Assignment 2
Generic Divider

By
Vraj

A generic Divider design that can be reused for n-bit operand division operation in serial form is made in this assignment. The ASM chart implemented in this report is taken for the Assigment-2 divider diagram.

Input/Output of the Divider

For this example A = 255, B = 14, Therefor O_Q = 18 (quotient) and O_R = 3 (remainder). In Figure-1

A, B => Inputs
O_R, O_Q => Output
Start => Input (Start dividing)
Done => Output (Results available at the Output)

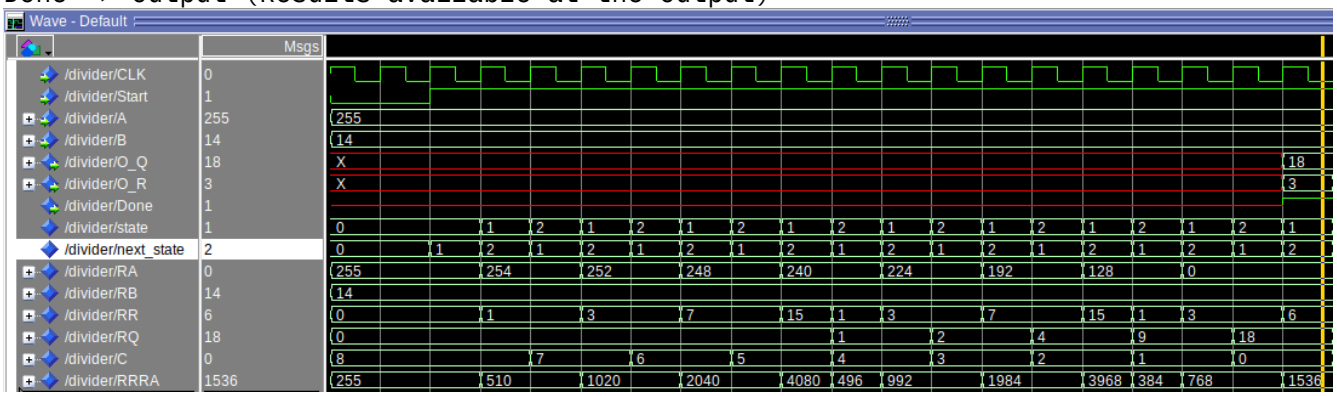


Fig-1:- 8-bit Divider for 255/14

Figure 1 shows all the signals of the generic divider, but the top-level entity will only have CLK, Start, Done, A, B, O_R, and O_Q, as seen from the image after the start bit is '1'. The divider switches between states 1 and 2 eight times for 8-bit operands. When the done bit is high, we obtain the result at O_R and O_Q—the generic divider is placed in a different file, as seen in the ASS2-code-sim file.

Area vs Operand

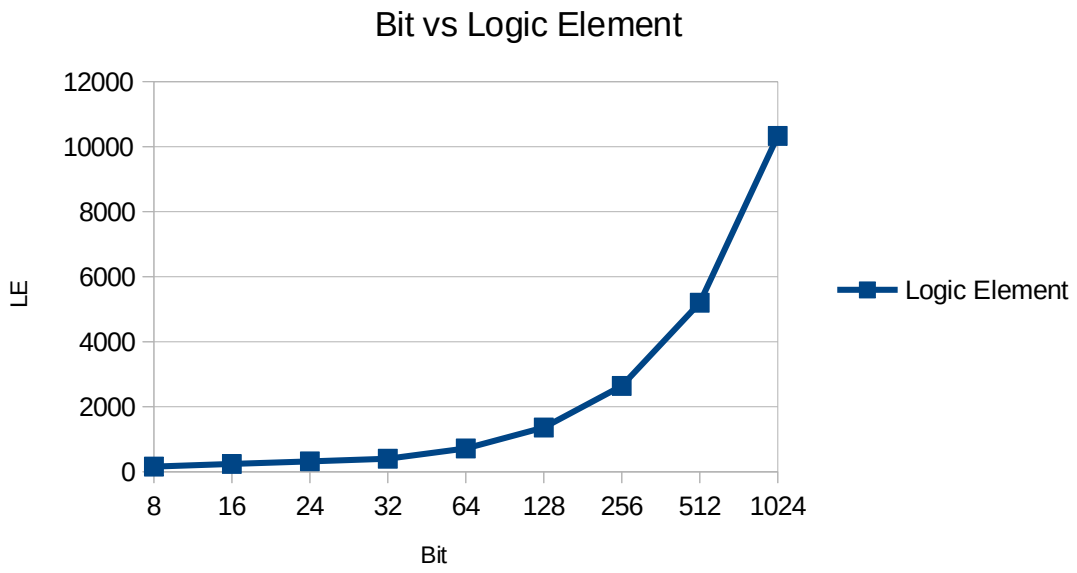


Table-1 :- Bits vs Logic Element

BIT Size	Logic Element	Frequency (MHz)
8	161	190.84
16	241	204.67
24	321	179.95
32	400	189.19
64	720	
128	1361	
256	2641	
512	5201	
1024	10331	

As seen in Table-1 and Chat above, we can see the number of the logic elements increases as the operand size increases; it is interesting to note that the FPGA itself can implement 128, 256, 512, and 1024-bit operand division even more, but is limited by the number of I/O pins. We can design an interface for serial I/O or multiplexed I/O for this divider, as the number of pines is limited to 310. This shows the silicon internally can implement various complex logic, such as 512 carry look-ahead adders in a matrix form for data crunching. Still, the main limiting factor of data processing will be the input-output data movement, even internal data movement.

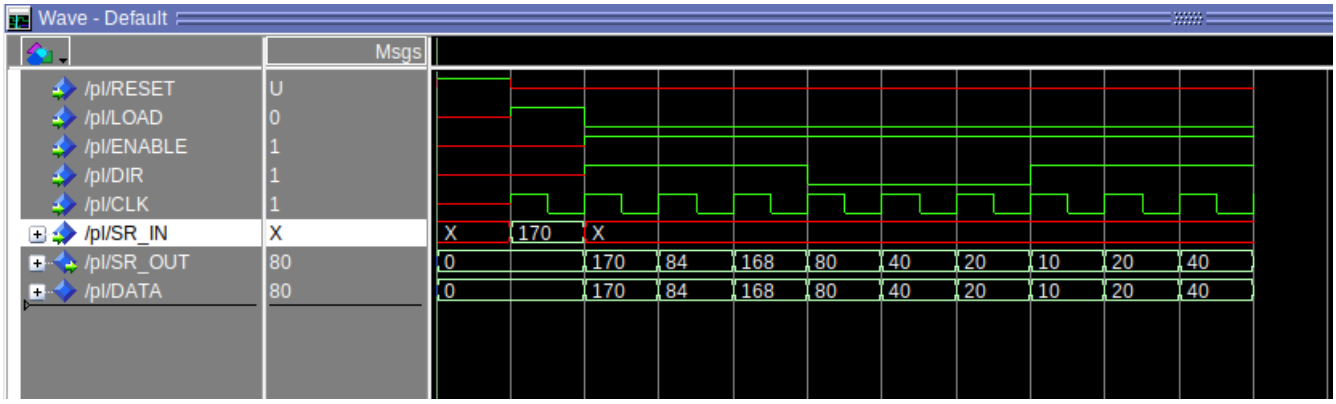
Code:
Q1: 8 bit divider
File-1:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity pl is
port(
--    RESET: in std_logic_vector(9 downto 0);
      RESET,LOAD, ENABLE, DIR, CLK: in std_logic;
      SR_IN: in std_logic_vector(7 downto 0);
      SR_OUT: out std_logic_vector(7 downto 0)
);
end entity;

architecture arch of pl is
      signal DATA: std_logic_vector(7 downto 0);
      begin
      process(CLK, LOAD, ENABLE, RESET,DIR)
            begin
                  if(RESET = '1') then
                        DATA <= (others => '0');
                  else
                  if    (LOAD = '1' and rising_edge(CLK)) then
                        DATA <= SR_IN;
                  elsif(ENABLE = '1' and rising_edge(CLK) and DIR = '0') then
                        DATA <= '0' & DATA(7 downto 1);
                  elsif(ENABLE = '1' and rising_edge(CLK) and DIR = '1') then
                        DATA <= DATA(6 downto 0) & '0';
                  end if;
                  end if;
      end process;
      SR_OUT <= DATA;
end architecture;
```

Output:



Q2:

File-1: N-Bit divider

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity gendiv is
port(
--    RESET: in std_logic_vector(9 downto 0);
      CLK, Start: in std_logic;
      A, B: in std_logic_vector(7 downto 0);
      O_Q, O_R: out std_logic_vector(7 downto 0);
      Done: out std_logic
);
end entity;

architecture arch of gendiv is
      component divider
            generic(
                  n : integer := 8
            );
            port(
                  CLK, Start: in std_logic;
                  A, B: in std_logic_vector(n-1 downto 0);
                  O_Q, O_R: out std_logic_vector(n-1 downto 0);
                  Done: out std_logic
            );
      end component;
begin
divider1 : divider generic map (n => 8) port map(CLK => CLK, Start => Start, A =>
A, B => B, O_Q => O_Q, O_R => O_R, Done => Done);

end architecture;
```

File-2: Generic Divider

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity divider is
generic(
 n : integer := 8
 );
port(
--    RESET: in std_logic_vector(9 downto 0);
      CLK, Start: in std_logic;
      A, B: in std_logic_vector(n-1 downto 0);
      O_Q, O_R: out std_logic_vector(n-1 downto 0);
```

```
      Done: out std_logic
);
end divider;

architecture arch of divider is
      signal state, next_state: integer range 0 to 2;
      signal RA, RB: std_logic_vector(n-1 downto 0);
      signal RR, RQ: std_logic_vector(n-1 downto 0);
      signal C: integer := n;

      begin
      process (state, Start)

      begin
                  case state is
                        when 0 =>
                              if Start = '1' then
                                    next_state <= 1;
                              else
                                    next_state <= 0;
                              end if;
                        when 1 =>
                              next_state <= 2;
                        when 2 =>
                              if (c=0) then
                                    next_state <= 0;
                              else
                                    next_state <= 1;
                              end if;
                  end case;
      end process;

      process (CLK)
      begin
      if (rising_edge(CLK)) then
            state <= next_state;
      end if;
      end process;
      process (state)
      begin
            if (state = 0) then
                        RR <= (others => '0');
                        RQ  <= (others => '0');
                        RA <= A; RB <= B;
                        c <= n;
            elsif (state = 1) then
                  RR <= RR(n-2 downto 0)&RA(n-1);
                  RA <= RA(n-2 downto 0)&'0';
            elsif (state = 2) then
                  c <= c-1;
                  if (RR >= RB) then
```

```
                        RR <= RR - RB;
                        RQ <= RQ(n-2 downto 0)&'1';
                  else
                        RQ <= RQ(n-2 downto 0)&'0';
                  end if;
            end if;
            if (c = 0) then
                  Done <= '1';
                  O_R <= RR;
                  O_Q <= RQ;
            end if;
      end process;
end architecture;
```

Output: