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

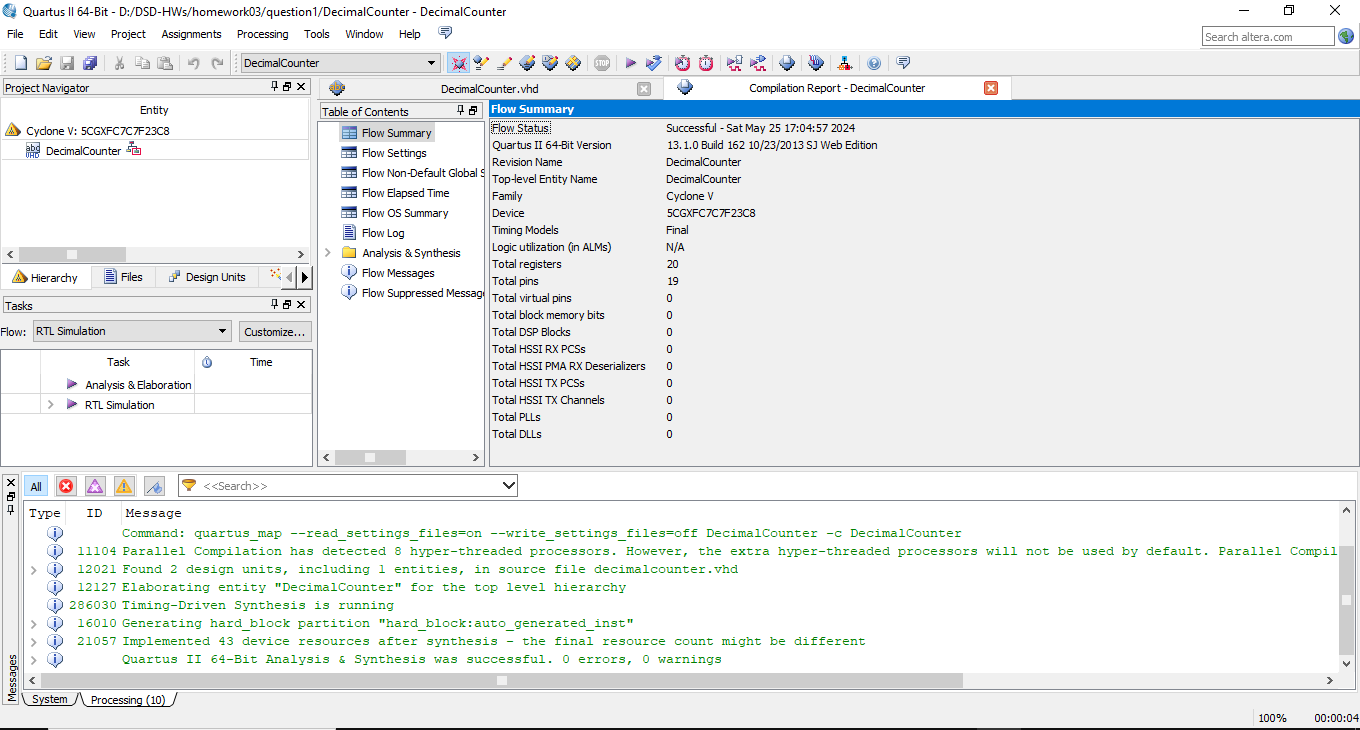
count((i\*4)+3 downto i\*4) <= digit\_values(i);

end loop;

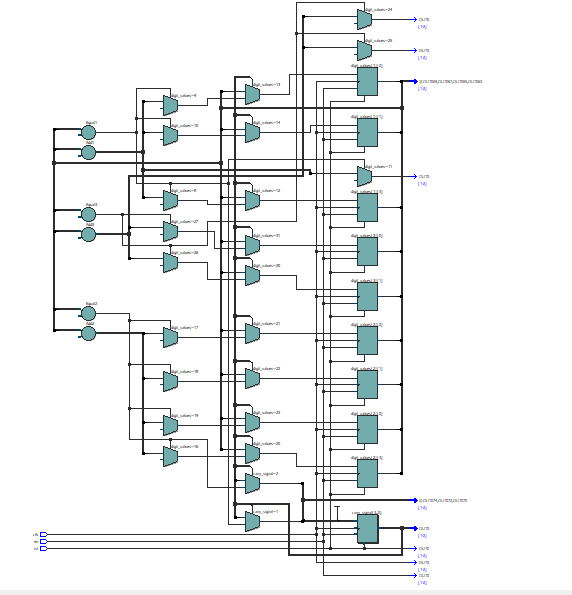
end process;

end Behavioral;

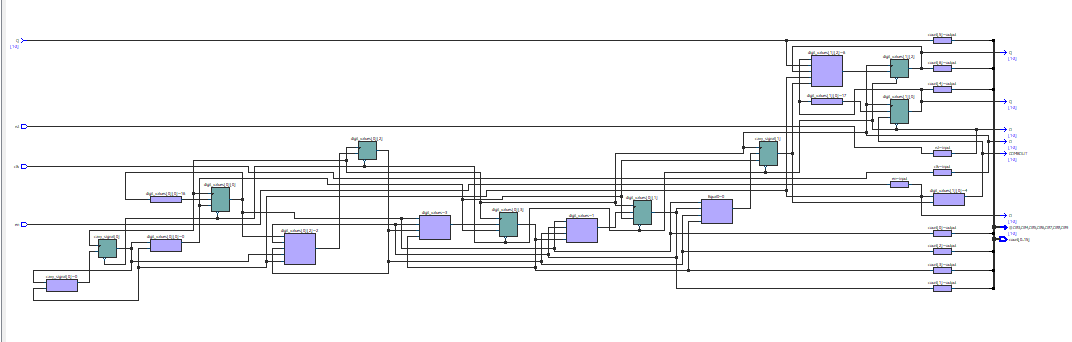
Compilation Report:



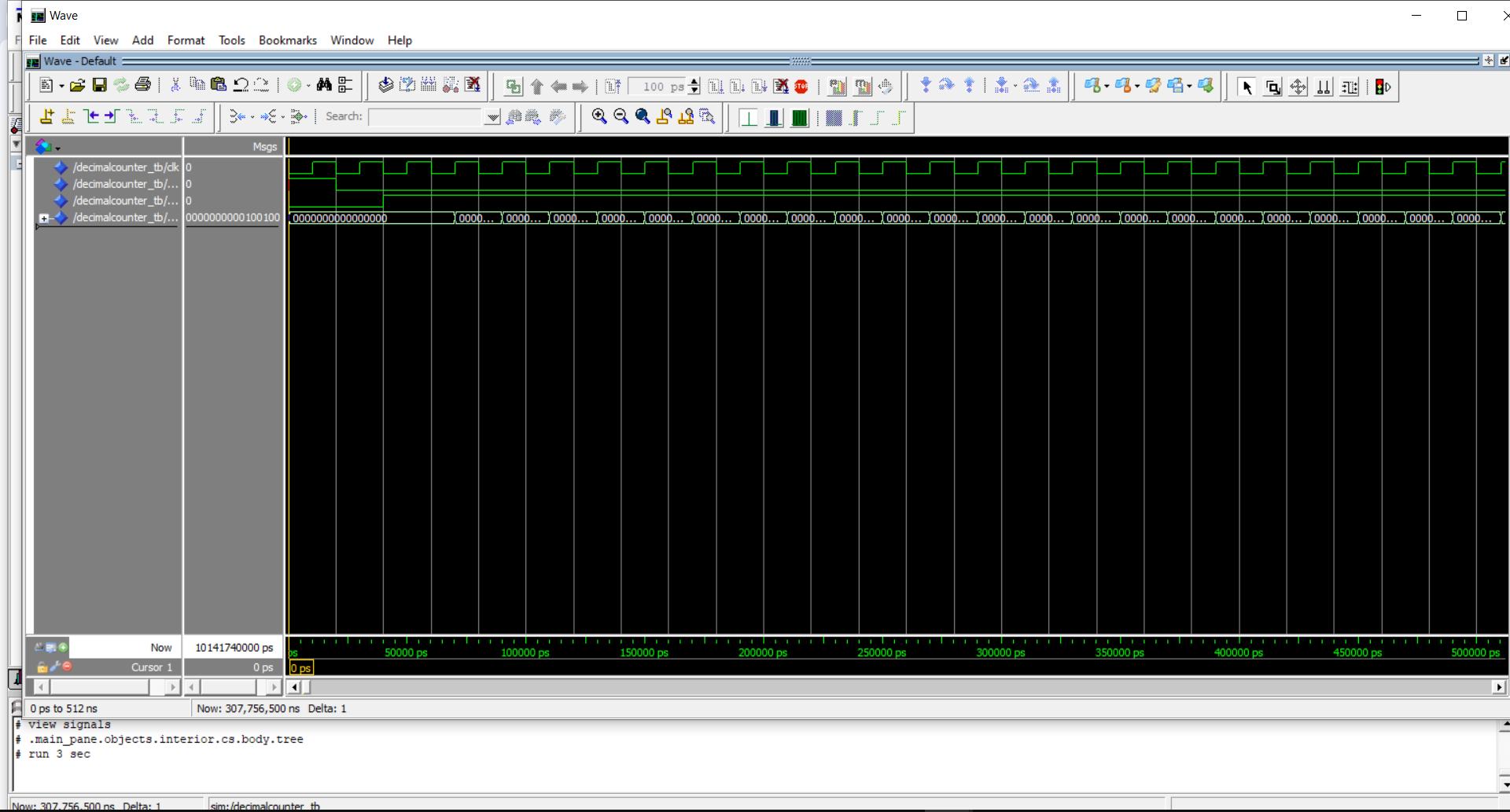
RTL View:



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.