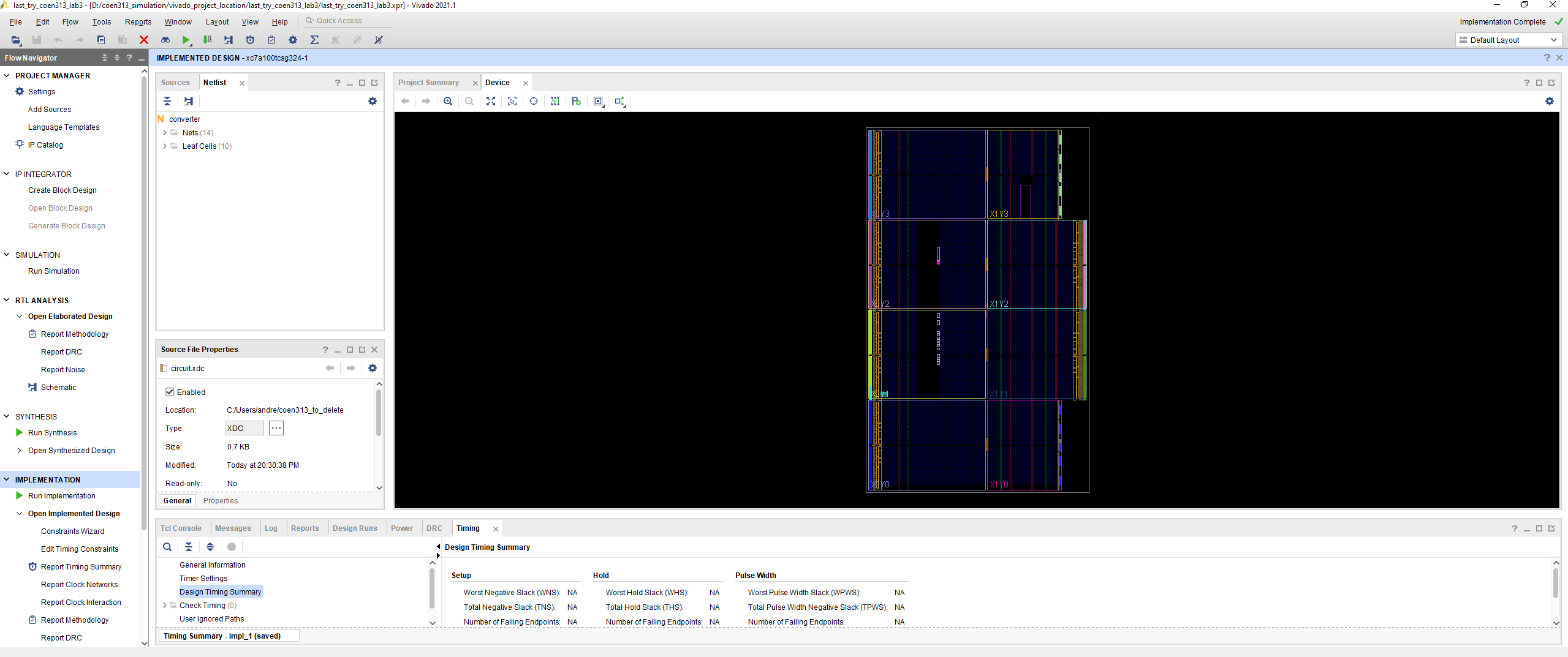


Figure 2.2. Default Implemented Design View

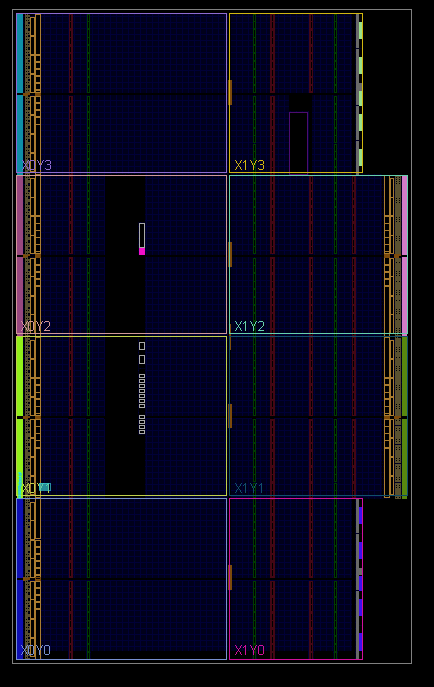


Figure 2.3. Default Implemented Design View (Zoomed In)

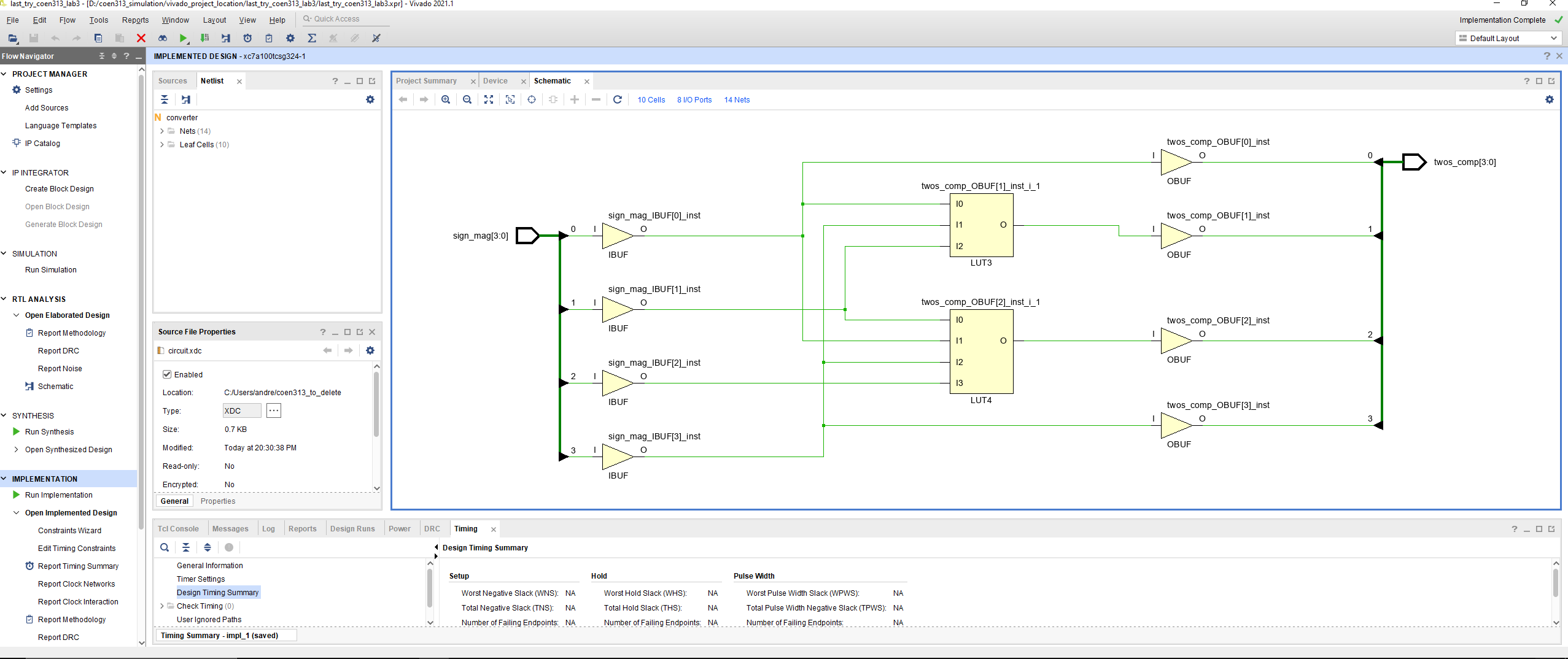


Figure 2.4. Schematic View of the Implemented Design

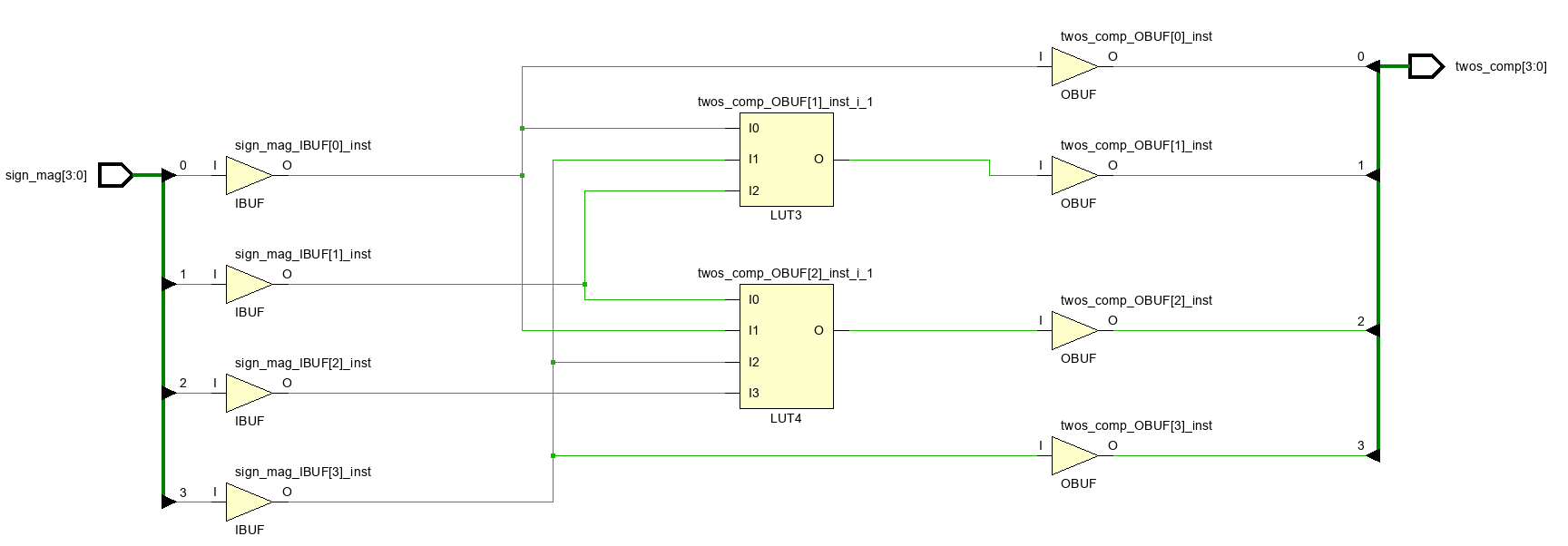


Figure 2.5. Schematic View of the Implemented Design (Zoomed In)

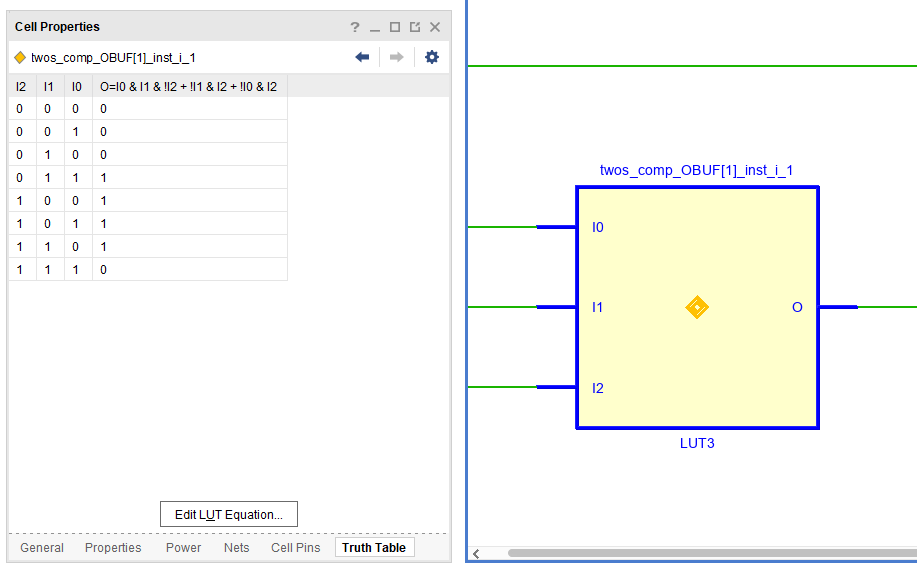
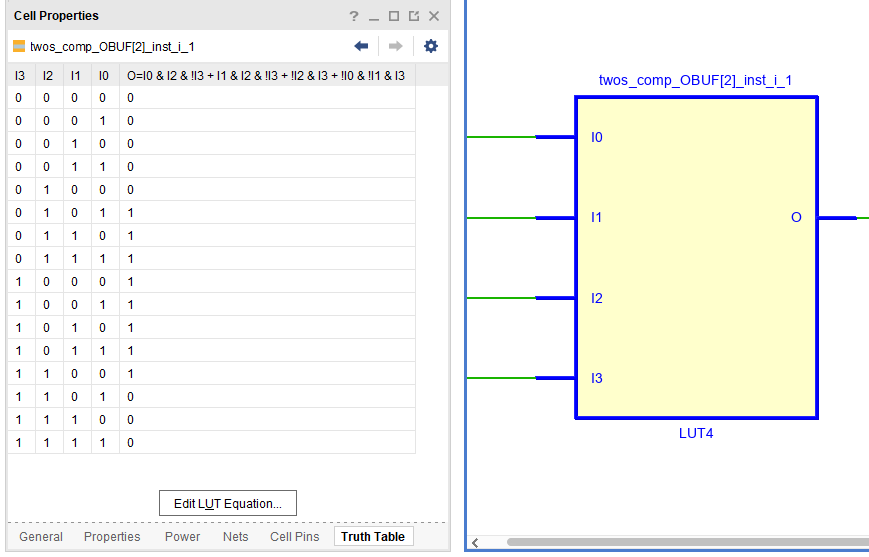


Figure 2.6. Truth Table of LUT3 (left) and LUT4 (right)

**3) Synthesis and Implementation Log Files by Vivado**

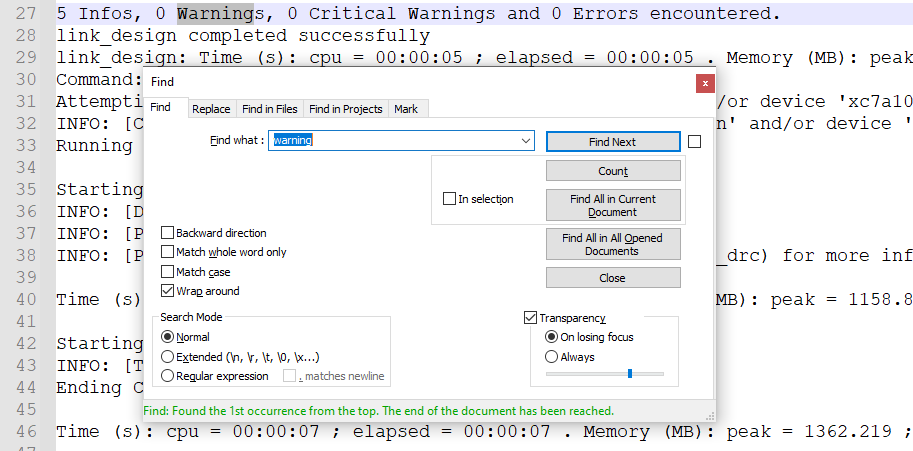


Figure 3.1.a. Warning of the Implementation Log Files

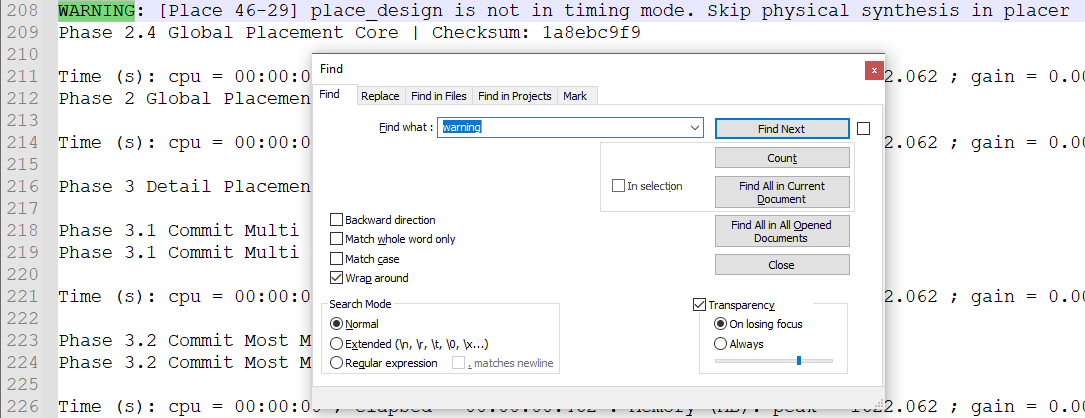


Figure 3.1.b. Warning of the Implementation Log Files

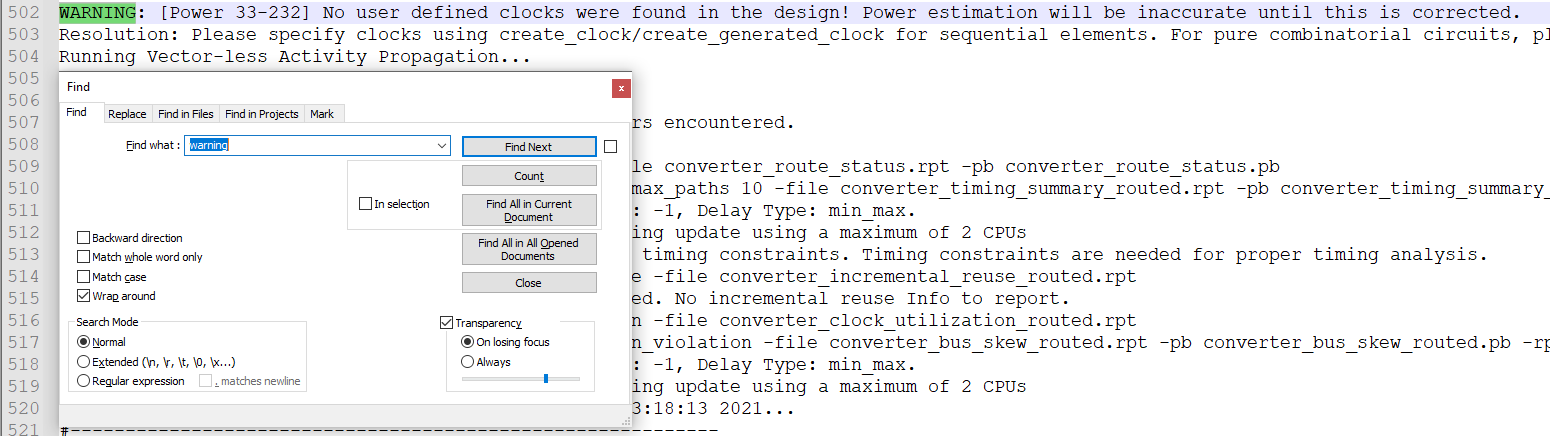


Figure 3.1.c. Warning of the Implementation Log Files

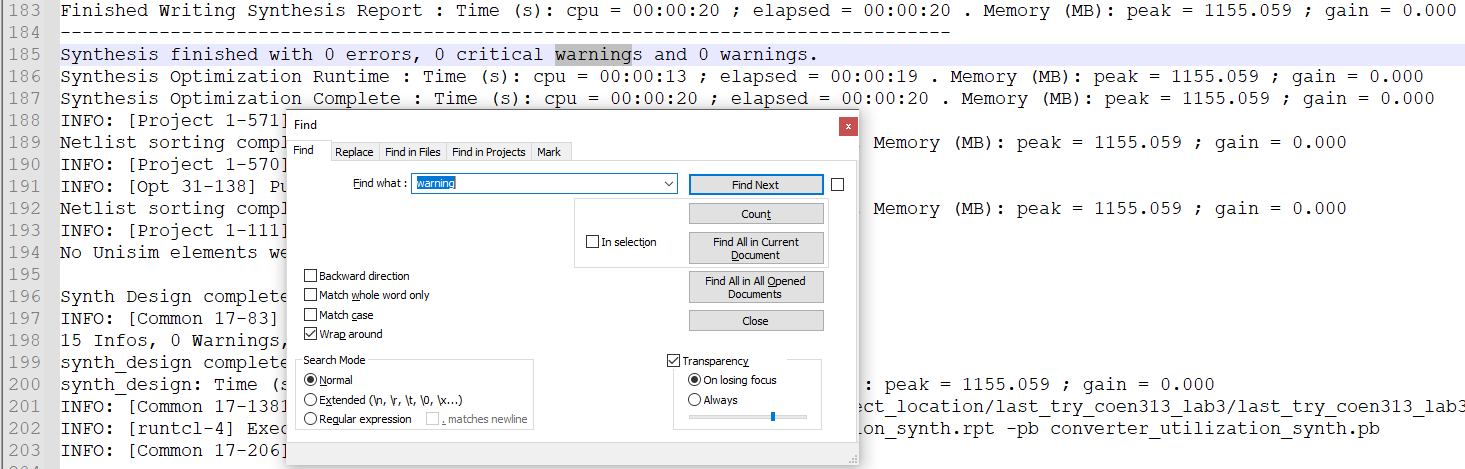


Figure 3.2.a. Warning of the Synthesis Log Files

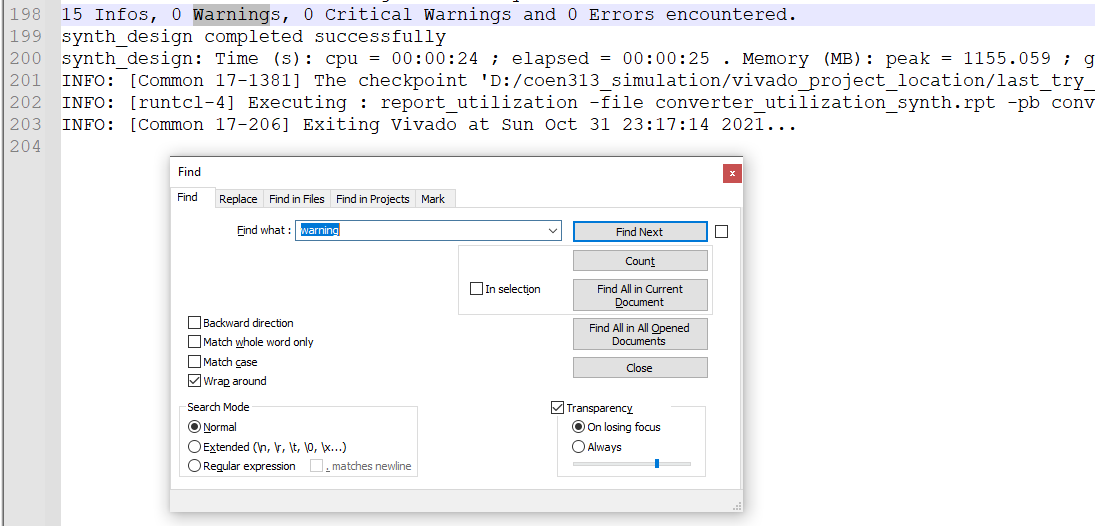


Figure 3.2.b. Warning of the Synthesis Log Files

The messages, errors and warnings generated by Vivado can be useful in pinpointing problems and help explaining why this area has an issue. These log files are found in the Vivado project directory in which the synthesis log file is under “/.runs/synth\_1/entity\_name.vds” while the implementation log file is under “/.runs/impl\_1/entity\_name.vdi”. The above figure 3.1 and 3.2 are some examples of warning messages encountered during implementation and synthesis, respectively. The importance of these generated messages or warning is to aide the user to locate potential problems and solve them easily by reading the type of error messages. In the long run, unresolved minor warnings can cause unwanted results, thus promoting the user to always solve their “warnings”, “critical warning” and “errors encountered”. The best-case scenario is when the log file indicates that there are zero problems such as the shown synthesis log file in figure 3.2.b.

**4) VHDL Code for my Designs**

4.1) VHDL Code Using Signals

library IEEE;

use IEEE.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity converter is

port ( sign\_mag : in std\_logic\_vector (3 downto 0);

twos\_comp : out std\_logic\_vector (3 downto 0));

end converter;

architecture if\_else of converter is

begin

process (sign\_mag)

begin

if sign\_mag(3) = '0' then

twos\_comp <= sign\_mag;

else

twos\_comp <= std\_logic\_vector(unsigned((not sign\_mag) + 1));

twos\_comp(3) <= '1';

end if;

end process;

end if\_else;

4.2) VHDL Code Using Variables

library IEEE;

use IEEE.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity converter is

port ( sign\_mag : in std\_logic\_vector (3 downto 0);

twos\_comp : out std\_logic\_vector (3 downto 0));

end converter;

architecture if\_else of converter is

begin

process (sign\_mag)

variable inv : unsigned(2 downto 0);

variable temp : std\_logic\_vector(2 downto 0);

variable first : std\_logic;

begin

if sign\_mag(3) = '0' then

first := sign\_mag(3);

temp := sign\_mag(2 downto 0);

else

first := sign\_mag(3);

inv := unsigned(not sign\_mag(2 downto 0));

inv := inv + 1;

temp := std\_logic\_vector(inv);

end if;

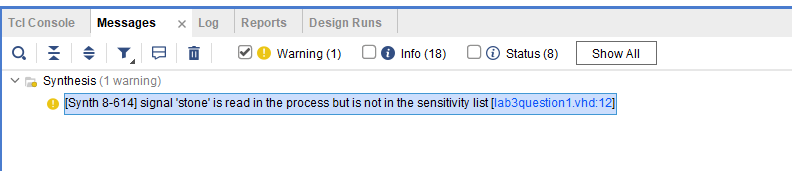
twos\_comp <= first & temp;

end process;

end if\_else;

**5) Questions**

**5.1) What will result (during synthesis) if a signal appears on both sides of the signal assignment operator (<=) within a combinational VHDL process**

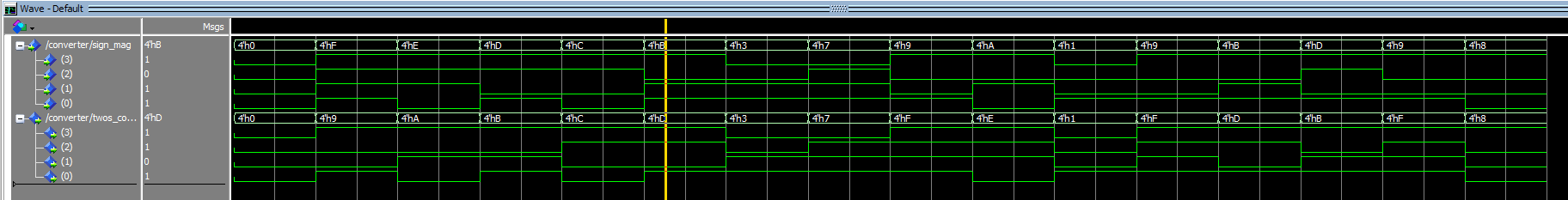


Given that a signal appears on both sides of the signal assignment operator within a combinational VHDL process, the combinational feedback will be inferred. In other words, during synthesis, it will be at risk of forming a closed feedback loop and the synthesis will assume the signal values. After synthesising the provided VHDL code as shown on the figure above, there are no warnings except mentioning that ‘stone’ isn’t in the sensitivity list. Since there are no other errors, it can support the argument that during synthesis, the signal value feedback will be inferred.

**5.2)** **What will happen during simulation if a signal is read from within a combinational process but does not appear in the process sensitivity list?**

When a signal is indicated in the sensitivity list, the process will evaluate its value whenever the signal changes at the end of the process. On the other hand, if the signal is not part of the sensitivity list, the value of the signal can change as many times without having the process re-evaluate what the new outputs should be. In addition, when a process has no sensitivity list, the process will continue to execute until it reaches a wait statement and it will suspend. Knowing this, during simulation, when a signal is read, but is not part of the sensitivity list, no change nor new value of the signal will be available at the end of the process, making the outputs unchanging despite the various combination of inputs.

**5.3) Simulation of VHDL Code Using Variables**



The above figure is the Modelsim simulated design of the VHDL circuit using only variables in which these results are the same as the one shown on figure 1.1. While there is no explicit difference between their output, the main difference lies in their usage and application in the VHDL code where variables can only be used inside the process while signals can also be used outside of them. Also, variables have a stricter syntax format where they need to be defined between “begin” and “process”, while signals can be defined in the architecture. Finally, the most obvious change is the assignment symbol where for variables, it is “:=”, while for signals, it is “<=”. Overall, while the final results are the same, the steps and process to reach them vary depending if the VHDL code was written using signals or variables.