Three-function barrel shifter

Name	ID
Adham Ehab Erfan Mahmoud	**21010221**
Adham Abo Bakr Morsy Mohamed	**21010219**
Ashraf Qasim Abd El Hafeez	**21010276**
Ahmed Omar Magawry Ibrahim	**21010134**
Ahmed Yasser Ahmed El Arouisy	**21010210**

GROUP 9

Introduction

A barrel shifter is a circuit that can shift input data by any number of positions. Shifting operations can be done either the left or right direction and are divided into rotate, logic shift and arithmetic shift.

Specification of the system

The system has the following specifications:

1) 8-bit input to input the date you aim to process it.
2) 8-bit output to output the processed data.
3) 4-bit input to input the number of steps to execute the operation on the date in binary form.
4) 3-bit input to choose the operation and the direction of operation executing on the date as shown in following table.
5) 1-bit input to input the clock of the system.
6) 1-bit input to reset the system to process new data.

selectors			operation
C	B	A	
0	0	0	No Change
C = 0 for Right direction	0	1	Logical Shift Right
	1	0	Arithmetic Shift Right
	1	1	Rotate Right
1	0	0	Parallel load
C = 1 for Left direction	0	1	Logical Shift Left
	1	0	Arithmetic Shift Left
	1	1	Rotate Left

Note: A is the LSB.

• It is clear that the selector C to determine the direction of the operation left or right.

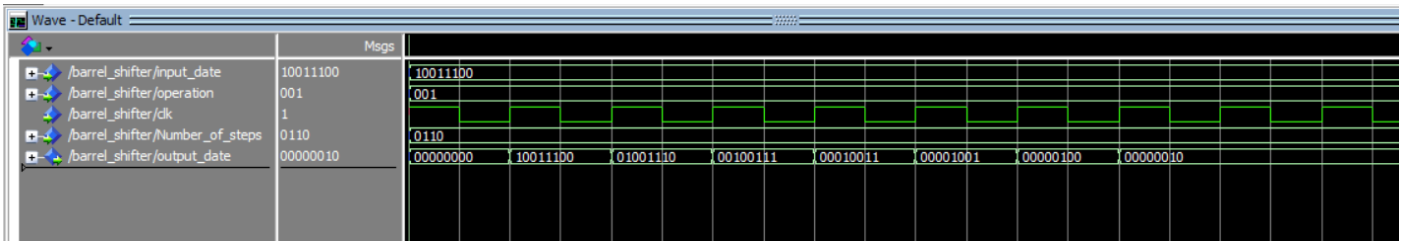
• And also, the type of the operation is determined by B & A selector so that

1. BA = 01 → Logical Shift
2. BA = 10 → Arithmetic Shift
3. BA = 11 → Rotation

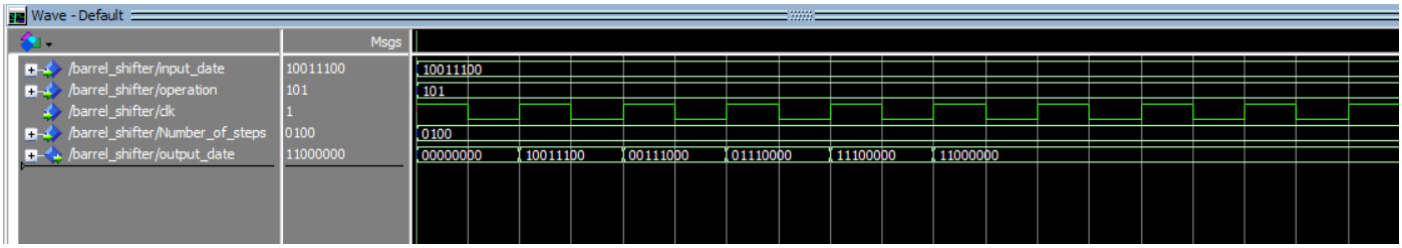
The simulation results

- The following is results of all operation on the date " 10011100 "

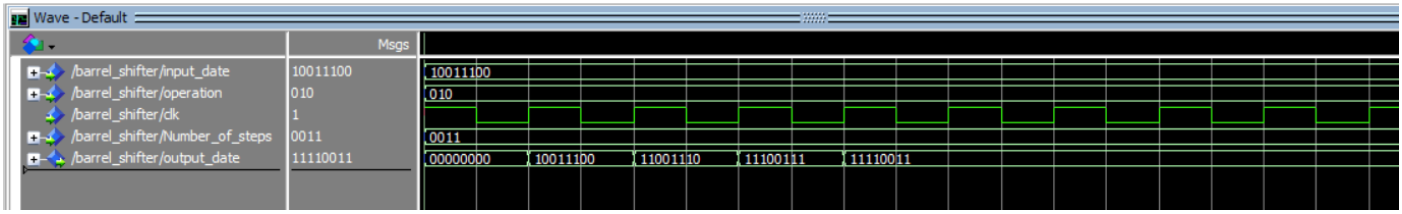
1) The logical shift right with number of steps equal to 6.



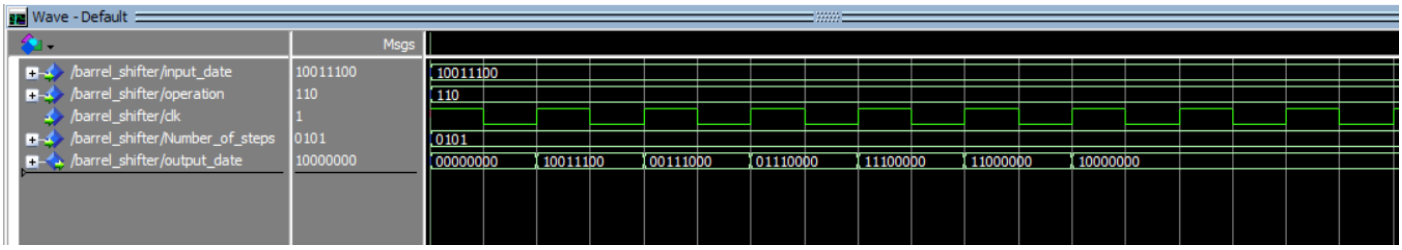
2) The logical shift left with number of steps equal to 4.



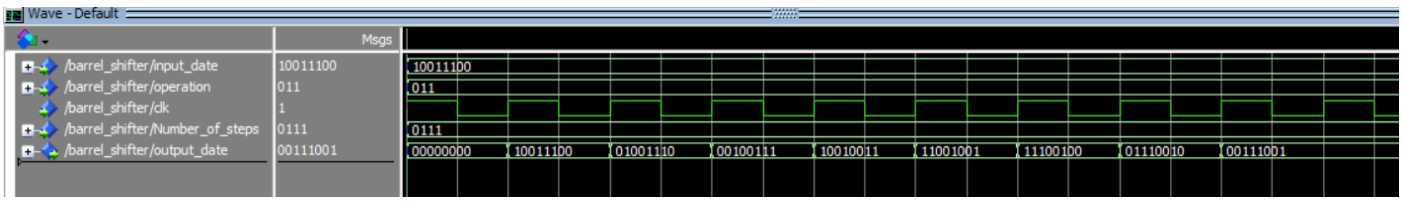
3) The Arithmetic Shift Right with number of steps equal to 3.



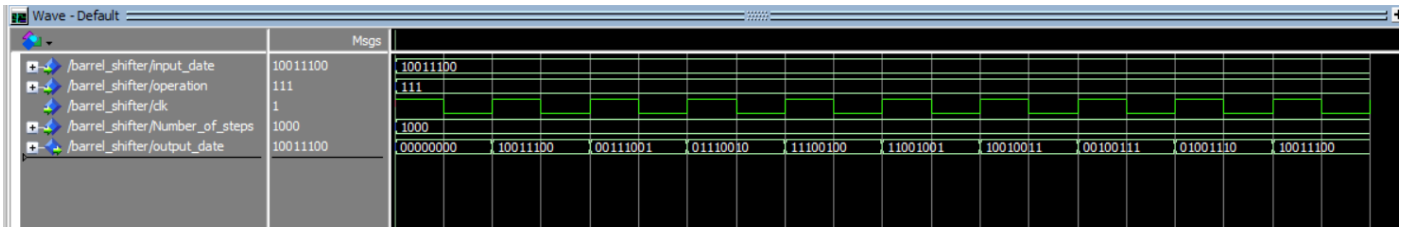
4) The Arithmetic Shift Left with number of steps equal to 5.



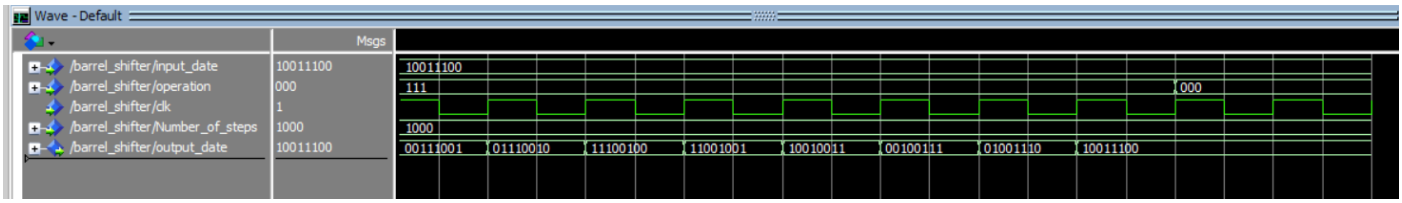
5) The Rotate Right with number of steps equal to 7.



6) The Rotate Left with number of steps equal to 8 → the same data is result.



7) The no change case.

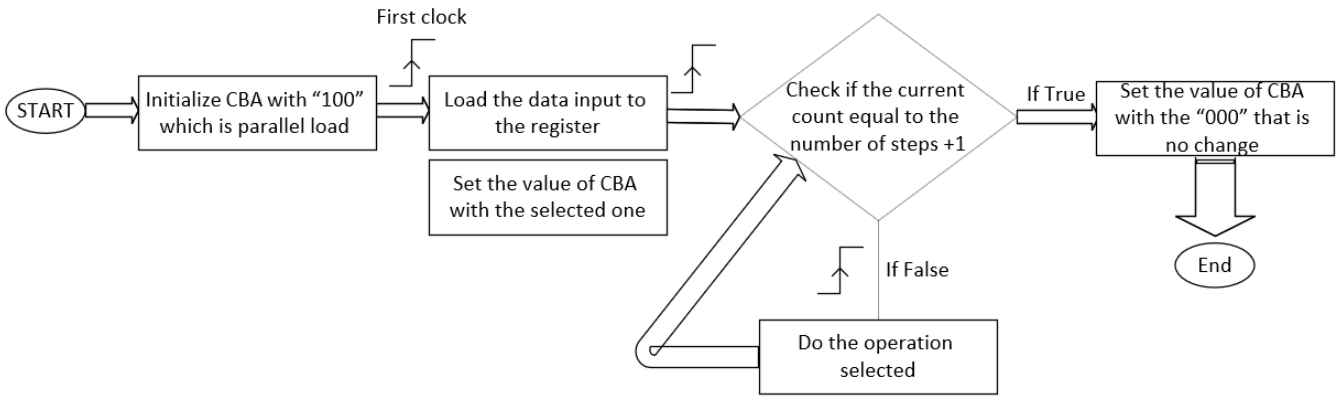


→ The parallel case is done internally at the 1st clock as illustrated in the previous figures. 😊

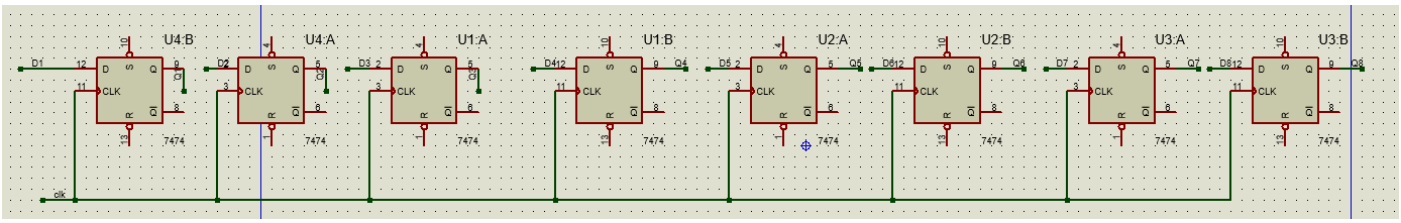
The schematic and steps of design

We build the system at first in proteus simulation to ease the process of design.

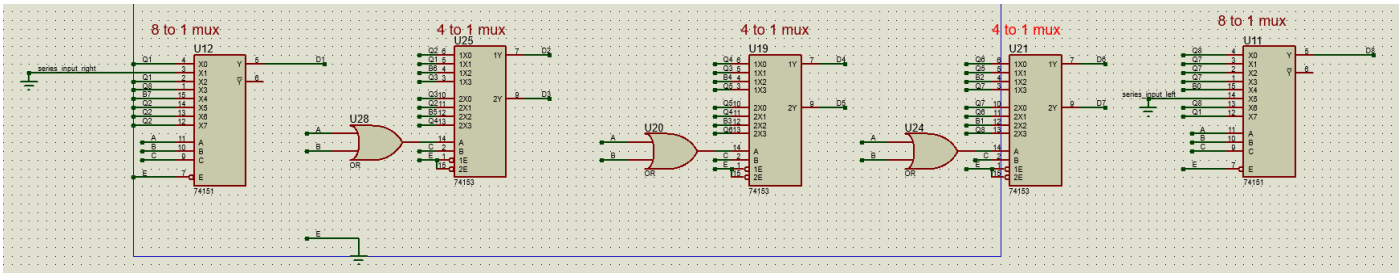
The simple algorithm we thought about to design the system.



First, we design the register which is 8 D-type flip flops which have common clk.



And each D input is connected with the output of the MUXs and the input of the MUXs is the output Q of each flip flop.

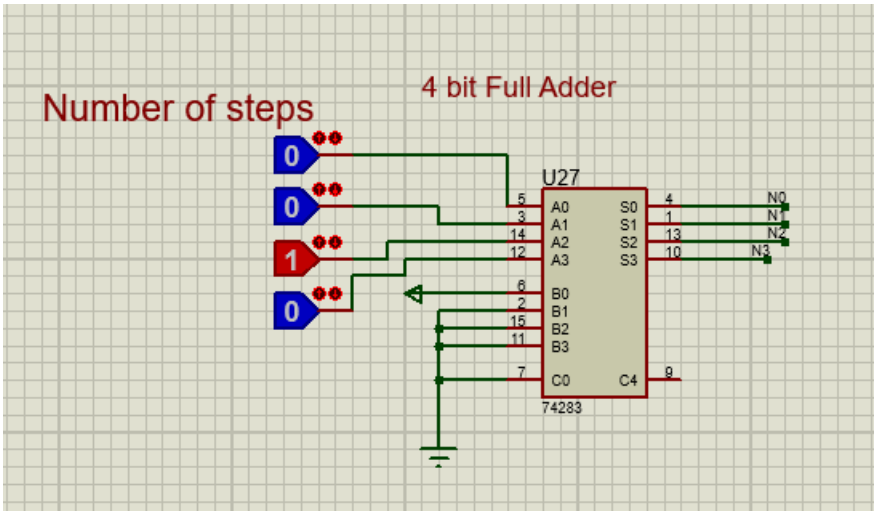


The boundary MUXs are 8 to 1 type while the other ones are 4 to 1 type and the reason for that is simply the all operation is just shifting either right or left or no change or load the date input and that is the four cases on the other hand the boundary MUXs have the whole eight cases that we illustrated before.

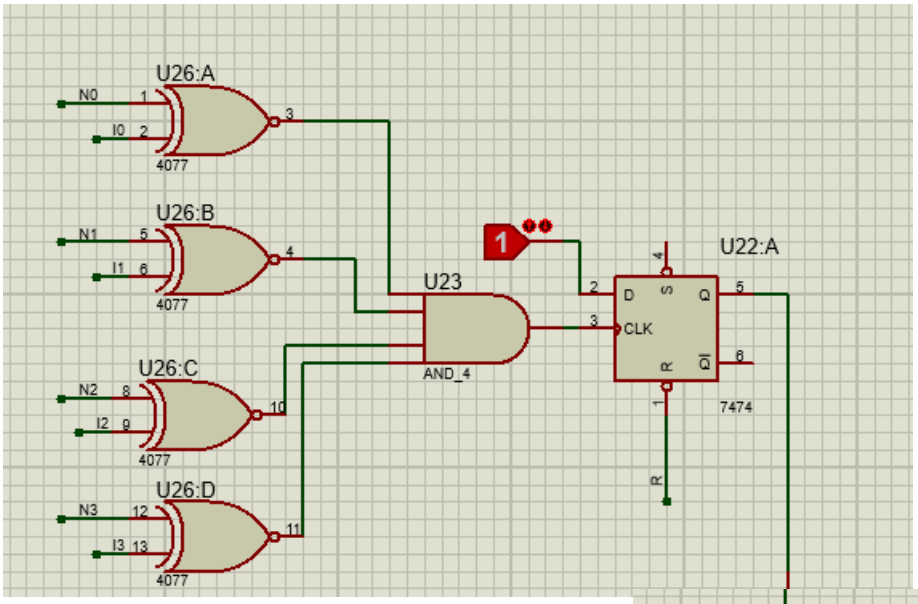
So up till now we build the operation functions and the next step is to make it repeat itself specific number of steps each rising edge clock.

So we have specific number of steps, For this reason, we build a 4-bit counter that is count each clock from "0000" → "1111", And the reason for 4-bit especially is simply when we repeat the operation 8 times in case of rotation the data will be the same and in case of logical shifting the data will be zeros and finally in case of arithmetic shifting the date will be either zeros or ones based on the date itself so it is meaningless to need counter more than that.

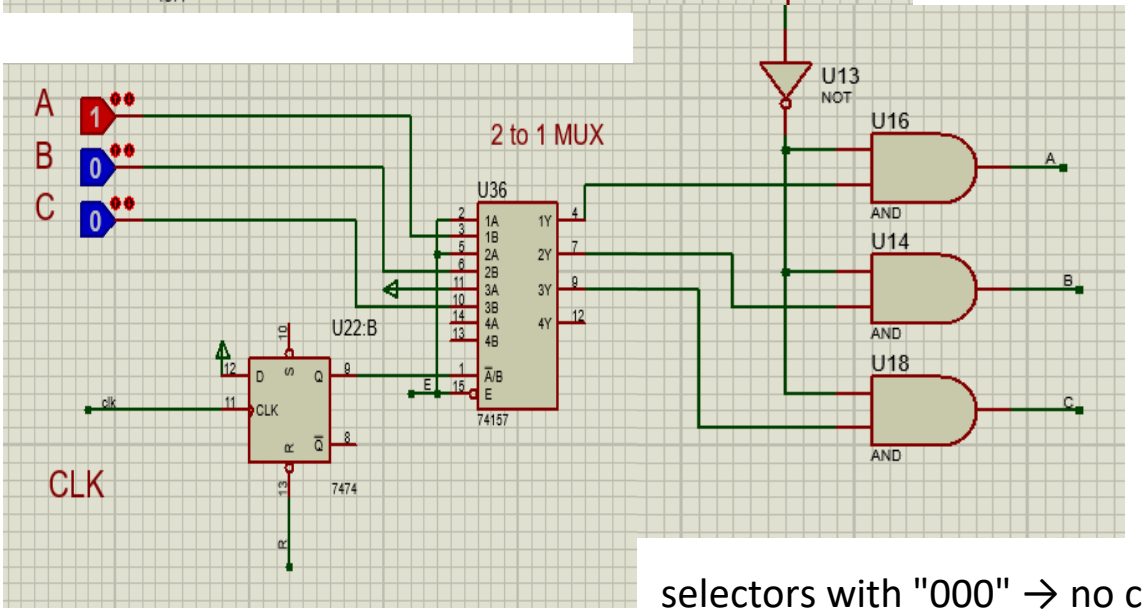
The next step is building sub-circuit that checks if the input number of steps +1 is equal to the current count or not -the reason for '+1' is the first clock period reserved for loading the date and the rest is the actual number of steps that the user wanted.



This is 4-bit full adder to increment the number of steps with 1.



This is sub-circuit to check if the incremented number of steps is equal to the current count or not, and if this condition is true then latch the output of the flip flop with high, and we will use this high signal to force the selector to be "000".



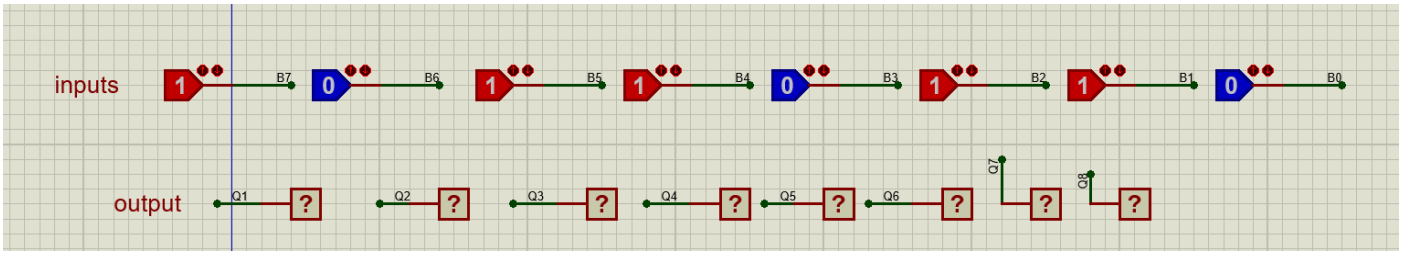
The high signal that generated before and indicates that we reached the count we wants will be inverted and make the selectors with "000" → no change case.

So now we can control the steps to process the date, now we want to make the first clock to load the date and that mean to make the selectors with "100",

And that can be done by the above snapped picture "2 to 1 MUXs" is always initialize the selectors with value "100" and just the first rising clock edge the flip flop latches its output value from 0 to 1 and this make the MUXs to select the input value from the user.

And finally, the output date we can take it from the output of each flip flop

And the input one is put to the certain input MUX that is load one "100".



The description of the sysyem using VHDL code

Before introducing the VHDL code there are some issues we will discuss

1) We did not build the counter using flip flops like in proteus but we build it like we take at the lectures

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity counter is
6      port(
7          C, CLR : in std_logic;
8          Q : out std_logic_vector(3 downto 0)
9      );
10 end counter;
11
12 architecture counter_imp of counter is
13     signal tmp: unsigned(3 downto 0) := (others => '0');
14     begin
15     process (C, CLR)
16         begin
17             if (CLR = '1') then
18                 tmp <= (others => '0');
19             elsif rising_edge(C) then
20                 tmp <= tmp + 1;
21             end if;
22     end process;
23
24         Q <= std_logic_vector(tmp);
25 end architecture;
26
```

2) We did not build a 4-bit full adder and replace it in the code with the following line of code.

```
125        int_N       <= std_logic_vector(unsigned(Number_of_steps) + 1);
126
```

3) The following is the description code for (2 to 1 MUX) & (8 to 1 MUX) respectively and we also use them as a module in the main code and we did not build 4 to 1 MUX but use the 8 to 1 instead because it will do the same thing and use 4 to 1 in proteus to show the optimum version for the circuit.

```vhdl
library ieee;
use ieee.std_logic_1164.all;


entity mux_2_to_1 is
    port(s0:in std_logic;
         x:in  std_logic_vector(1 downto 0);
         f:out std_logic);
end mux_2_to_1;

architecture mux_2_to_1_imp of mux_2_to_1 is

    begin
    with s0 select
       f <= x(0) when '0',
            x(1) when others;

end architecture;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mux_8_to_1 is
      port(S:in  std_logic_vector(2 downto 0);
           F:out std_logic;
           x:in  std_logic_vector(8 downto 1) );
end mux_8_to_1;

architecture mux_8_to_1_imp of mux_8_to_1 is

   begin
      with S select --CBA
      F <= x(1) when "000",
           x(2) when "001",
           x(3) when "010",
           x(4) when "011",
           x(5) when "100",
           x(6) when "101",
           x(7) when "110",
           x(8) when others;
end architecture;
```

4) Finally, the D-type flip flop code and also used in the main code as module to build register and other ones.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity D_flip_flop is
    Port ( clk  : in STD_LOGIC;
           rst  : in STD_LOGIC;
           d    : in STD_LOGIC;
           q    : out STD_LOGIC);
end D_flip_flop;

architecture D_flip_flop_imp of D_flip_flop is
    signal internal_q : STD_LOGIC := '0';

begin
    process(clk, rst)
    begin
        if rst = '1' then
            internal_q <= '0';
        elsif rising_edge(clk) then
            internal_q <= d;
        end if;
    end process;

    q <= internal_q;
end architecture;
```

The main VHDL code

```vhdl
library ieee                ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all    ;
-- ================================================================
entity barrel_shifter is
        port(input_date           : in  std_logic_vector(7 downto 0)      ;
                  operation       : in  std_logic_vector(2 downto 0)      ;
                  clk             : in  std_logic                         ;
                  Number_of_steps : in  std_logic_vector(3 downto 0)      ;
                  output_date     : buffer std_logic_vector(7 downto 0)   ;
                  sel             : buffer  std_logic_vector(2 downto 0)) ;
end barrel_shifter;
-- ================================================================

-- ================================================================
architecture barrel_shifter_imp of barrel_shifter is
-- ======================================
        component mux_8_to_1 is
                port(S:in  std_logic_vector(2 downto 0)   ;
                          F:out std_logic                 ;
                          x:in  std_logic_vector(8 downto 1) ) ;
        end component;
```

```
23 -- ========================================
24         component mux_2_to_1 is
25                 port(s0:in  std_logic                          ;
26                          x :in  std_logic_vector(1 downto 0) ;
27                          f :out std_logic)                     ;
28         end component;
29 -- ========================================
30         component counter is
31     port(
32         C, CLR : in std_logic;
33         Q      : out std_logic_vector(3 downto 0));
34         end component;
35 -- ========================================
36         component D_flip_flop is
37                  Port ( clk : in STD_LOGIC  ;
38                         rst : in STD_LOGIC ;
39                         d : in STD_LOGIC   ;
40                         q : out STD_LOGIC) ;
41         end component;
42 -- ========================================
43         signal int_x   : std_logic_vector(8 downto 1)           ;
44         signal int_q   : std_logic_vector(1 to 8)               ;
45         signal int_sel : std_logic_vector(2 downto 0)           ;
46         signal op      : std_logic_vector(2 downto 0) := "100"  ;
47         signal int_N   : std_logic_vector(3 downto 0) := "0000" ;
48         signal EN              : std_LOGIC := '0';
49         signal int_I   : std_logic_vector(3 downto 0) ;
50         signal t_clk_1 : std_LOGIC := '0';
51         signal test    : std_LOGIC := '0' ;
52 -- ========================================
53         begin
54
55         DFF_X : for i in 1 to 8 generate
56         DFF_x     : D_flip_flop port map(clk,'0',int_x(i),int_q(i));
57         end generate DFF_X;
58
59         DFF_test   : D_flip_flop port map(t_clk_1,'0','1',EN);
60         DFF_test1  : D_flip_flop port map(clk,'0','1',test);
61 -- ====================================================
62         MUX2_A :mux_2_to_1 port map(s0=>test,x(0)=>'0',x(1)=>op(0),f=>sel(0));
63         MUX2_B :mux_2_to_1 port map(s0=>test,x(0)=>'0',x(1)=>op(1),f=>sel(1));
64         MUX2_C :mux_2_to_1 port map(s0=>test,x(0)=>'1',x(1)=>op(2),f=>sel(2));
65 -- ====================================================
66         MUX8_1 : mux_8_to_1
67    port map (
68        S(0) => int_sel(0)    ,
69        S(1) => int_sel(1)    ,
70        S(2) => int_sel(2)    ,
71        F    => int_x(1)      ,
72        X(1) => int_q(1)      ,
73        X(2) => '0'           ,
74        X(3) => int_q(1)      ,
75        X(4) => int_q(8)      ,
76        X(5) => input_date(7),
77        X(6) => int_q(2)      ,
78        X(7) => int_q(2)      ,
79        X(8) => int_q(2)
80    );
81
82         mux : for i in 0 to 5 generate
83                 MUX8_X : mux_8_to_1
84     port map (
```

```
 85         S(0) => int_sel(0)      ,
 86         S(1) => int_sel(1)      ,
 87         S(2) => int_sel(2)      ,
 88         F    => int_x(2+i)      ,
 89         X(1) => int_q(2+i)      ,
 90         X(2) => int_q(1+i)      ,
 91         X(3) => int_q(1+i)      ,
 92         X(4) => int_q(1+i)      ,
 93         X(5) => input_date(6-i),
 94         X(6) => int_q(3+i)      ,
 95         X(7) => int_q(3+i)      ,
 96         X(8) => int_q(3+i)
 97     );
 98
 99         end generate mux;
100
101                 MUX8_8 : mux_8_to_1
102     port map (
103         S(0) => int_sel(0)    ,
104         S(1) => int_sel(1)    ,
105         S(2) => int_sel(2)    ,
106         F    => int_x(8)      ,
107         X(1) => int_q(8)      ,
108         X(2) => int_q(7)      ,
109         X(3) => int_q(7)      ,
110         X(4) => int_q(7)      ,
111         X(5) => input_date(0),
112         X(6) => '0'           ,
113         X(7) => int_q(8)      ,
114         X(8) => int_q(1)
115     );
116 -- ====================================================
117             counter_N : counter port map (
118                         C     => clk,
119                         CLR   => '0',
120                         Q     => int_I );
121 -- ====================================================
122                 op <= operation ;
123                 int_N       <= std_logic_vector(unsigned(Number_of_steps) + 1)
124
125                 t_clk_1     <= (int_N(0)xnor int_I(0)) and (int_N(1)xnor
126 int_I(1)) and (int_N(2)xnor int_I(2)) and (int_N(3)xnor int_I(3));
127                 int_sel(0)  <= (not EN) and sel(0);
128                 int_sel(1)  <= (not EN) and sel(1);
129                 int_sel(2)  <= (not EN) and sel(2);
130                 output_date <= int_q;
131 end architecture;
```

Conclusion

So, in the final we want to thank the Doctors and the teaching assistant for helping us to build huge and awesome project like this,

In fact, this project helps us to improve our technical and soft skills like team work and other.

Thanks a lot 😊