به نام خدا

عنوان:

تکلیف شماره سوم درس طراحی سیستم های دیجیتال

استاد:

دکتر مروستی

اعضا:

محمدعلی مجتهدسلیمانی- 992023039

تاریخ:

5/03/1403

**Q1.** for this question first we need a counter that can handle a generic number of decimal digits. Then we use the “for-generate” statement to handle multiple digits. After that we create a test bench to verify 4bit counter. The entity will have this ingredient: CLK, RESET, ENABLE and COUNT. For implement this we need an array of signal to hold the value for each digit. Then we use “for-generate” statement. At the end, combine the individual digit values into the final COUNT output.

* **Code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity DecimalCounter is

generic (

NUM\_DIGITS : integer := 4 -- Number of digits

);

Port (

clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

en : in STD\_LOGIC;

count : out STD\_LOGIC\_VECTOR((NUM\_DIGITS\*4)-1 downto 0) -- Output for the counter

);

end DecimalCounter;

architecture Behavioral of DecimalCounter is

type digit\_array is array (NUM\_DIGITS-1 downto 0) of STD\_LOGIC\_VECTOR(3 downto 0); -- Array for digits

signal digit\_values : digit\_array := (others => (others => '0'));

signal carry\_signal : STD\_LOGIC\_VECTOR(NUM\_DIGITS downto 0) := (others => '0');

begin

process(clk, rst)

begin

if rst = '1' then

digit\_values <= (others => (others => '0')); -- Reset all digits

carry\_signal <= (others => '0');

elsif rising\_edge(clk) then

if en = '1' then

carry\_signal(0) <= '1'; -- Initial carry to start counting

for i in 0 to NUM\_DIGITS-1 loop

if carry\_signal(i) = '1' then

if digit\_values(i) = "1001" then

digit\_values(i) <= "0000";

carry\_signal(i+1) <= '1';

else

digit\_values(i) <= digit\_values(i) + 1;

carry\_signal(i+1) <= '0';

end if;

end if;

end loop;

end if;

end if;

end process;

-- Concatenate digits into count

process(digit\_values)

begin

for i in 0 to NUM\_DIGITS-1 loop

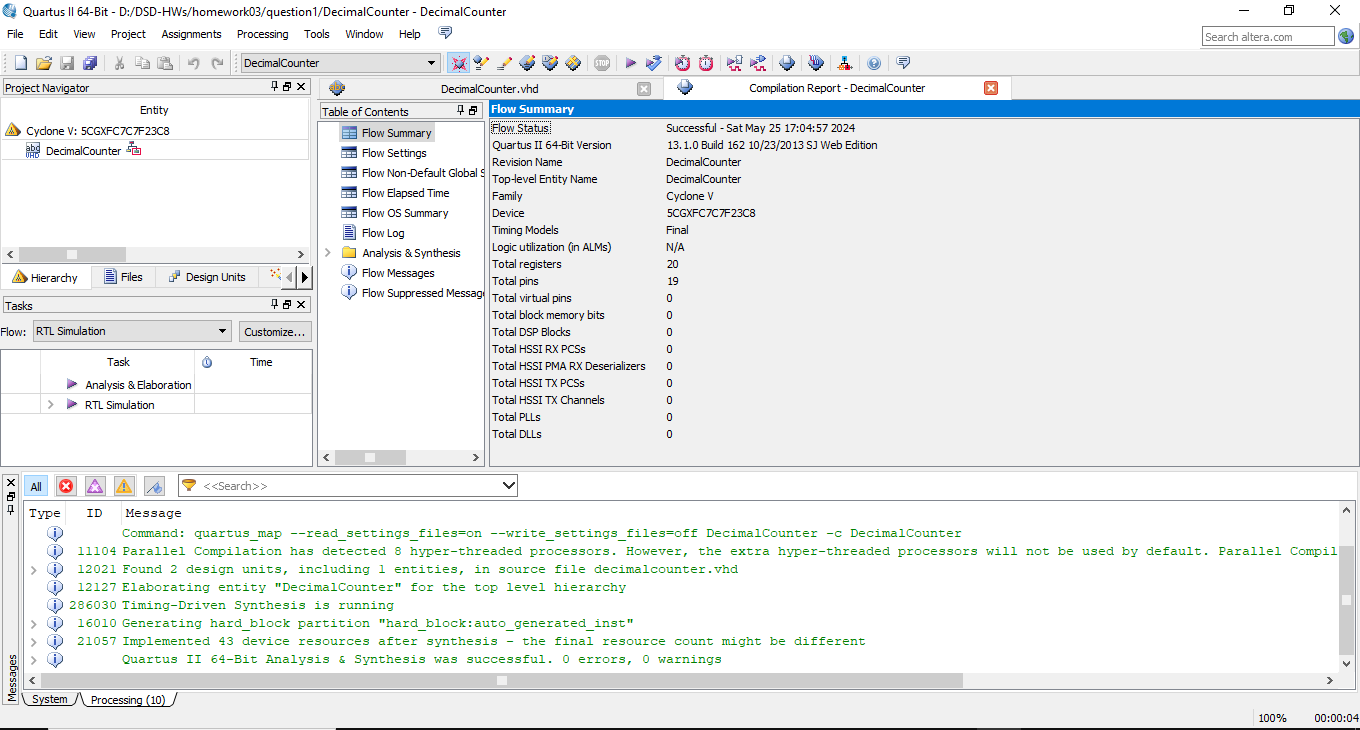
count((i\*4)+3 downto i\*4) <= digit\_values(i);

end loop;

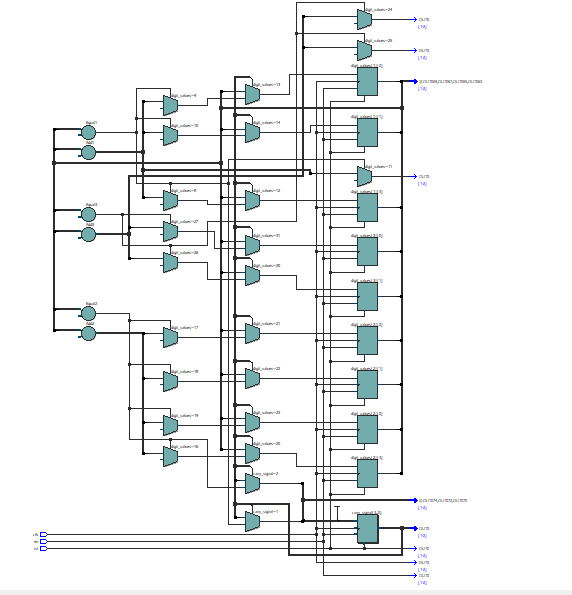
end process;

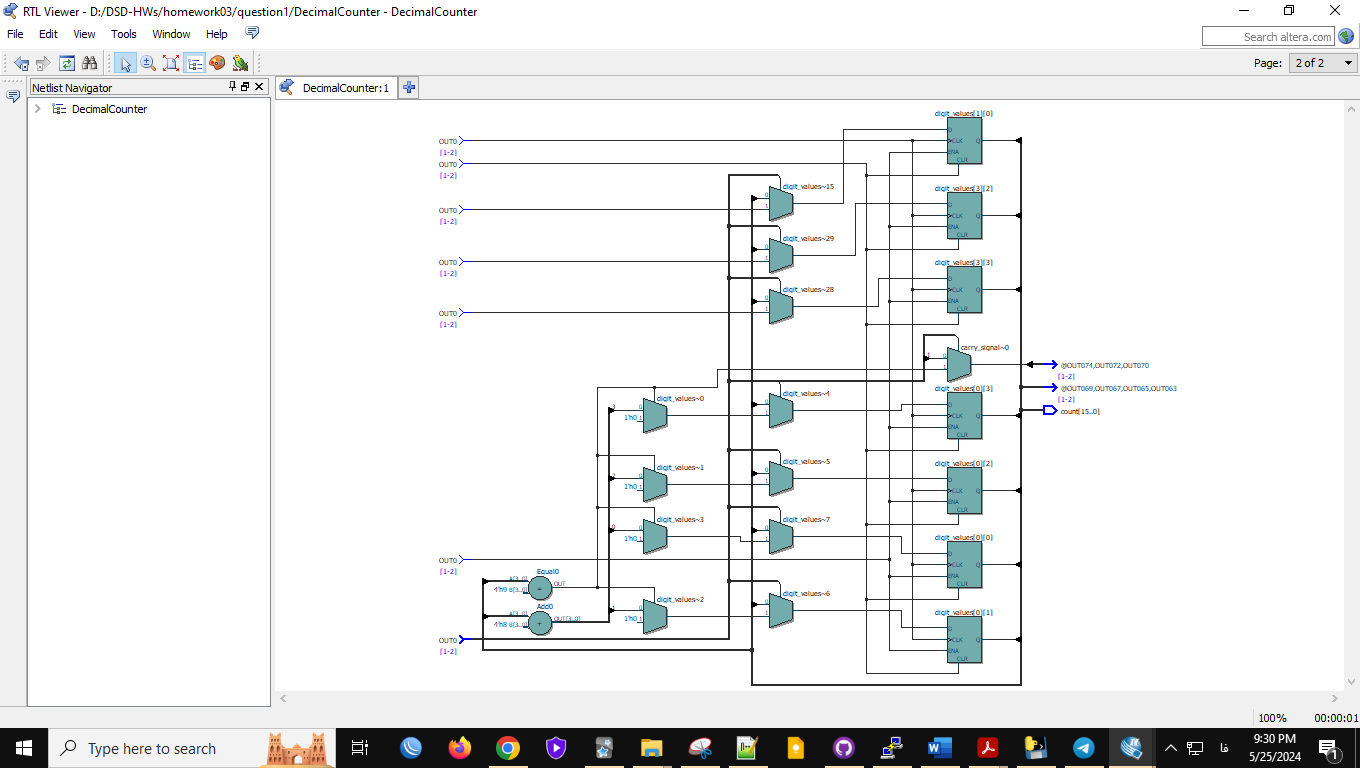
end Behavioral;

* **Compilation Report:**

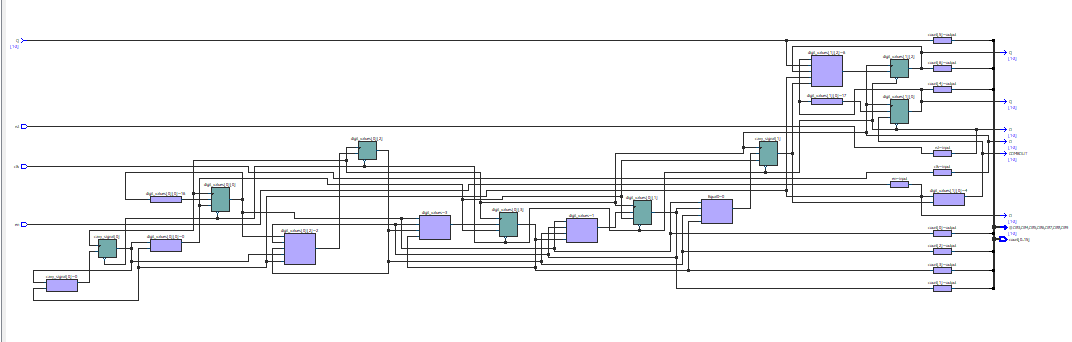


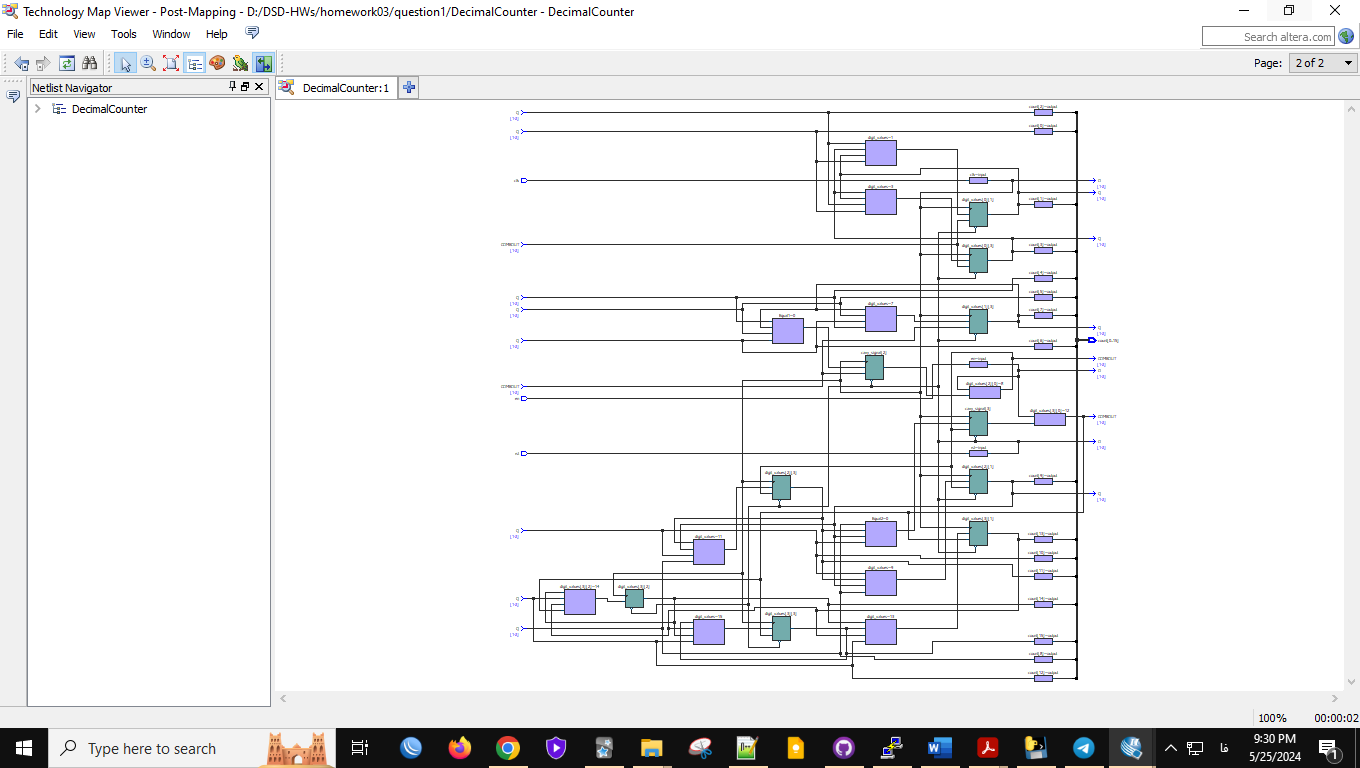
* **RTL View:**



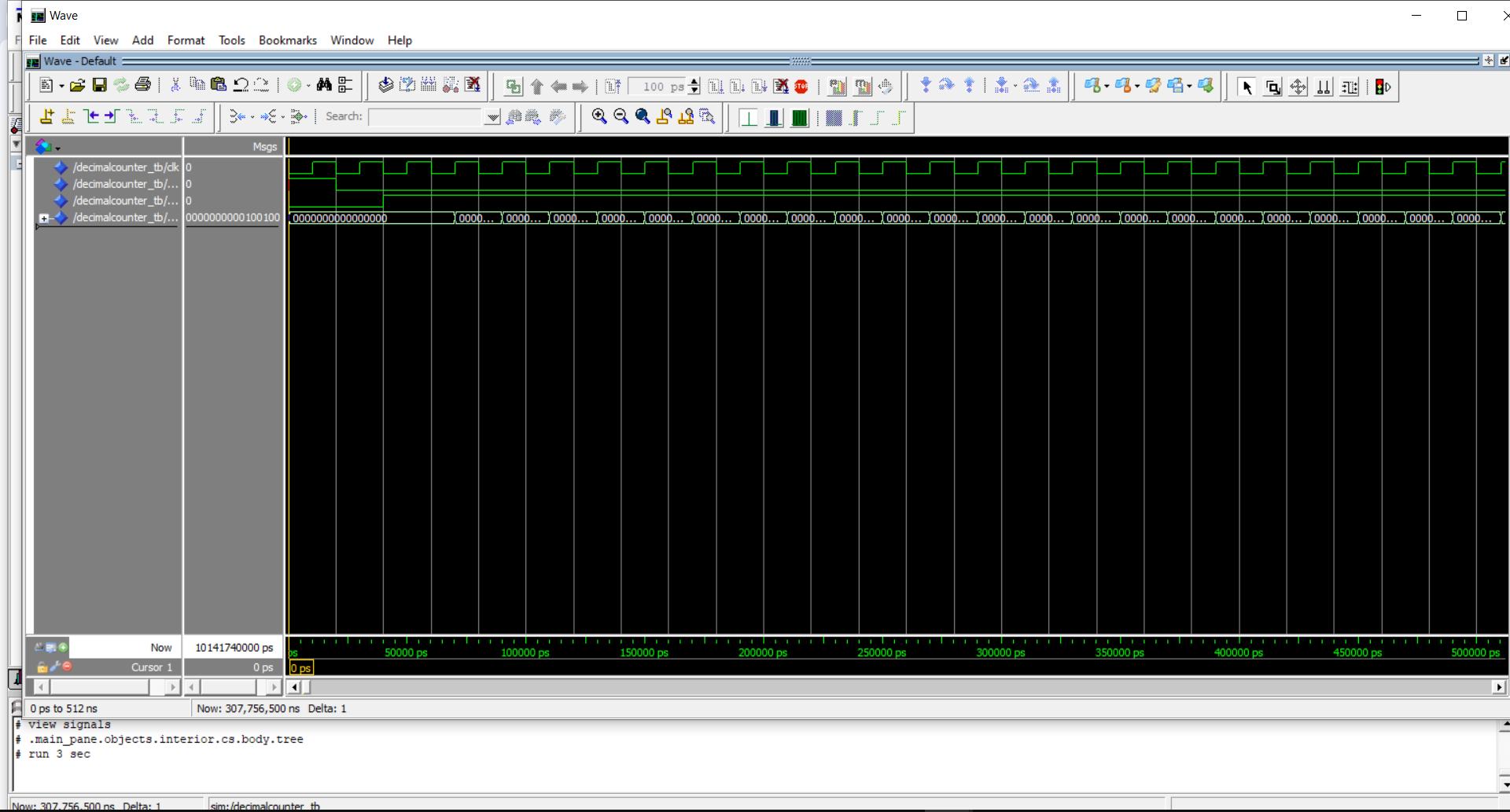


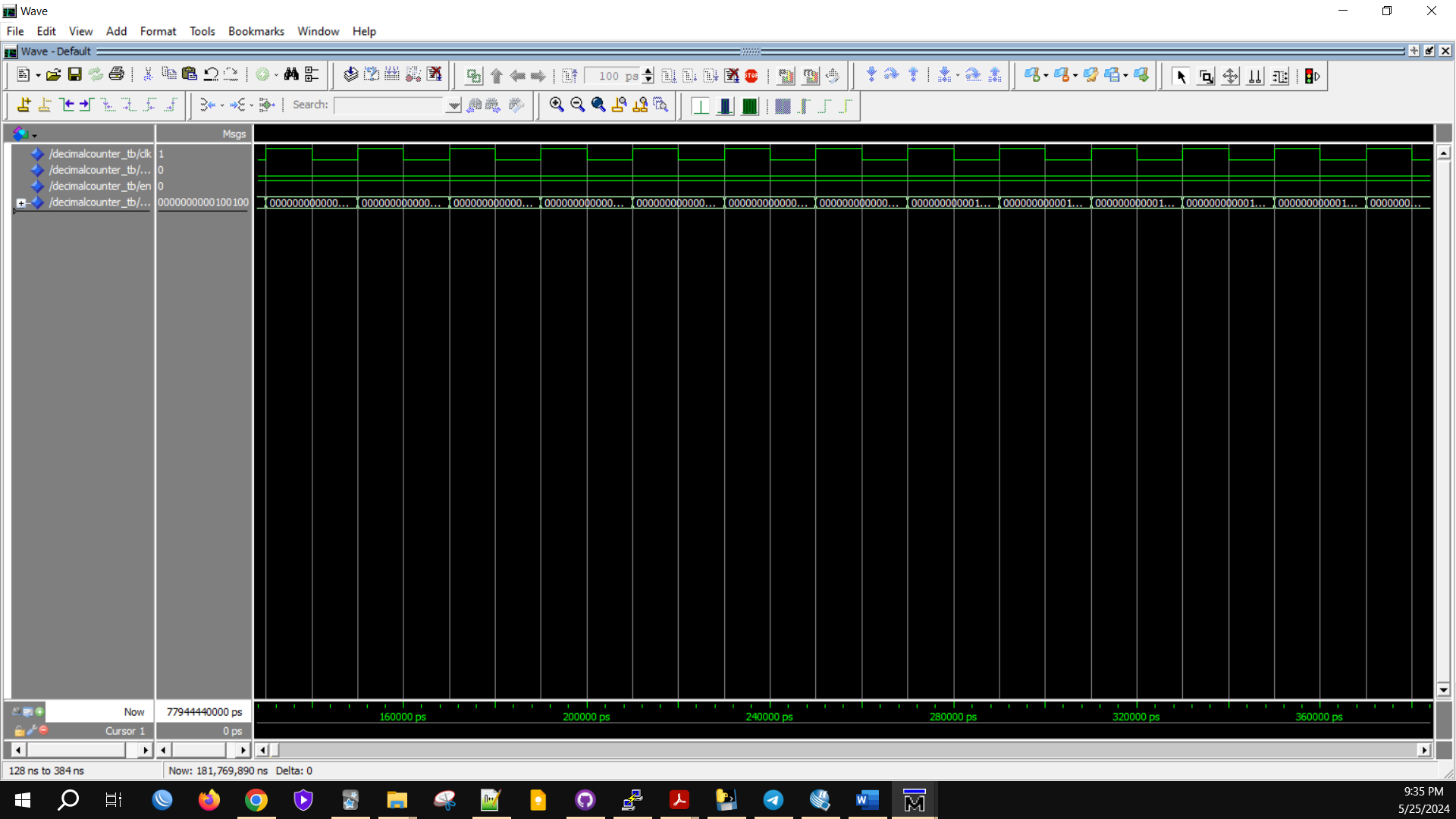
* **Post-mapping:**

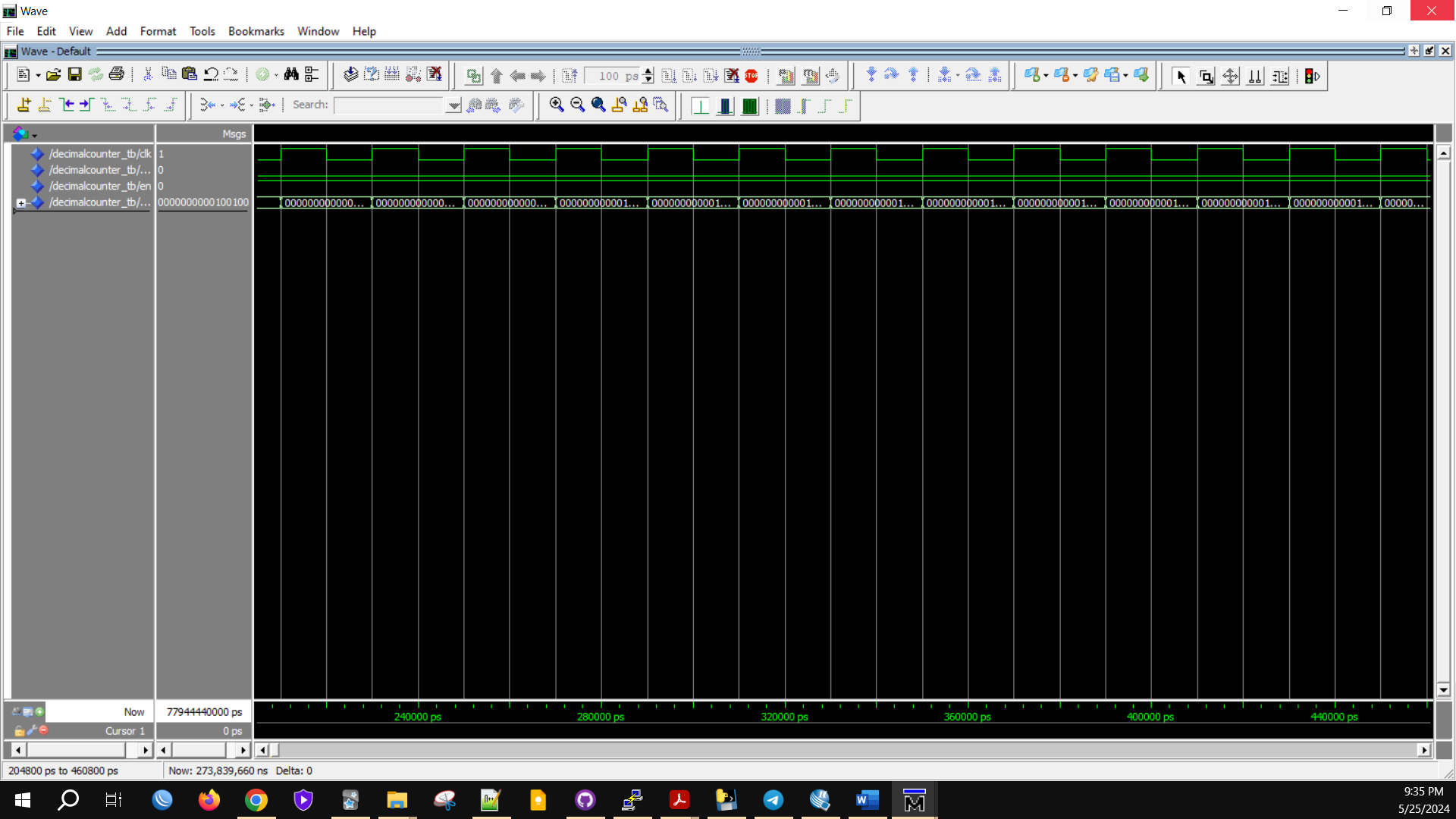


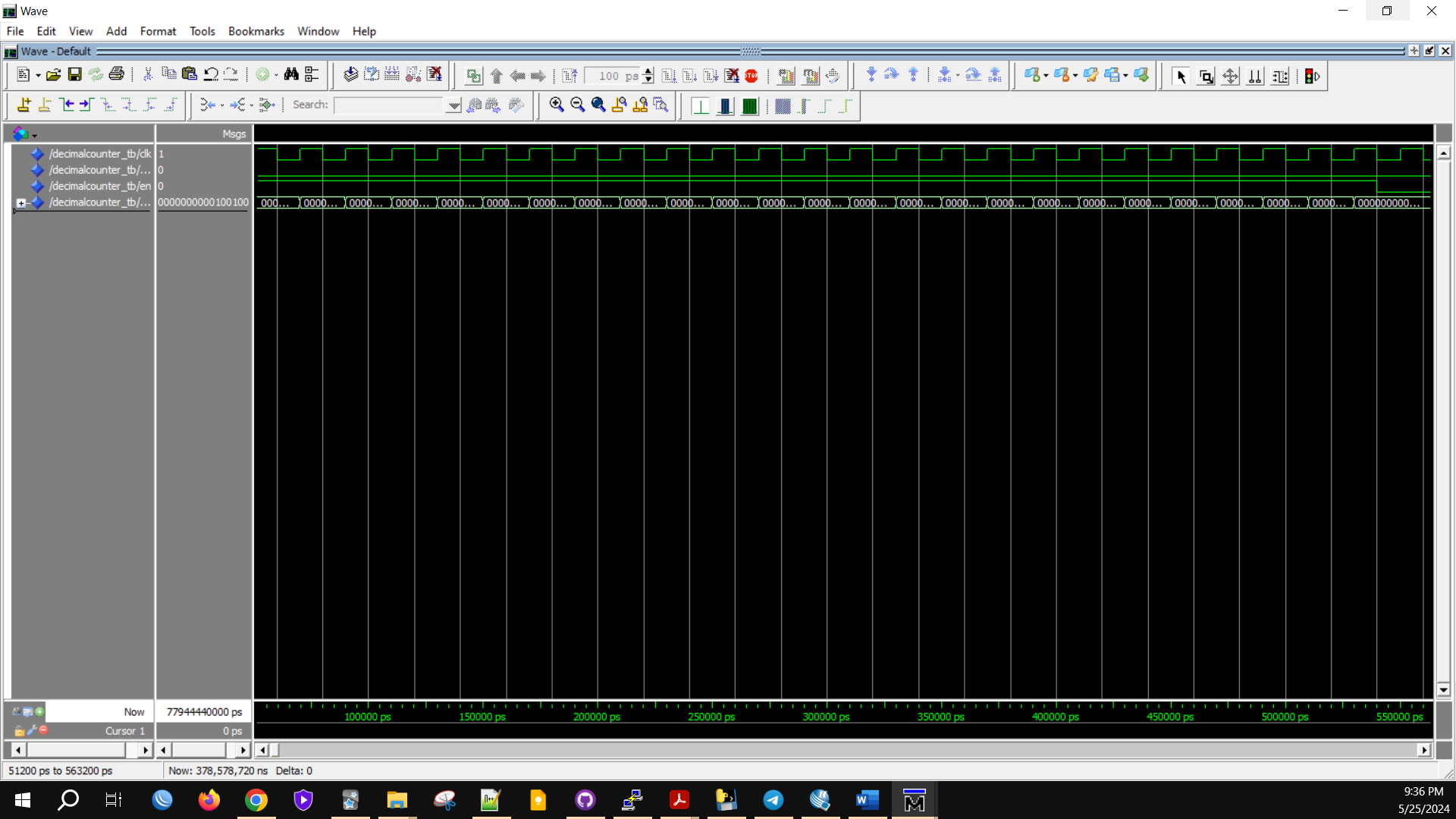


* **Wave Form:**









* **Test bench code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity DecimalCounter\_tb is

end DecimalCounter\_tb;

architecture Behavioral of DecimalCounter\_tb is

constant NUM\_DIGITS : integer := 4; -- Number of digits for testing

signal clk : STD\_LOGIC := '0';

signal rst : STD\_LOGIC := '0';

signal en : STD\_LOGIC := '0';

signal count : STD\_LOGIC\_VECTOR((NUM\_DIGITS\*4)-1 downto 0);

component DecimalCounter

generic (

NUM\_DIGITS : integer

);

Port (

clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

en : in STD\_LOGIC;

count : out STD\_LOGIC\_VECTOR((NUM\_DIGITS\*4)-1 downto 0)

);

end component;

begin

UUT: DecimalCounter

generic map (

NUM\_DIGITS => 4 -- Specify 4 digits

)

port map (

clk => clk,

rst => rst,

en => en,

count => count

);

-- Clock generation

clk\_process: process

begin

while True loop

clk <= '0';

wait for 10 ns;

clk <= '1';

wait for 10 ns;

end loop;

end process;

-- Stimulus process

stimulus: process

begin

-- Apply reset

rst <= '1';

wait for 20 ns;

rst <= '0';

wait for 20 ns;

-- Enable counting

en <= '1';

wait for 500 ns;

-- Disable counting

en <= '0';

wait for 20 ns;

wait;

end process;

end Behavioral;

**Q2.** So for this question we should design a FIFO buffer for 64-bit system. We need 8 slots, with each slot capable of holding an 8-byte (64 bit) word. Two main components: Register File and FIFO Controller. It should handle write “WR” and read “RD” operations. The FIFO should be able to indicate when it is full “FULL” or empty “EMPTY”.

* **For this FIFO buffer:**
* First, we define entity:

We need these ports: “CLK”, “RESET”, “WR”, “RD”, “W\_DATA”, “R\_DATA”, “FULL” AND “EMPTY”

* **For Register File:**

We need an array of 8 slots, each 64 bits wide.

And for FIFO Controller we need handle read and write signals.

* **Code:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity fifo\_Buffer is

port (

clk: in std\_logic;

reset: in std\_logic;

wr: in std\_logic;

rd: in std\_logic;

w\_data: in std\_logic\_vector (63 downto 0);

r\_data: out std\_logic\_vector (63 downto 0);

full: out std\_logic;

empty: out std\_logic

);

end fifo\_Buffer;

architecture Behavioral of fifo\_Buffer is

type mem\_type is array (0 to 7) of std\_logic\_vector(63 downto 0);

signal mem: mem\_type := (others => (others => '0'));

signal w\_ptr, r\_ptr: integer range 0 to 7 := 0;

signal count: integer range 0 to 8 := 0;

signal full\_signal, empty\_signal: std\_logic;

begin

full <= full\_signal;

empty <= empty\_signal;

process(clk, reset)

begin

if reset = '1' then

w\_ptr <= 0;

r\_ptr <= 0;

count <= 0;

full\_signal <= '0';

empty\_signal <= '1';

elsif rising\_edge(clk) then

if wr = '1' and full\_signal = '0' then

mem(w\_ptr) <= w\_data;

w\_ptr <= (w\_ptr + 1) mod 8;

count <= count + 1;

end if;

if rd = '1' and empty\_signal = '0' then

r\_data <= mem(r\_ptr);

r\_ptr <= (r\_ptr + 1) mod 8;

count <= count - 1;

end if;

if count = 8 then

full\_signal <= '1';

else

full\_signal <= '0';

end if;

if count = 0 then

empty\_signal <= '1';

else

empty\_signal <= '0';

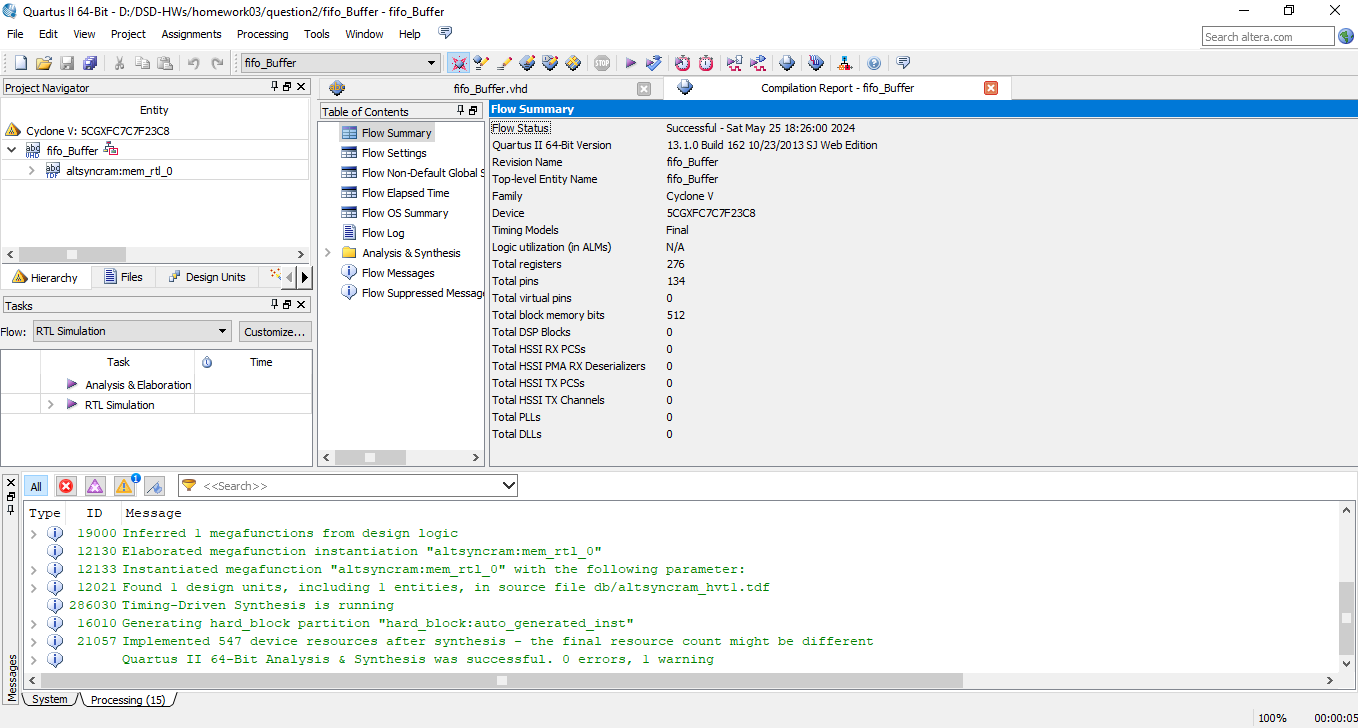
end if;

end if;

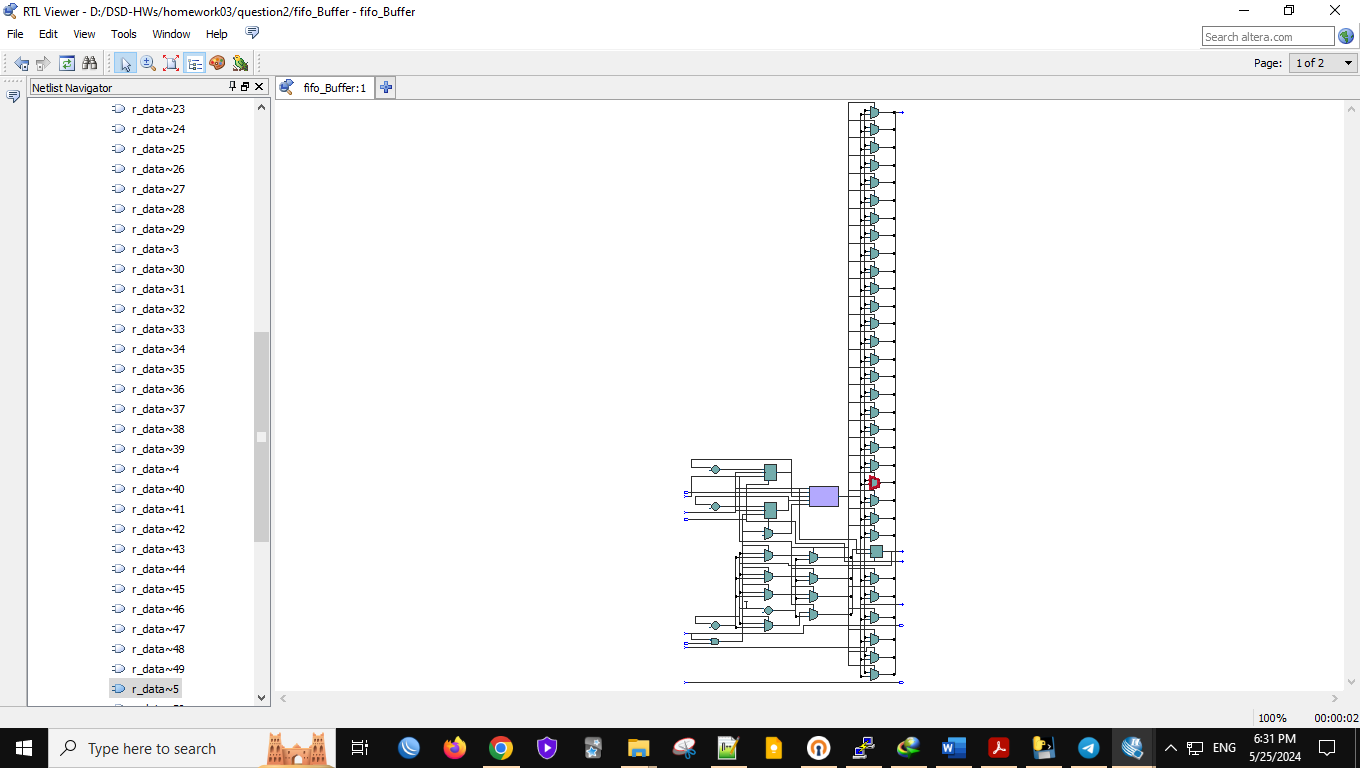
end process;

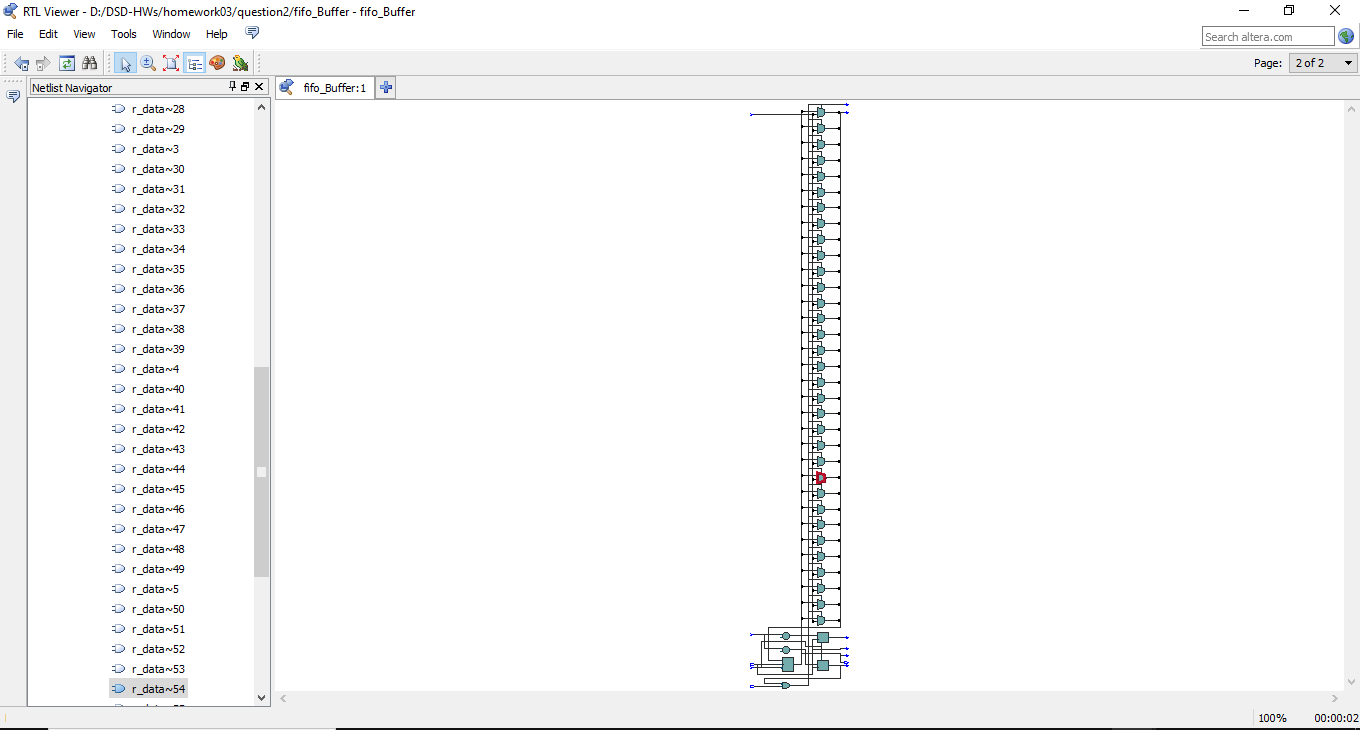
end Behavioral;

* **Compilation Report:**



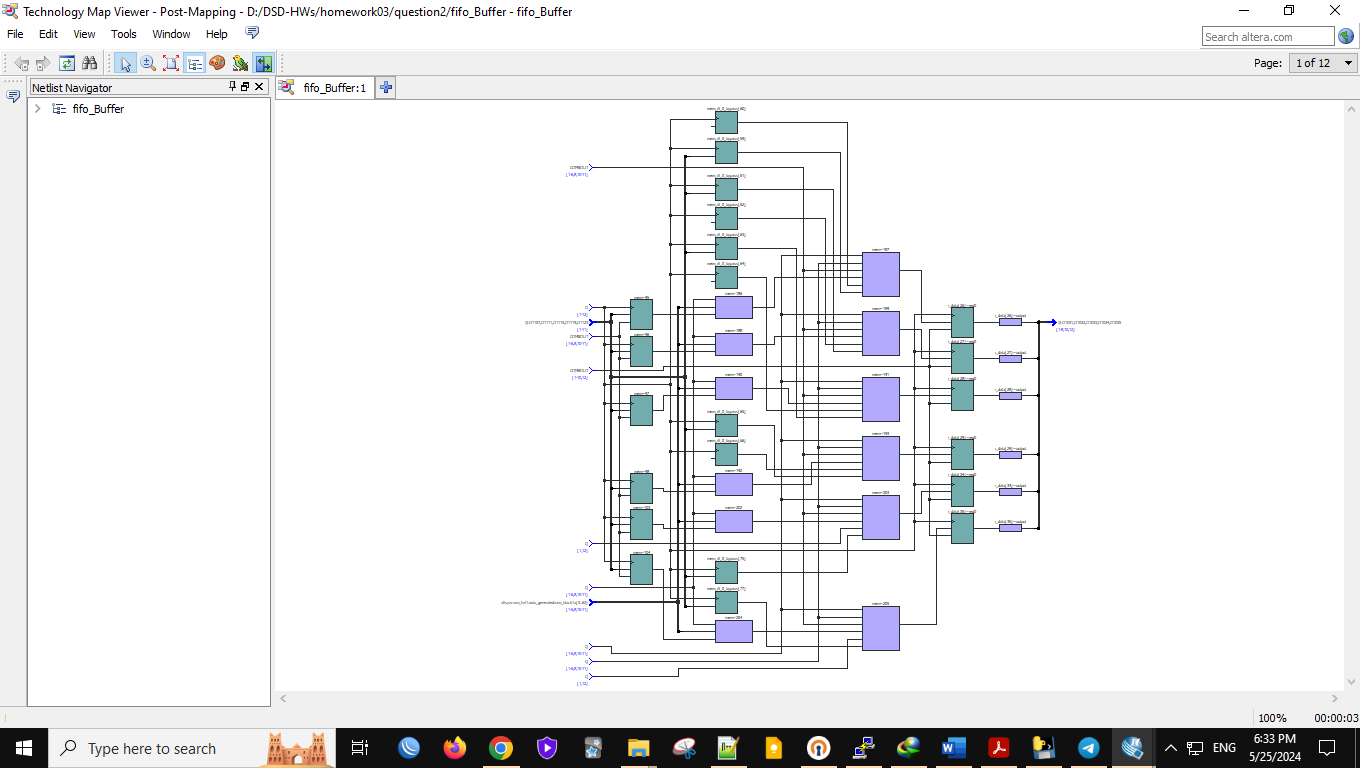
* **RTL View:**



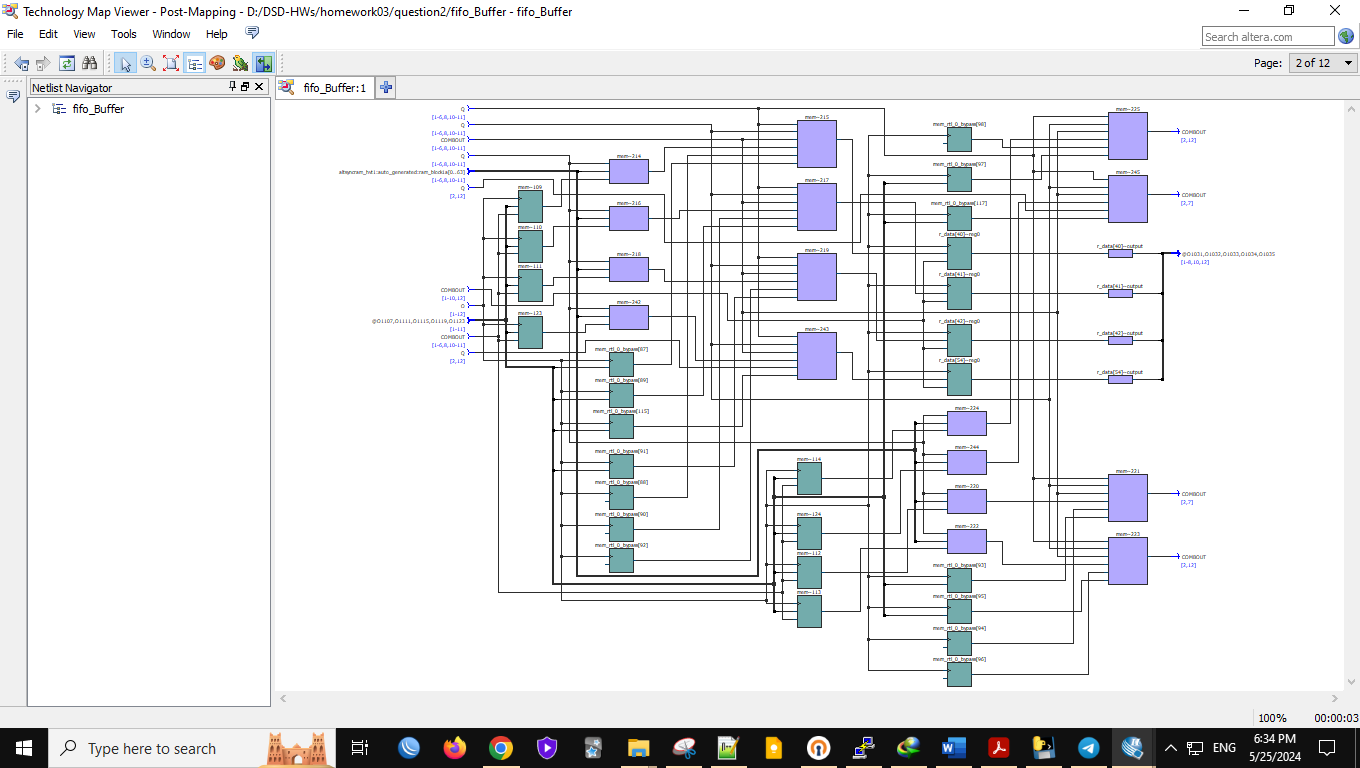


**Post-Mapping View:**

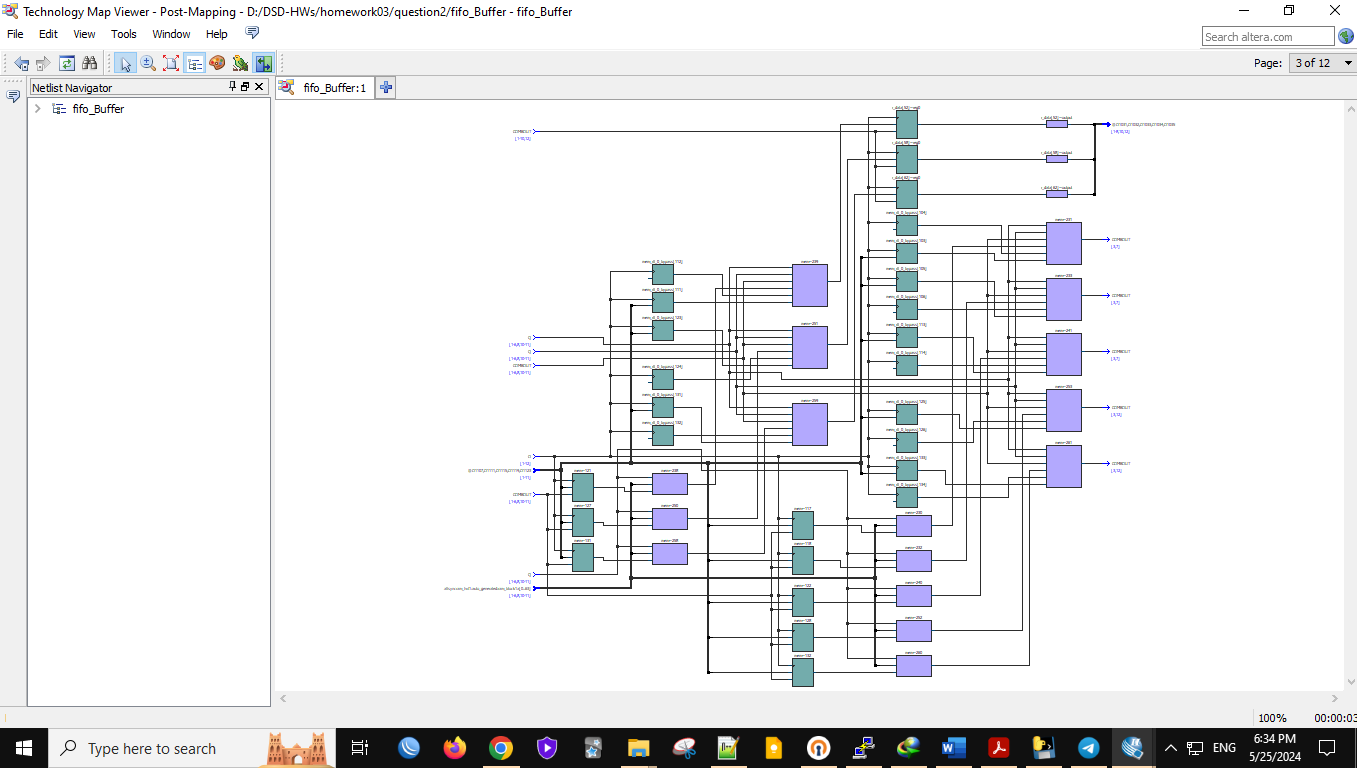
1:



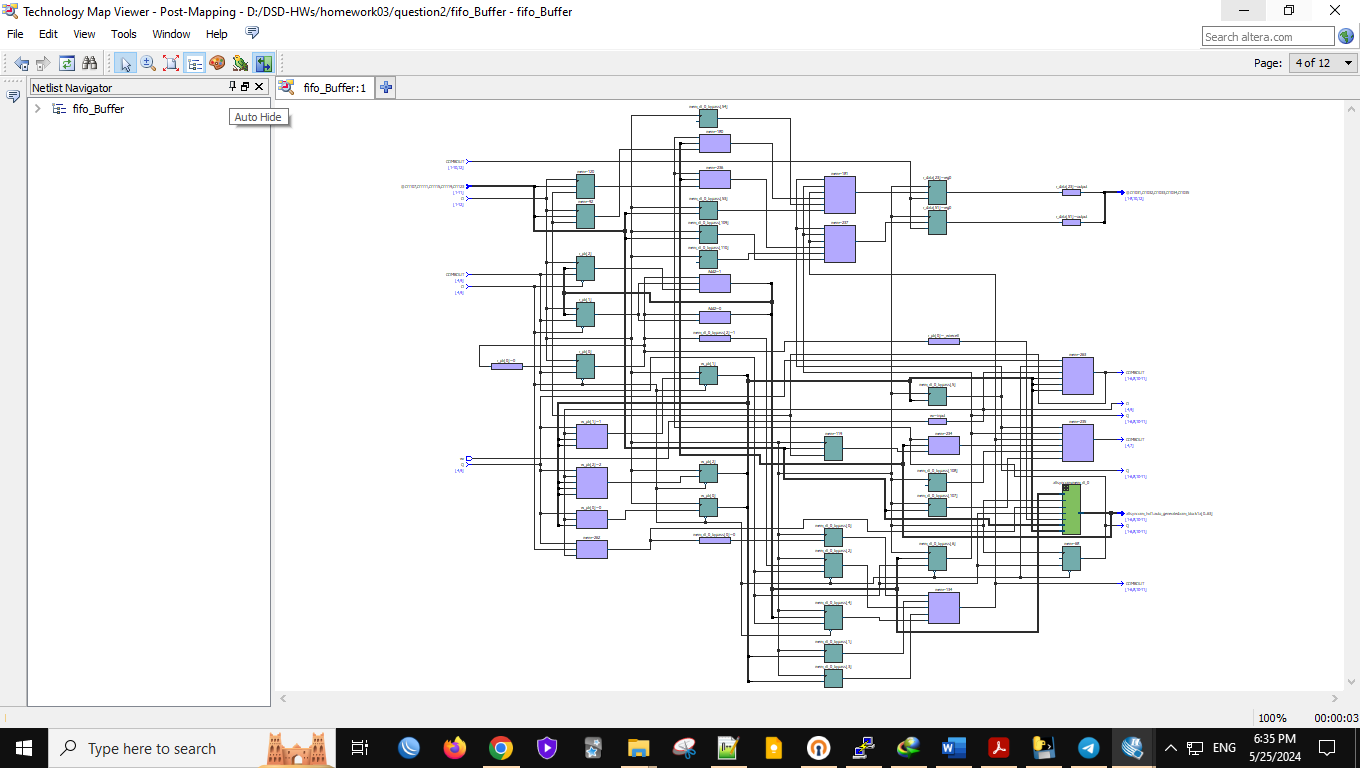
2:



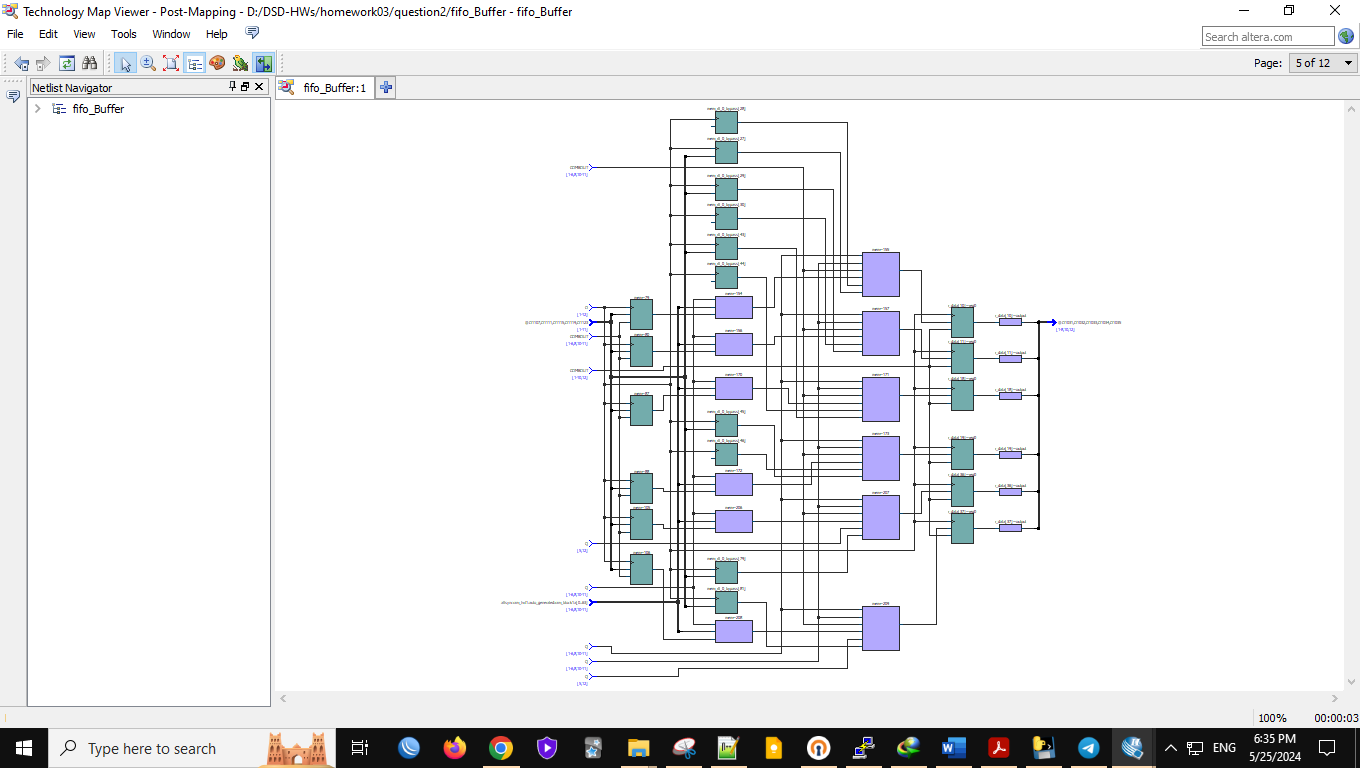
3:



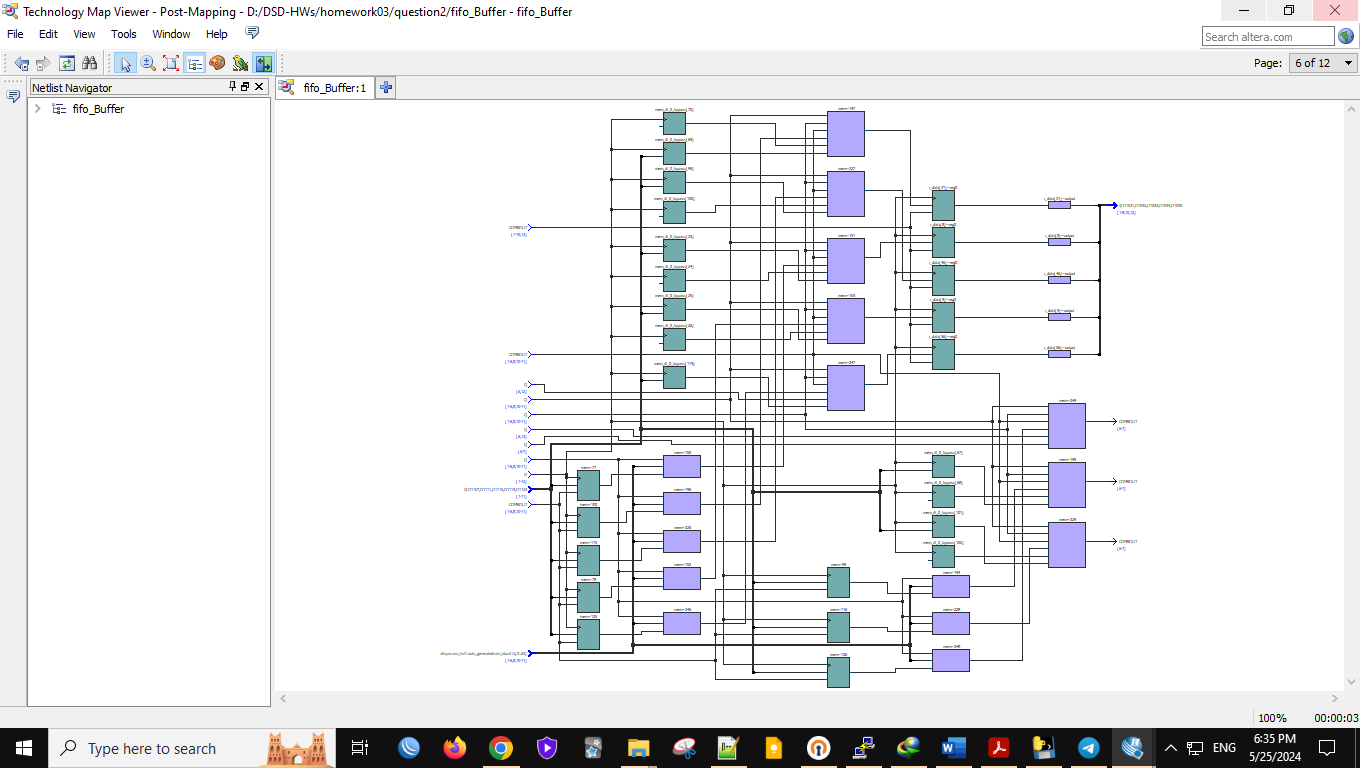
4:



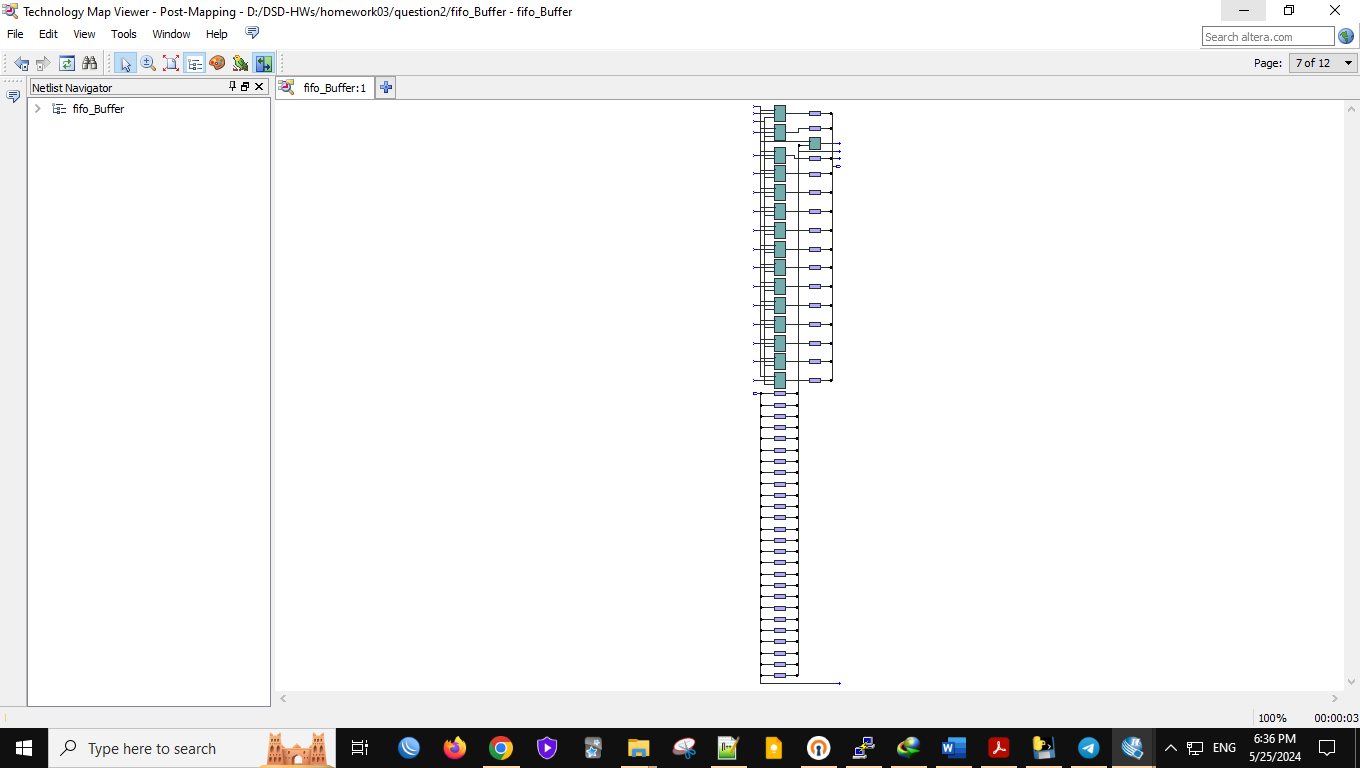
5:



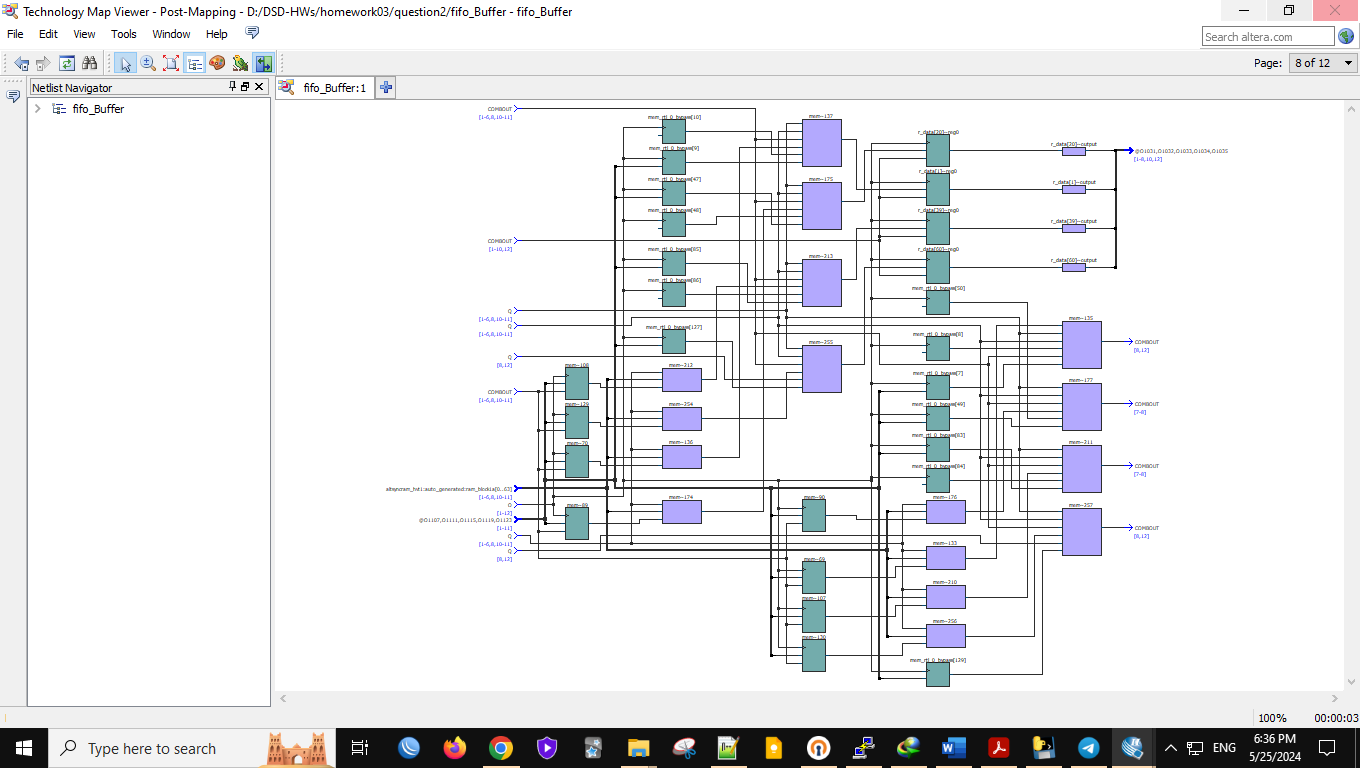
6:



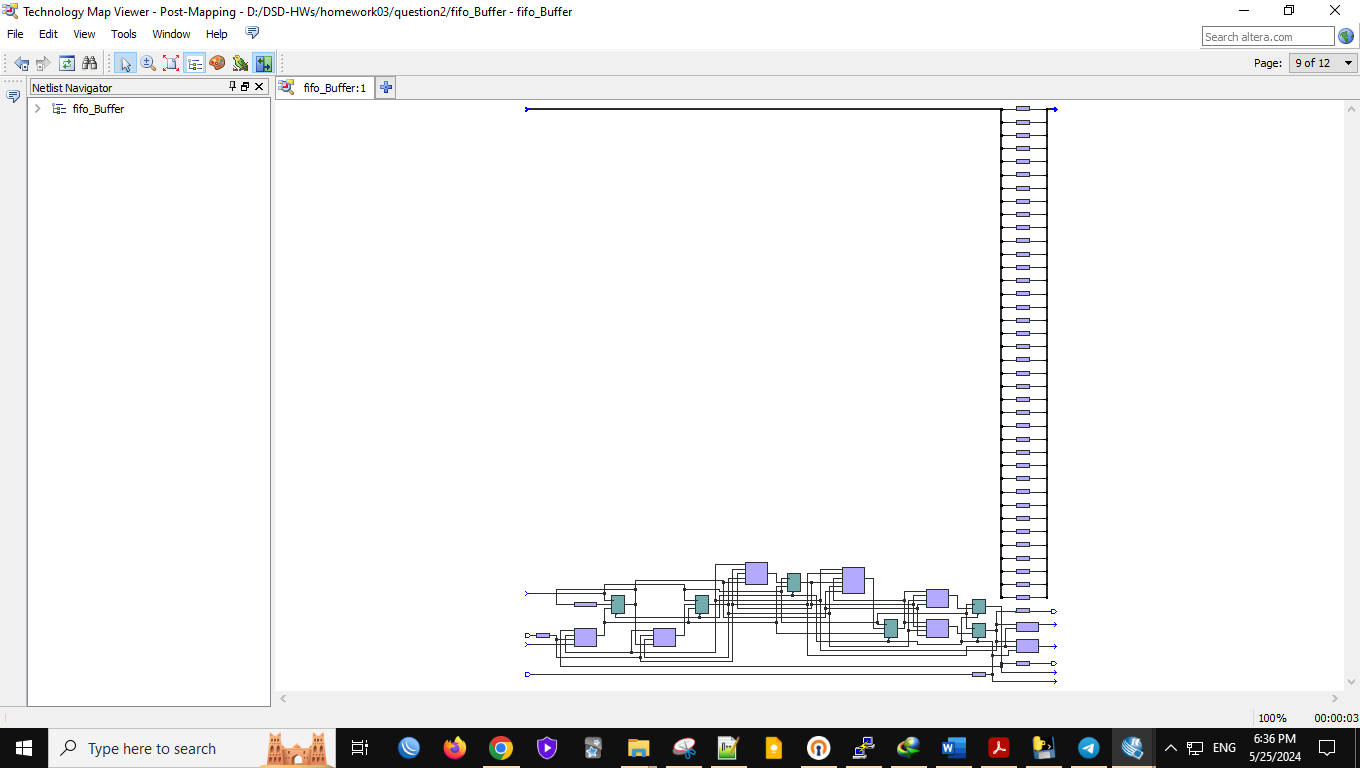
7:



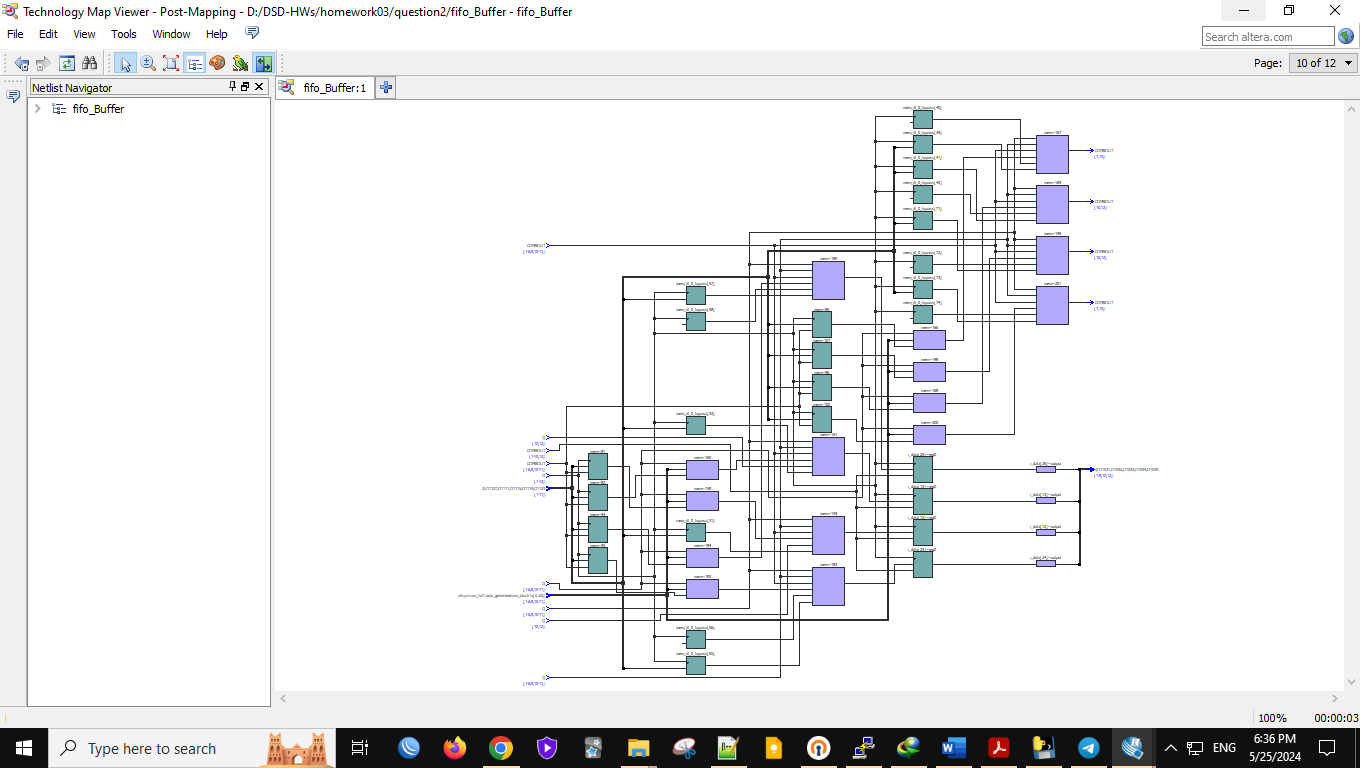
8:



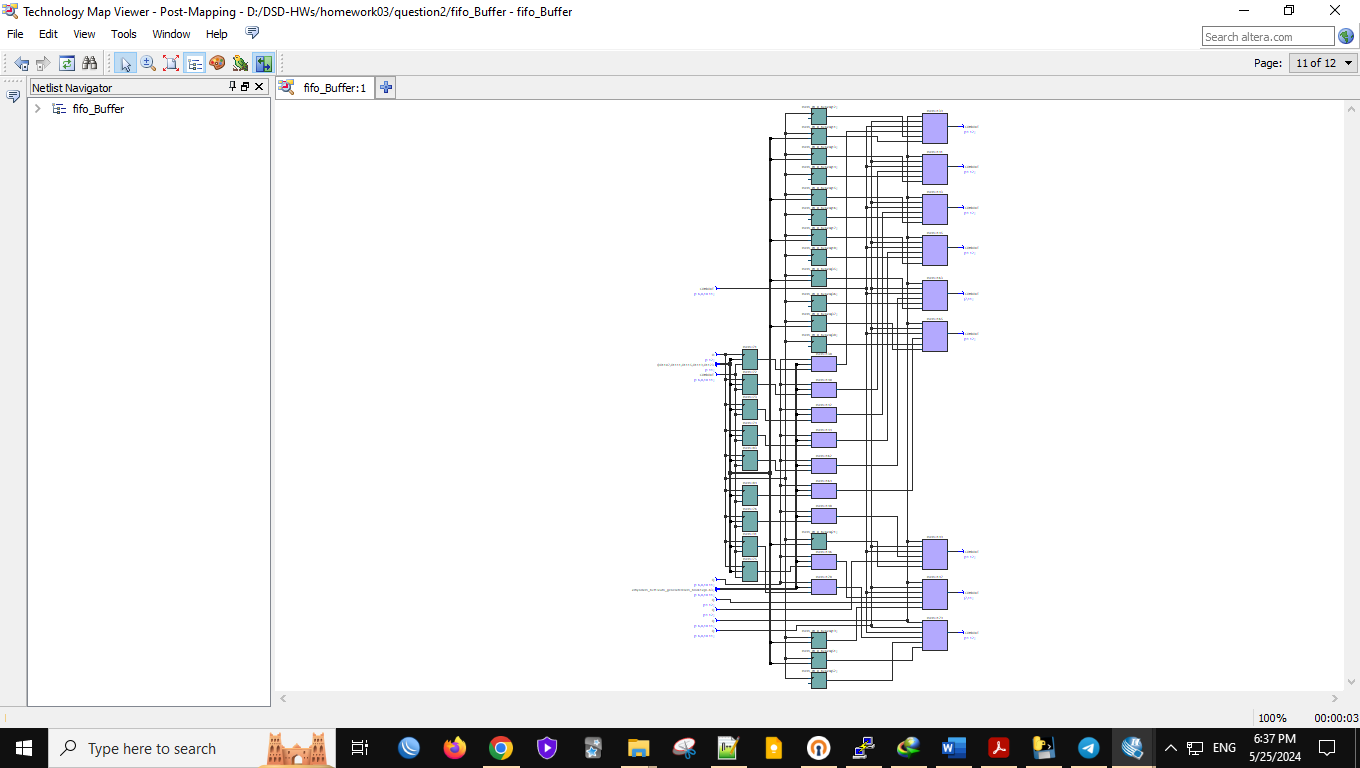
9:



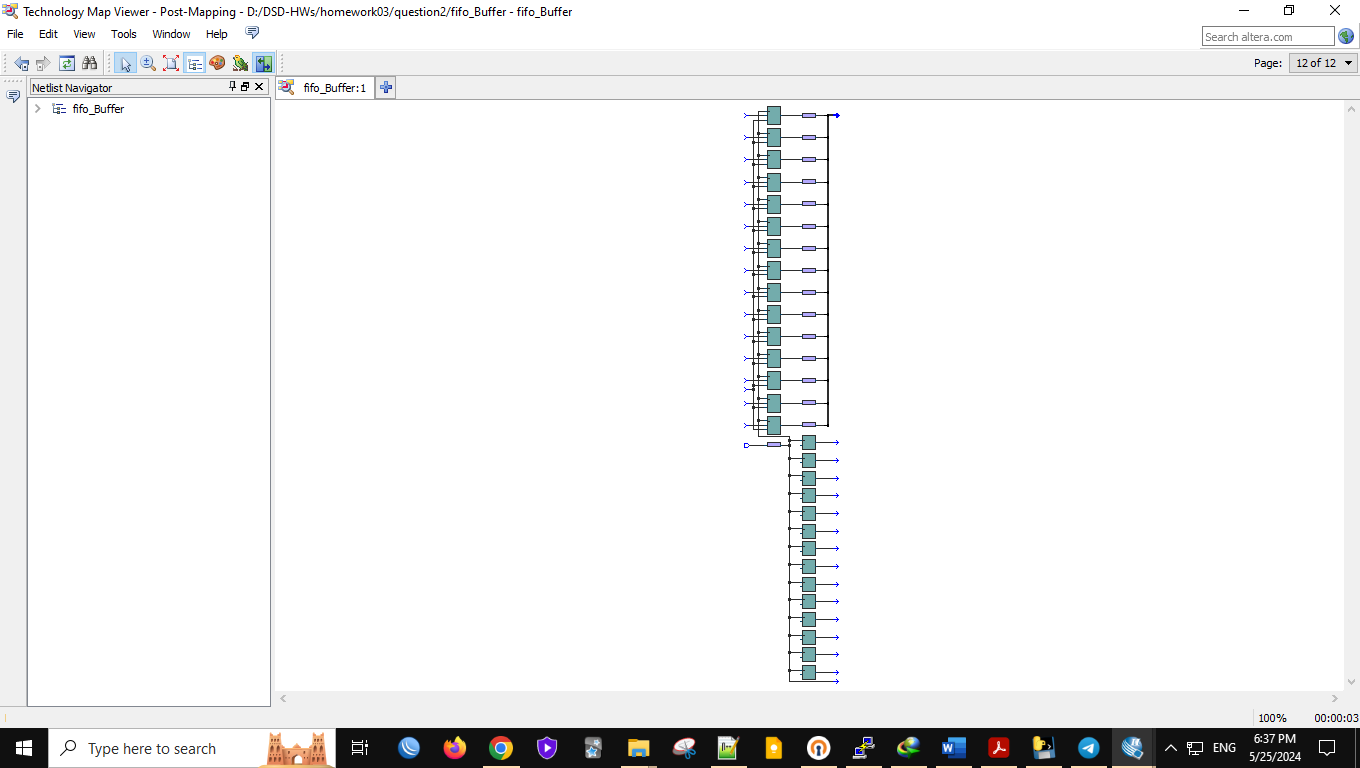
10:



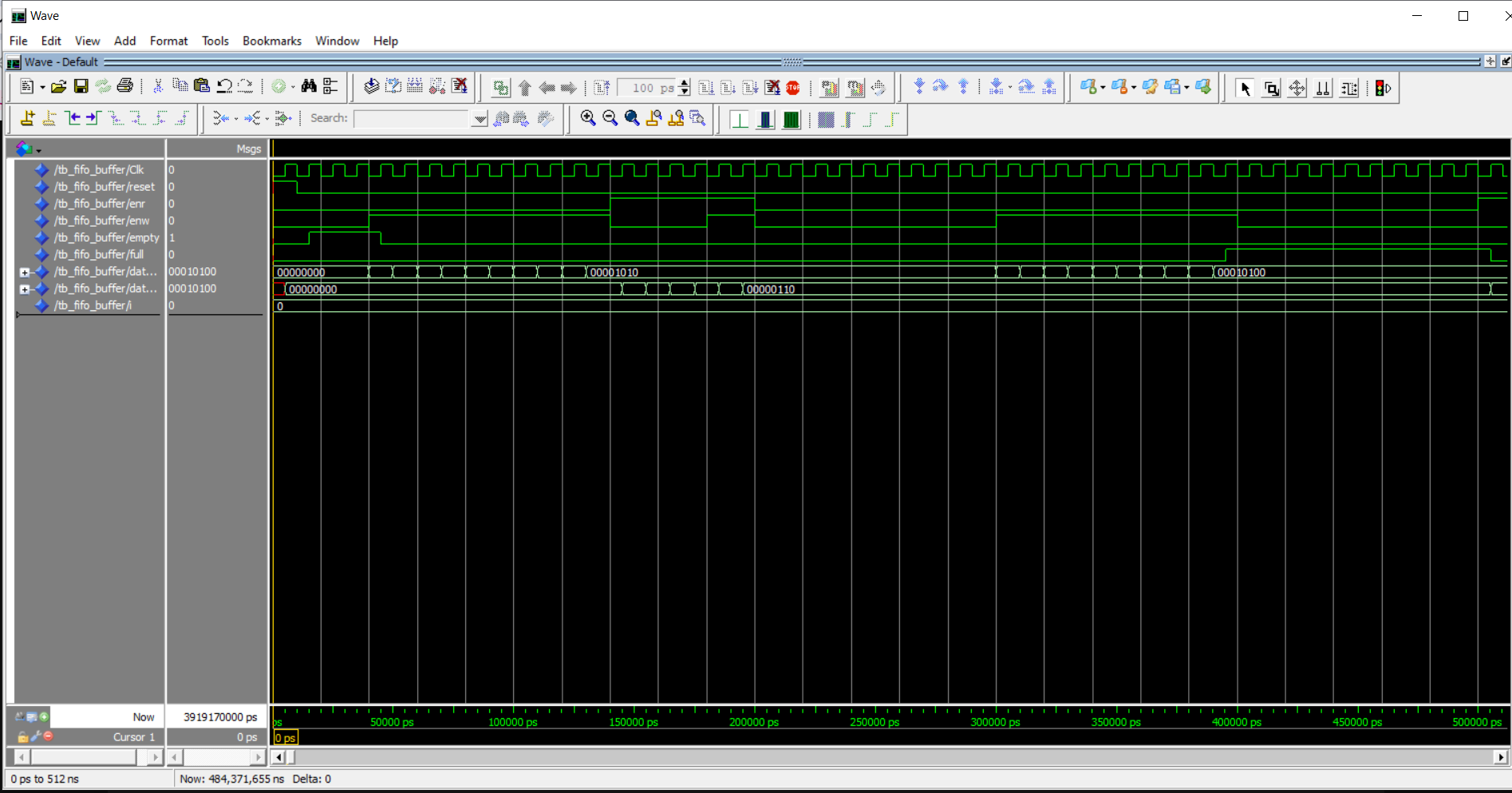
11:

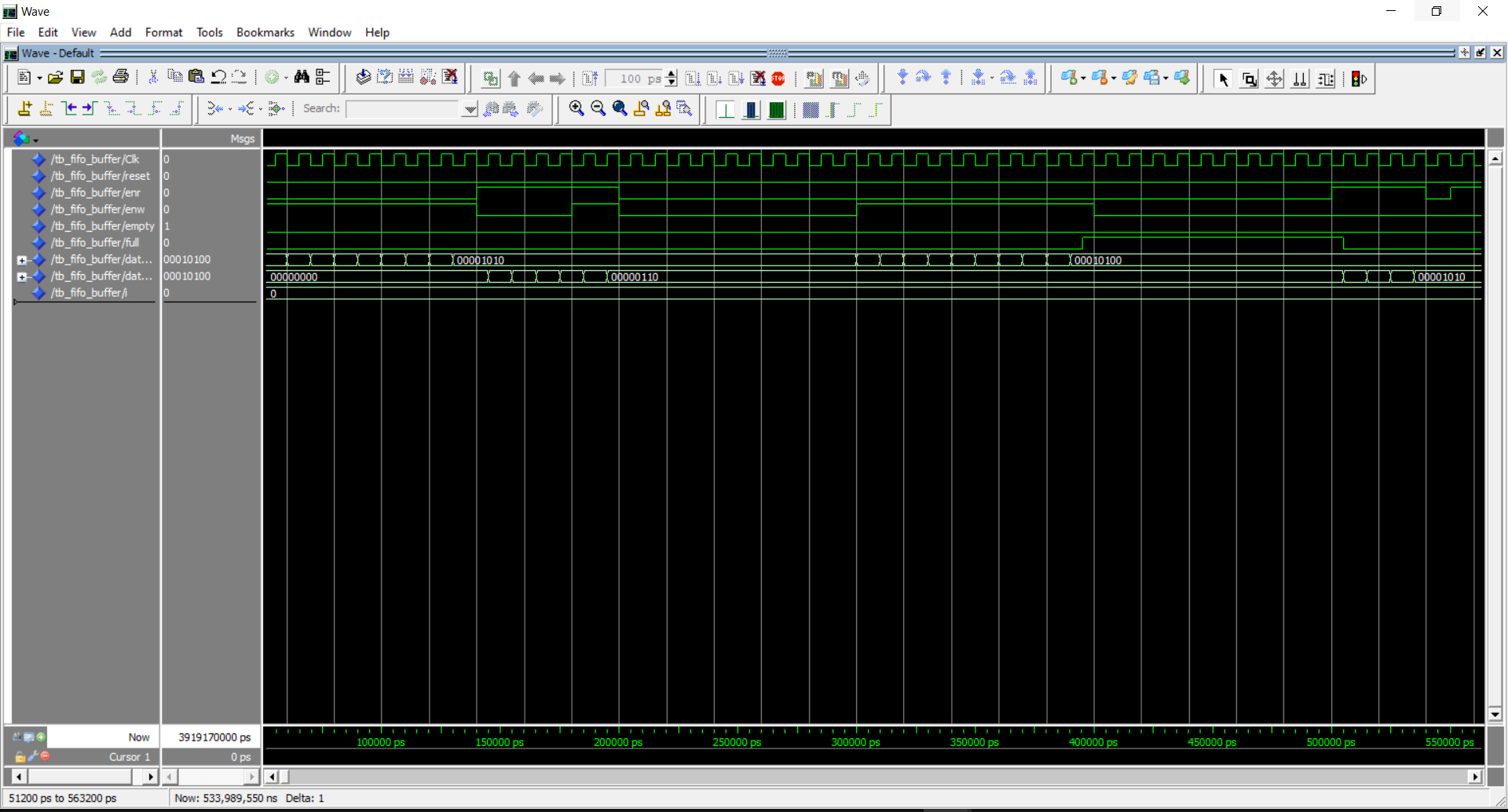


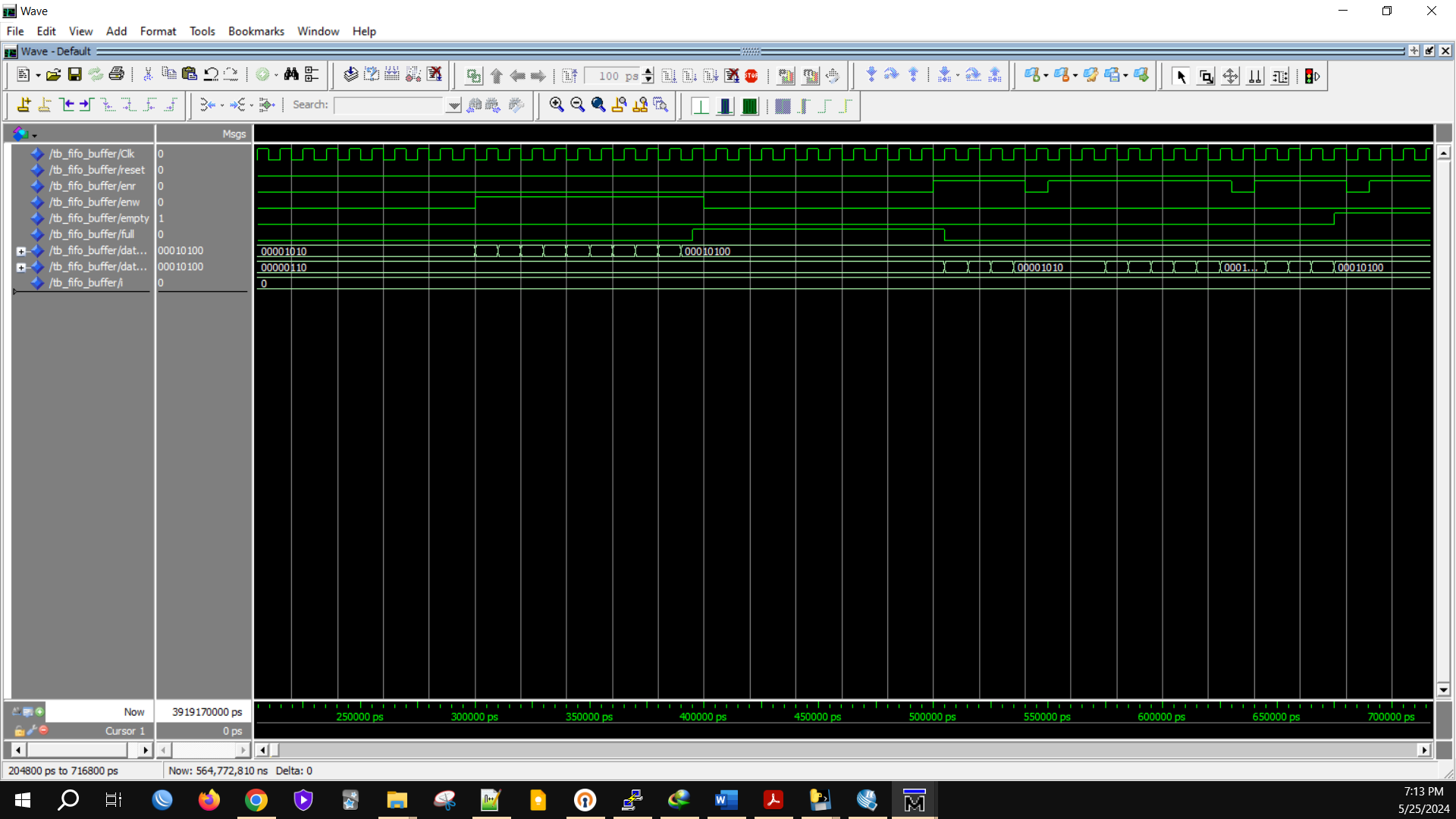
12:

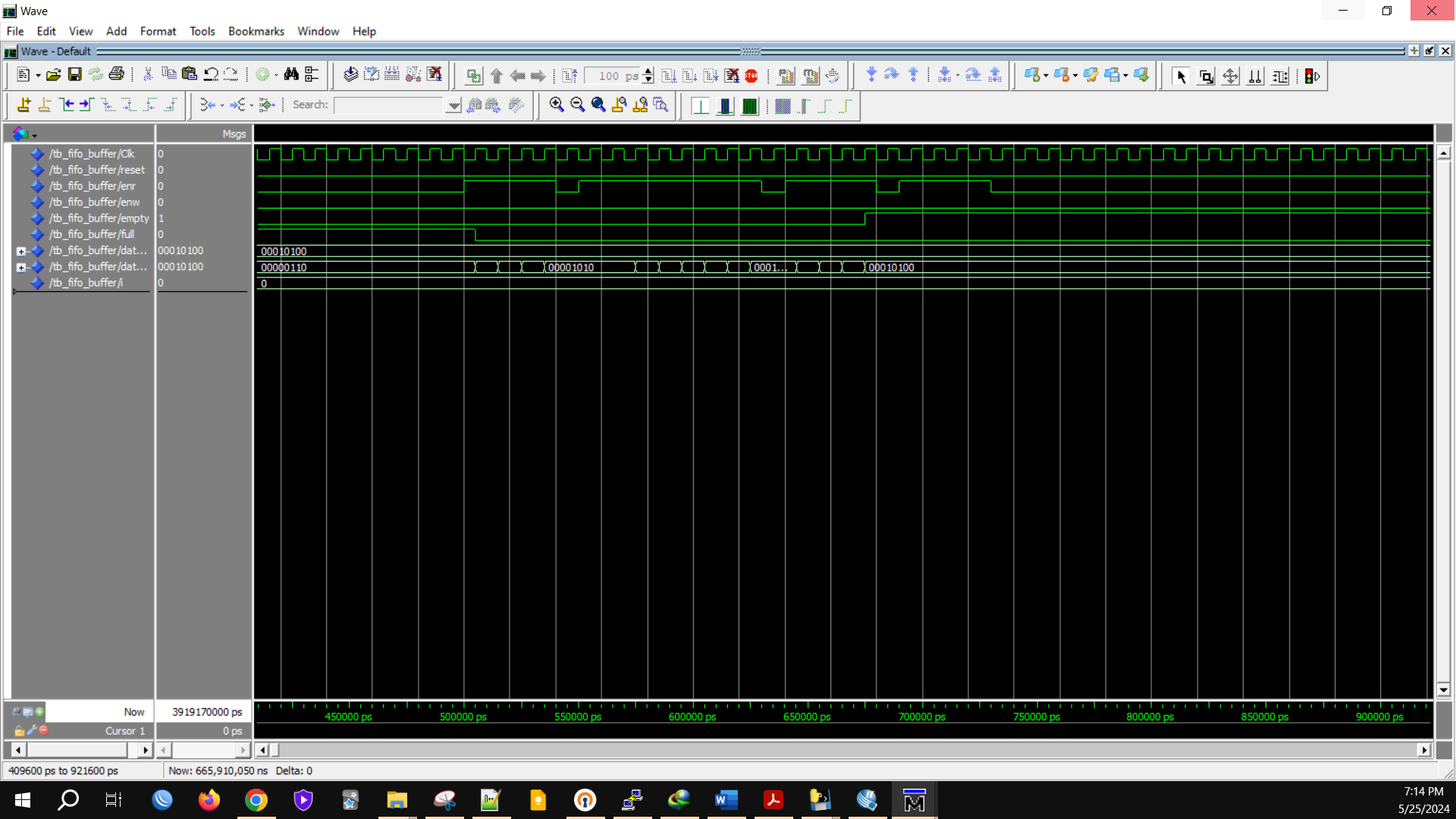


* **Wave Form Code:**









* **Test Bench Code:**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

ENTITY tb\_fifo\_Buffer IS

END tb\_fifo\_Buffer;

ARCHITECTURE behavior OF tb\_fifo\_Buffer IS

--Inputs and outputs

signal Clk,reset,enr,enw,empty,full : std\_logic := '0';

signal data\_in,data\_out : std\_logic\_vector(7 downto 0) := (others => '0');

--temporary signals

signal i : integer := 0;

-- Clock period definitions

constant Clk\_period : time := 10 ns;

constant DEPTH : integer := 16; --specify depth of fifo here.

BEGIN

-- Instantiate the Unit Under Test (UUT)

uut: entity work.fifo\_Buffer

generic map(DEPTH => DEPTH)

PORT MAP (clk => clk,

reset => reset,

enr => enr,

enw => enw,

data\_in => data\_in,

data\_out => data\_out,

fifo\_empty => empty,

fifo\_full => full);

-- Clock process definitions

Clk\_process :process

begin

Clk <= '0';

wait for Clk\_period/2;

Clk <= '1';

wait for Clk\_period/2;

end process;

-- Stimulus process

stim\_proc: process

begin

reset <= '1'; --apply reset for one clock cycle.

wait for clk\_period;

reset <= '0';

wait for clk\_period\*3; --wait for 3 clock periods(simply)

enw <= '1'; enr <= '0'; --write 10 values to fifo.

for i in 1 to 10 loop

data\_in <= conv\_std\_logic\_vector(i,8);

wait for clk\_period;

end loop;

enw <= '0'; enr <= '1'; --read 4 values from fifo.

wait for clk\_period\*4;

enw <= '1'; enr <= '1'; --read and write at the same time for 2 clock cycles

wait for clk\_period\*2;

enw <= '0'; enr <= '0'; --neither read nor write

wait for clk\_period\*10; --wait for some clock cycles.

enw <= '1'; enr <= '0'; --write 10 values to fifo.

for i in 11 to 20 loop

data\_in <= conv\_std\_logic\_vector(i,8);

wait for clk\_period;

end loop;

enw <= '0'; enr <= '0'; --neither read nor write

wait for clk\_period\*10; --wait for some clock cycles.

enw <= '0'; enr <= '1'; --read 4 values from fifo.

wait for clk\_period\*4;

enw <= '0'; enr <= '0'; --neither read nor write

wait for clk\_period;

enw <= '0'; enr <= '1'; --read 4 values from fifo.

wait for clk\_period\*8;

enw <= '0'; enr <= '0'; --neither read nor write

wait for clk\_period;

enw <= '0'; enr <= '1'; --read 8 values from fifo.

wait for clk\_period\*4;

enw <= '0'; enr <= '0'; --neither read nor write

wait for clk\_period;

enw <= '0'; enr <= '1'; --read 4 values from fifo.

wait for clk\_period\*4;

enw <= '0'; enr <= '0'; --neither read nor write

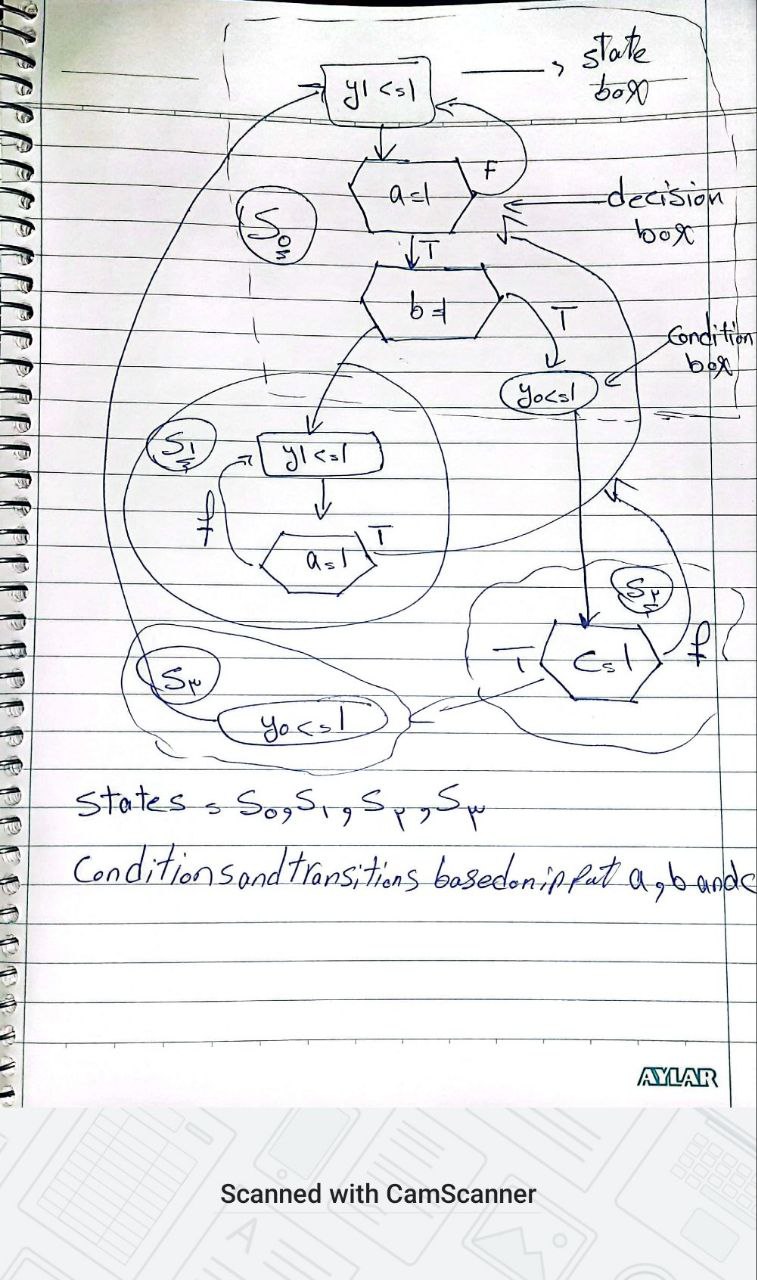
wait;

end process;

END;

**Q3.** To Implement the FSM and verify its correctness we first need FSM module so we declare the input and outputs then describe its behavior. After that we need Look-ahead output buffer to make sure that glitch- free moore outputs, implement a buffer that update outputs on the clock edge.

* **ASM chart:**



* State s0:
* Output: y0 = 1, y1 = 0, y2 = 0
* Decision: If a = 1 -> go to s1
* Decision: If a = 0 -> stay in s0
* State s1:
* Output: y0 = 0, y1 = 1, y2 = 0
* Decision: If a = 1 and b = 0 -> go to s2
* Decision: If a = 0 -> stay in s1
* State s2:
* Output: y0 = 0, y1 = 0, y2 = 1
* Decision: If a = 0 and b = 1 -> go to s0
* Decision: If c = 1 -> go to s3
* State s3:
* Output: y0 = 0, y1 = 0, y2 = 1
* Decision: If c = 0 -> go to s0
* Decision: If c = 1 -> stay in s3
* **Code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity FSM is

Port ( clk : in STD\_LOGIC;

reset : in STD\_LOGIC;

a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

y0, y1, y2 : out STD\_LOGIC);

end FSM;

architecture Behavioral of FSM is

type state\_type is (s0, s1, s2, s3);

signal state, next\_state : state\_type;

signal y0\_reg, y1\_reg, y2\_reg : STD\_LOGIC := '0';

begin

process(clk, reset)

begin

if reset = '1' then

state <= s0;

elsif rising\_edge(clk) then

state <= next\_state;

end if;

end process;

process(state, a, b, c)

begin

case state is

when s0 =>

if a = '1' then

next\_state <= s1;

else

next\_state <= s0;

end if;

y0\_reg <= '1';

y1\_reg <= '0';

y2\_reg <= '0';

when s1 =>

if a = '1' and b = '0' then

next\_state <= s2;

elsif a = '0' then

next\_state <= s1;

else

next\_state <= s0;

end if;

y0\_reg <= '0';

y1\_reg <= '1';

y2\_reg <= '0';

when s2 =>

if a = '0' and b = '1' then

next\_state <= s0;

elsif c = '1' then

next\_state <= s3;

else

next\_state <= s2;

end if;

y0\_reg <= '0';

y1\_reg <= '0';

y2\_reg <= '1';

when s3 =>

if c = '0' then

next\_state <= s0;

else

next\_state <= s3;

end if;

y0\_reg <= '0';

y1\_reg <= '0';

y2\_reg <= '1';

when others =>

next\_state <= s0;

end case;

end process;

-- Look-ahead output buffer to ensure glitch-free Moore output

process(clk)

begin

if rising\_edge(clk) then

y0 <= y0\_reg;

y1 <= y1\_reg;

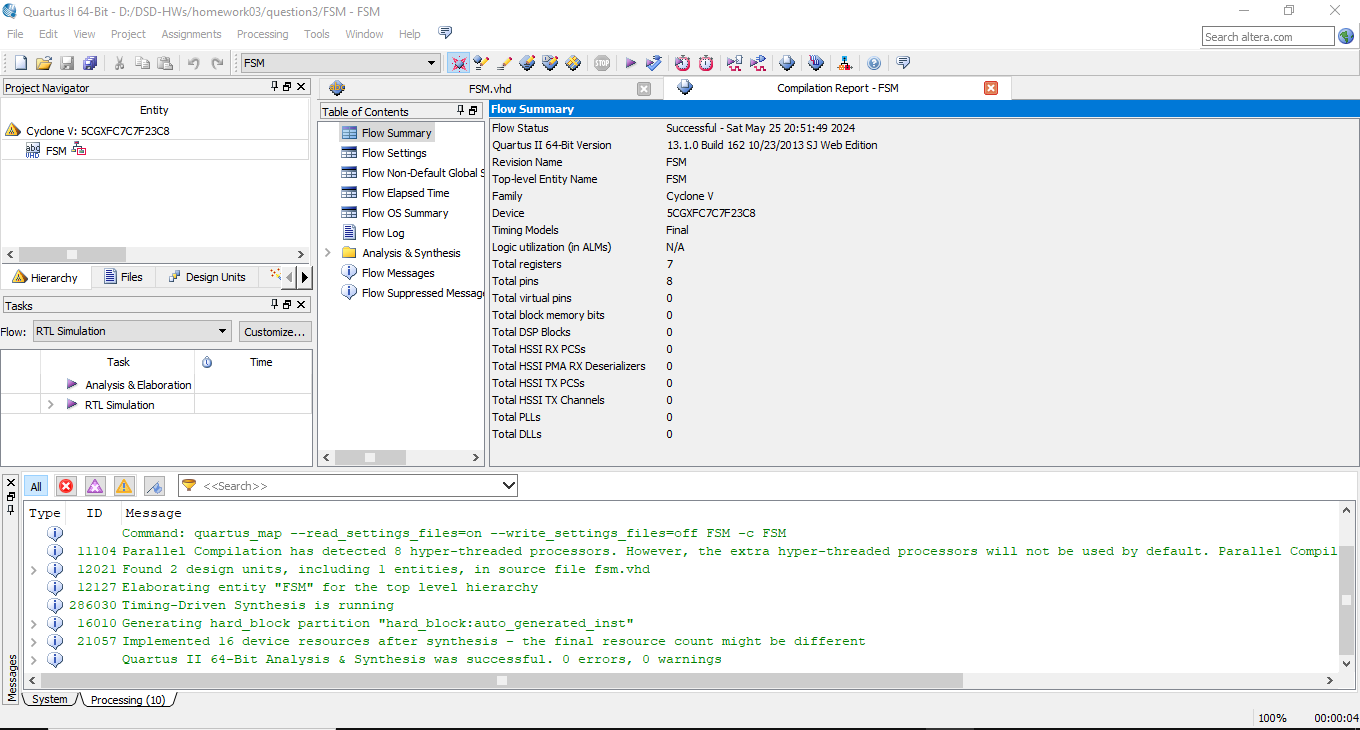
y2 <= y2\_reg;

end if;

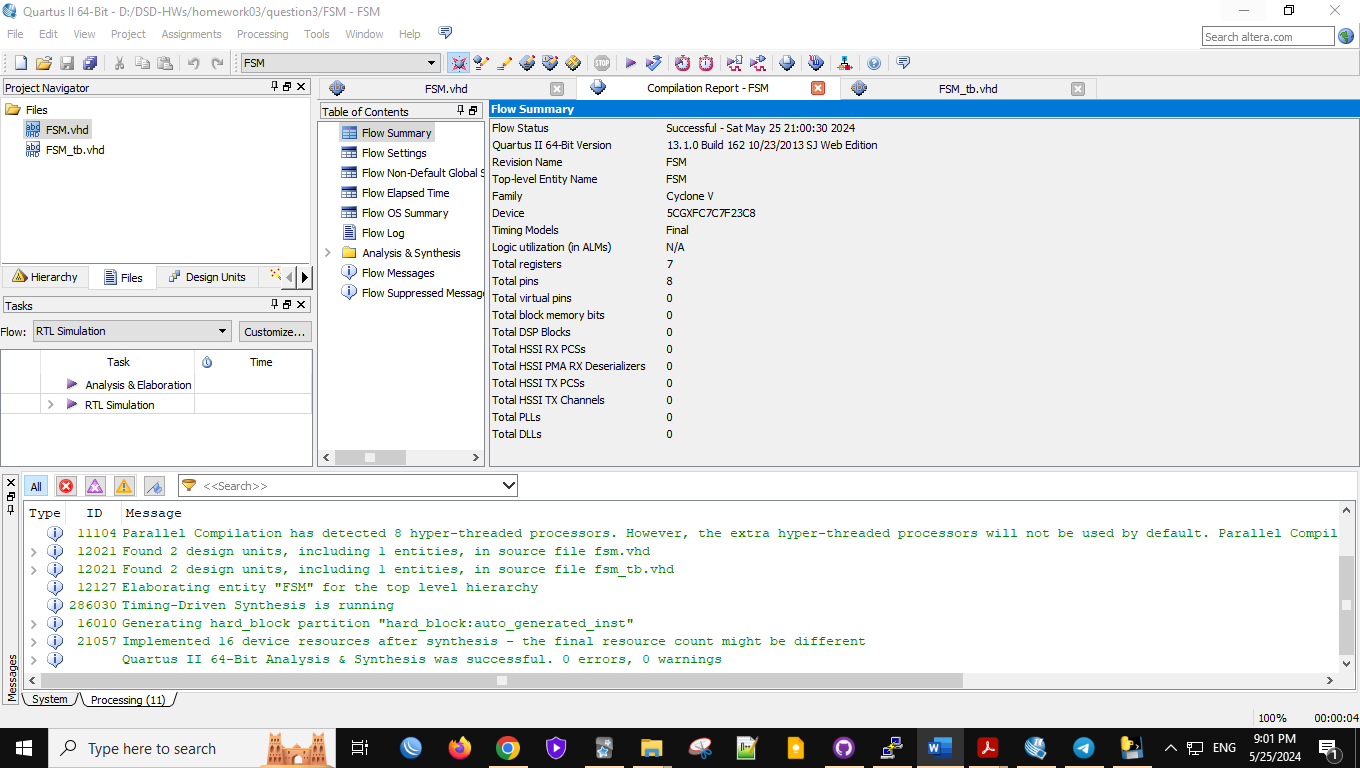
end process;

end Behavioral;

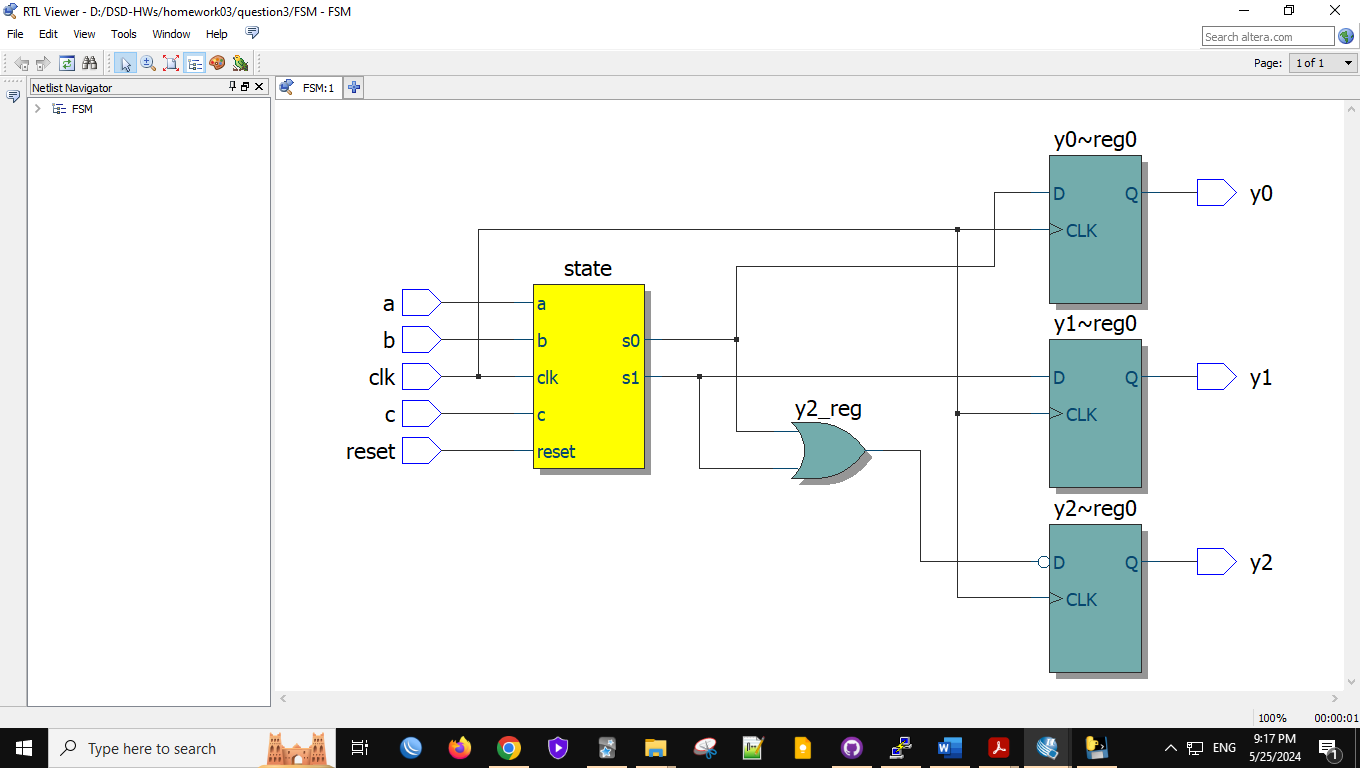
* **Compilation Report:**



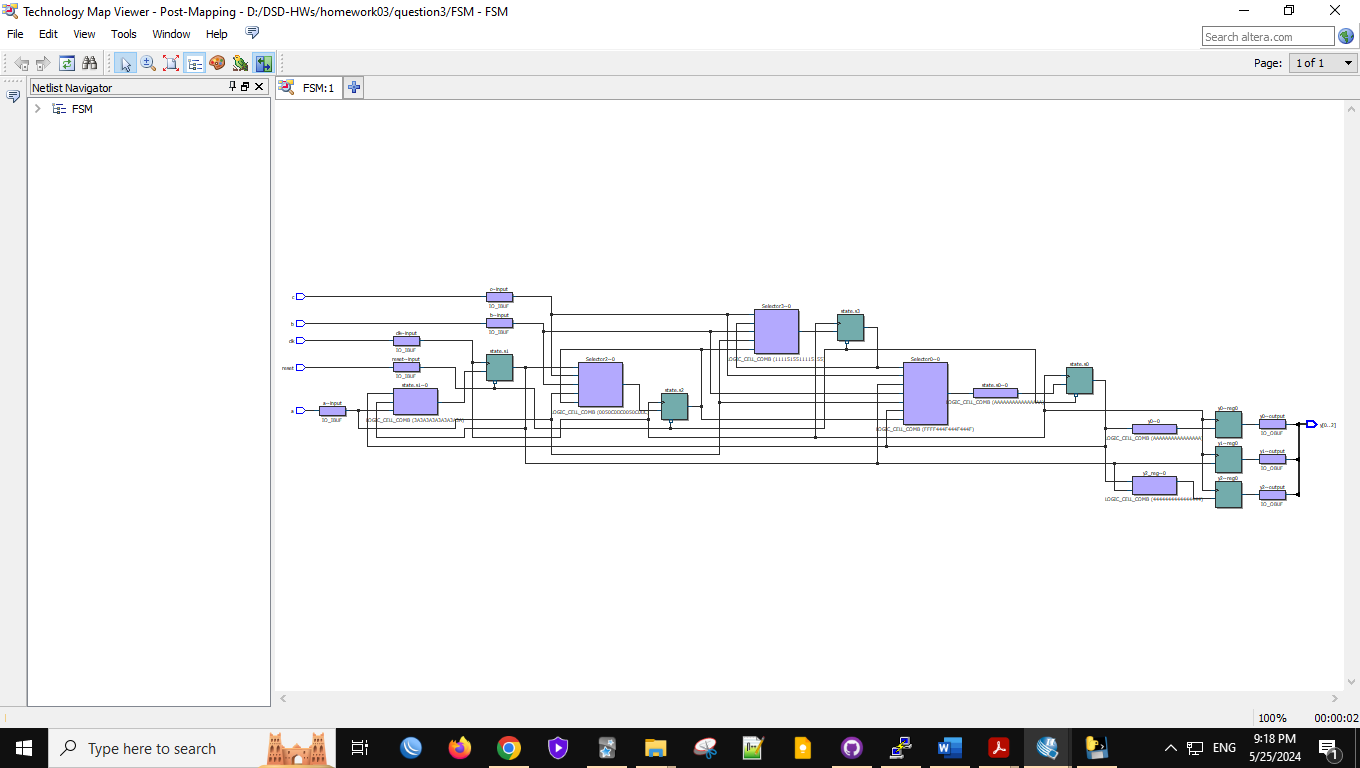
* **Compilation Report for Testbench Code:**



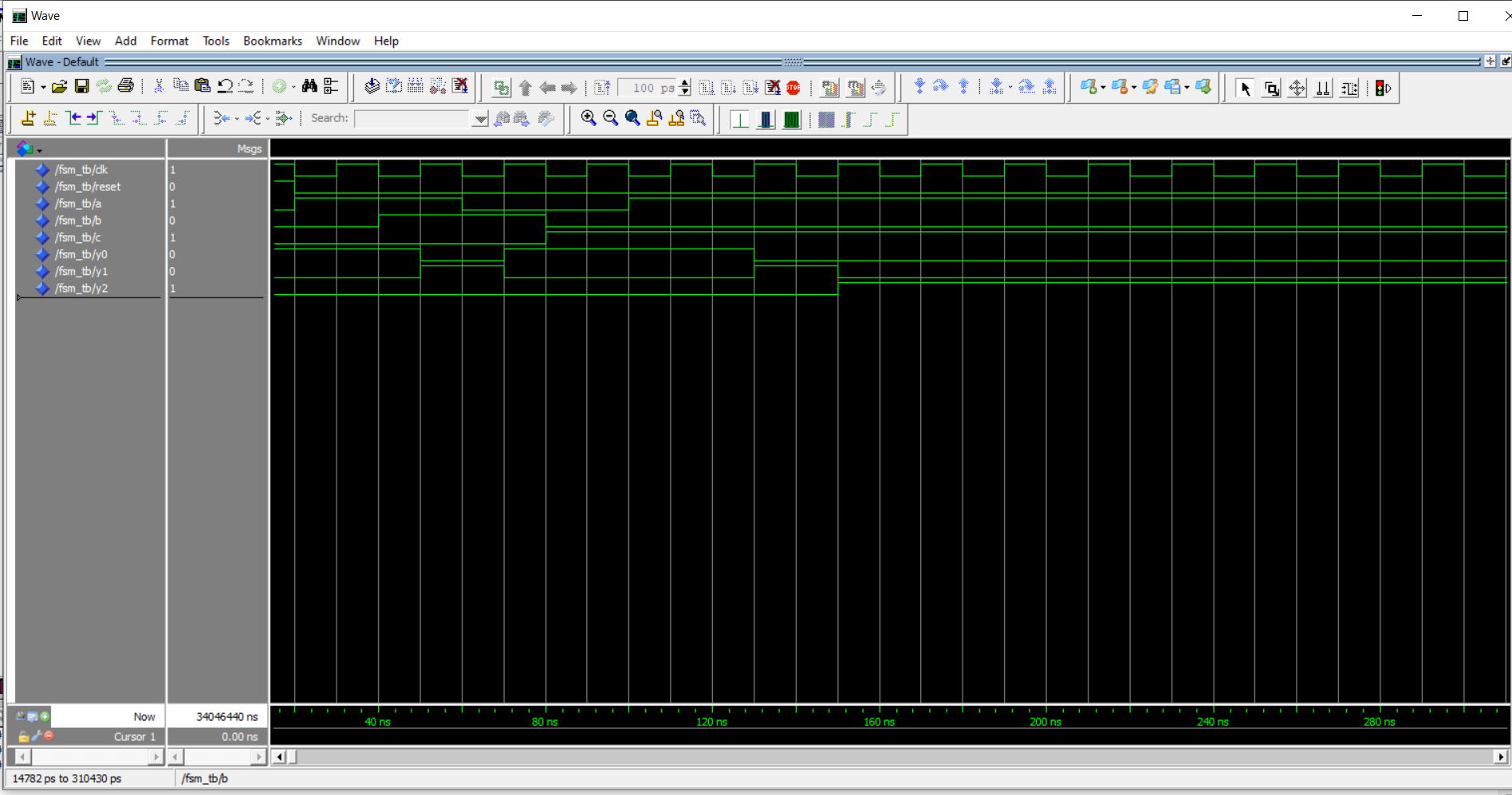
* **RTL View:**

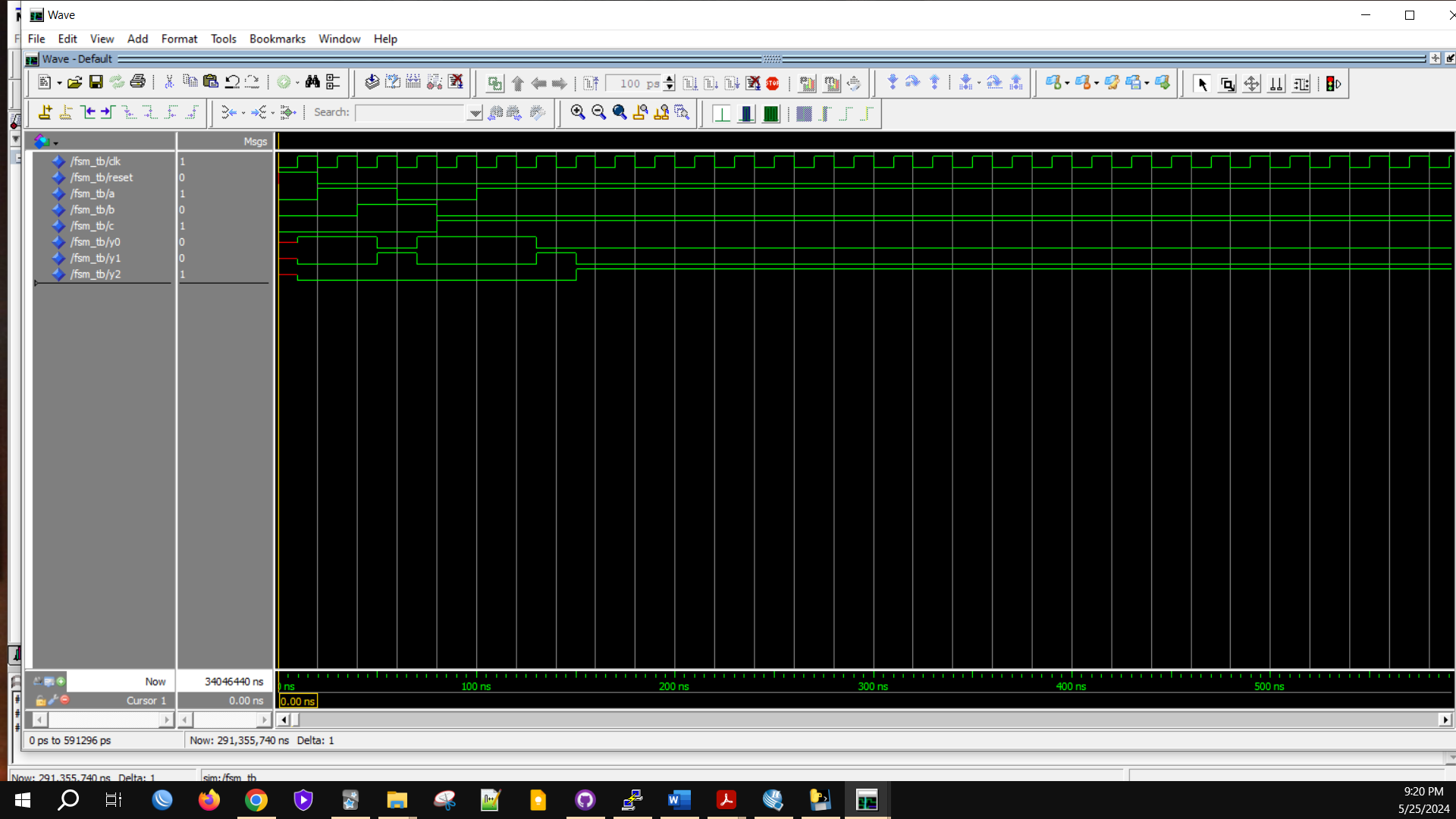


* **Post- Mapping View:**



* **Wave View:**





* **Testbench code:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity FSM\_tb is

end FSM\_tb;

architecture Behavioral of FSM\_tb is

signal clk : STD\_LOGIC := '0';

signal reset : STD\_LOGIC := '0';

signal a : STD\_LOGIC := '0';

signal b : STD\_LOGIC := '0';

signal c : STD\_LOGIC := '0';

signal y0, y1, y2 : STD\_LOGIC;

component FSM

Port ( clk : in STD\_LOGIC;

reset : in STD\_LOGIC;

a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

y0, y1, y2 : out STD\_LOGIC);

end component;

begin

uut: FSM Port map (

clk => clk,

reset => reset,

a => a,

b => b,

c => c,

y0 => y0,

y1 => y1,

y2 => y2

);

clk\_process: process

begin

clk <= '0';

wait for 10 ns;

clk <= '1';

wait for 10 ns;

end process;

stim\_proc: process

begin

-- Initialize Inputs

reset <= '1';

wait for 20 ns;

reset <= '0';

-- Test sequence

a <= '1'; b <= '0'; c <= '0';

wait for 20 ns;

a <= '1'; b <= '1'; c <= '0';

wait for 20 ns;

a <= '0'; b <= '1'; c <= '0';

wait for 20 ns;

a <= '0'; b <= '0'; c <= '1';

wait for 20 ns;

a <= '1'; b <= '0'; c <= '1';

wait for 20 ns;

-- Stop the simulation

wait;

end process;

end Behavioral;