2nd communication

Logic 2 project report

Logic_project_70

Super Register

Team members

Engy Ashraf 70

Andrew Morcos 71

Mina Nabil 296

Youstina Abd Elmassih 325

Super register project

1st Steps

We asked to design super register that performs eight operations:

- Load data
- Shift data to left
- Shift data to right
- Rotate data to left
- Rotate data to right
- Store data (store its state)
- Count up
- Count down

And to get first output of these operations according to control input -consists of three bits- and second output according to the first output.

So, we thought of this project in behavioral way, this by creating an entity containing all these ports:

- Data_in (8 bits vector input).
- Control (3 bits vector input).
- Sh_r,Sh_l ,clk(1 bit input).
- Data_out1 (8 bits vector output).
- Data_out2 (1 bit vector output).

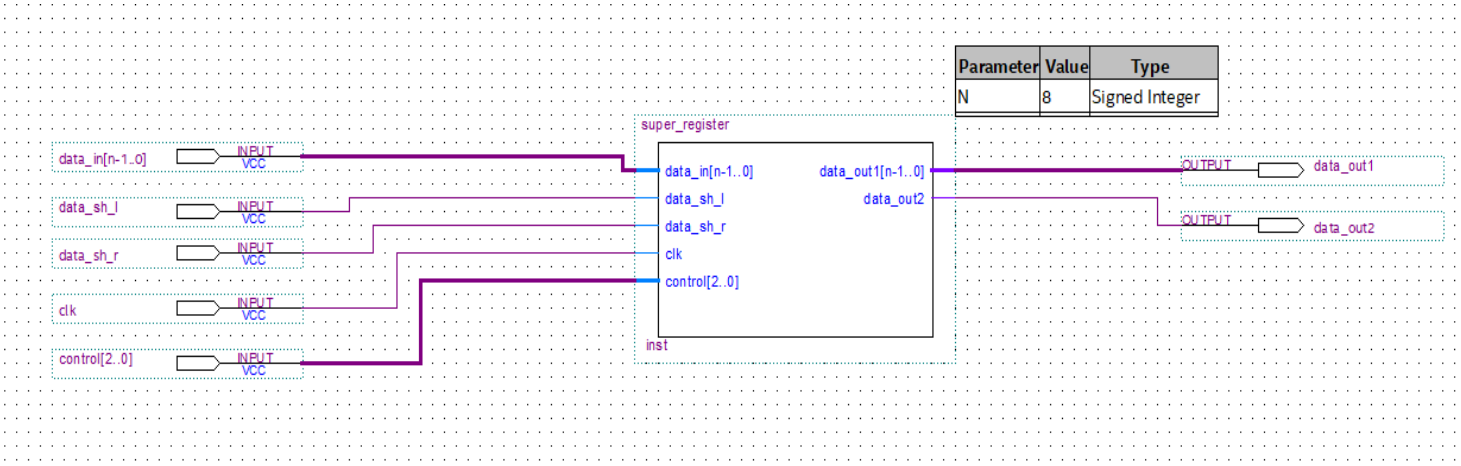
Then creating its architecture which contains:

- Process.

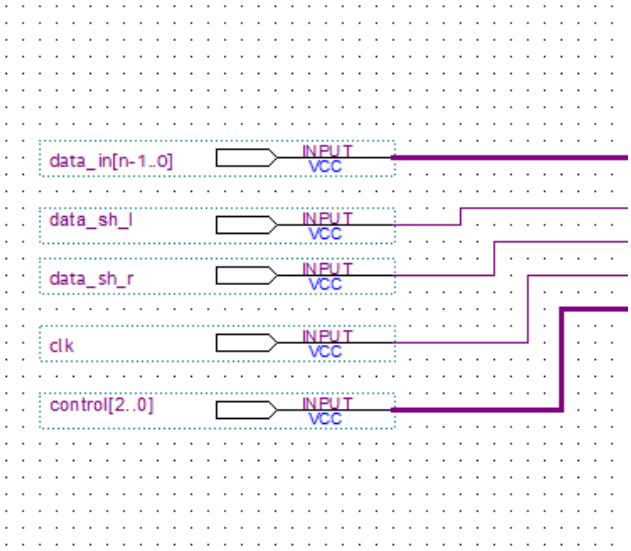
In the process we wrote our algorithm which depends on using case statements and we switched between statements using the control input, so we converted 8 operations into 8 case statements, in each case we wrote the code which performs this case (operation) and as process is sensitive to CLK and control so at each rising edge the process checks which case should be done.

2nd schematic

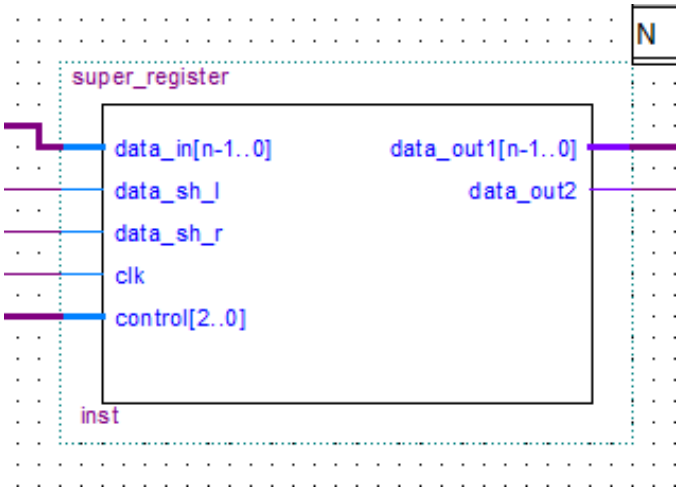
By using QUARTUS program we converted our code into this block diagram



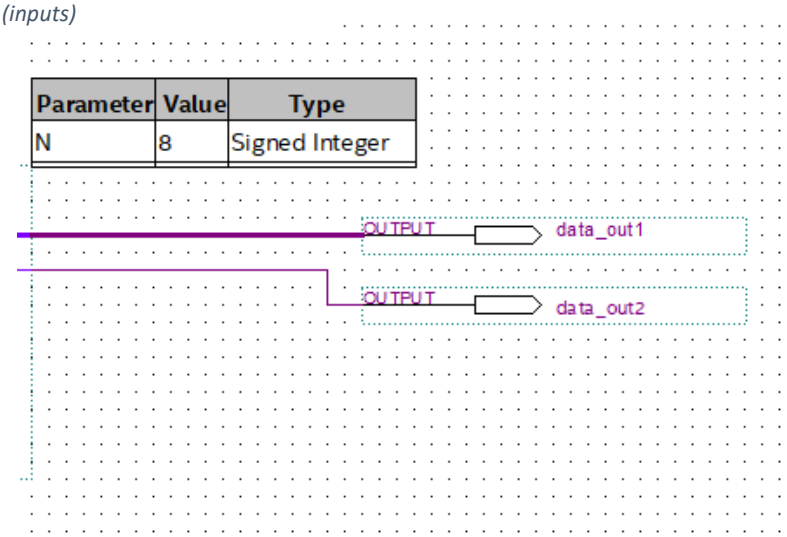
which consists of three main parts:



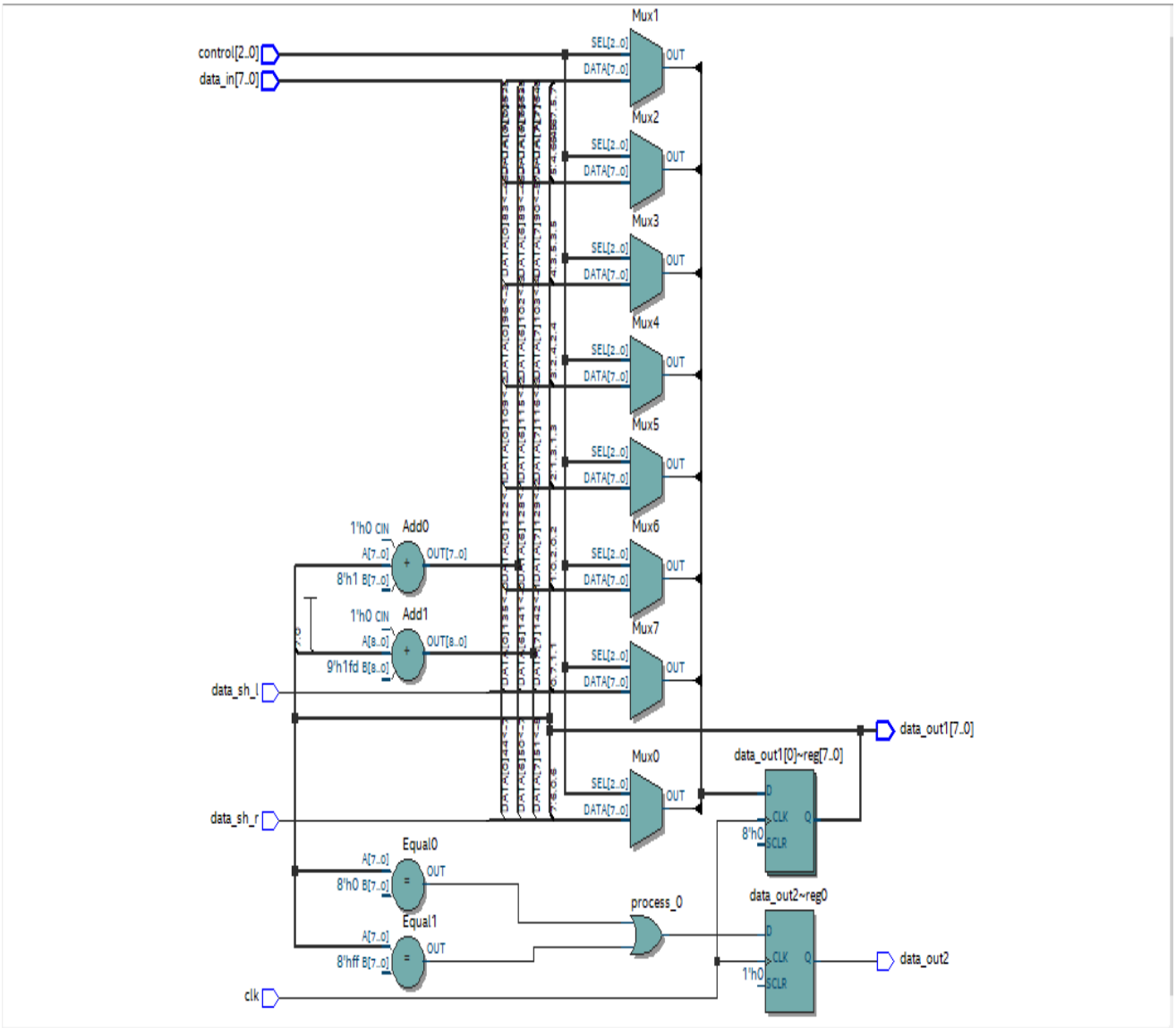
part1(inputs)



Part2(our operational block)



Part3(outputs)


Mux1
Mux2
Mux3
Mux4
Mux5
Mux6
Mux7
Mux0
SEL[2..0]
DATA[7..0]
OUT
control[2..0]
data_in[7..0]
Add0
Add1
1'h0 CIN
A[7..0]
8'h1 B[7..0]
OUT[7..0]
A[8..0]
9'h1fd B[8..0]
OUT[8..0]
data_sh_l
data_sh_r
Equal0
Equal1
8'h0 B[7..0]
8'hff B[7..0]
process_0
clk
data_out1[7..0]
data_out1[0]~reg[7..0]
data_out2~reg0
data_out2
D
CLK
Q
SCLR
8'h0
1'h0

3rd VHDL code

Text Editor - C:/intelFPGA_lite/17.1/super_register - super_register - [super_reg.vhd]

File Edit View Project Processing Tools Window Help

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity super_register is
6     generic ( N : integer := 8);
7     port( data_in : in std_logic_vector(N-1 downto 0);
8           data_sh_l,data_sh_r,clk : in std_logic;
9           control : in std_logic_vector(2 downto 0);
10           data_out1: buffer std_logic_vector(N-1 downto 0);
11           data_out2: out std_logic);
12 end super_register;
13
14 architecture behavoir of super_register is
15 begin
16    process(control,clk)
17       variable tmp : std_logic;
18       variable zero_all : std_logic_vector(N-1 downto 0) := (others => '0') ;
19       variable one_all : std_logic_vector(N-1 downto 0) := (others => '1');
20    begin
21       if clk'event and clk = '1' then
22          case control is
23             when "000" => --load new data
24                data_out1 <= data_in;
25             when "001" => -- right shift
26                l_shift: for i in 0 to N-2 loop
27                data_out1(i) <= data_out1(i+1);
28                end loop;
29                data_out1(N-1) <= data_sh_r;
30             when "010" => -- left shift
31                r_shift: for i in N-1 downto 1 loop
32                data_out1(i) <= data_out1(i-1);
33                end loop;
34                data_out1(0) <= data_sh_l;
35             when "011" => -- left rotate
36                tmp := data_out1(0);
37                l_rotate: for i in 0 to N-2 loop
38                data_out1(i) <= data_out1(i+1);
39                end loop;
40                data_out1(N-1) <= tmp;
41             when "100" => -- right rotate
42                tmp := data_out1(N-1);
43                r_rotate: for i in N-1 downto 1 loop
44                data_out1(i) <= data_out1(i-1);
45                end loop;
46                data_out1(0) <= tmp;
47             when "101" => --store present state
48                data_out1<=data_out1;
49             when "110" => -- count up
50                data_out1 <= data_out1 + 1 ;
51             when others => -- count down
52                data_out1 <= data_out1 - 1 ;
53          end case;
54
55          if(data_out1 = zero_all or data_out1 = one_all) then
56             data_out2 <= '1';
57          else
58             data_out2 <= '0';
59          end if;
60
61       end if;
62    end process;
63
64 end behavoir;
65
66
67
```

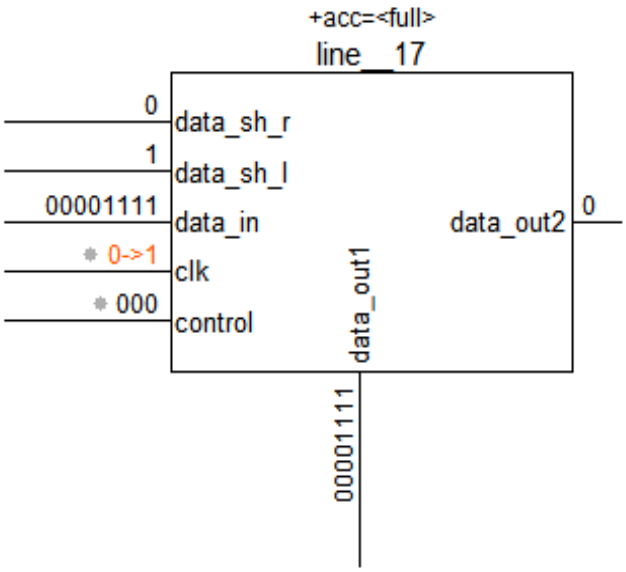
This is libraries declaration, we use here three libraries

- Ieee
- Ieee.std_logic_1164.all.
- Ieee.std_logic_unsigned.all (we use this to perform adding on a vector)

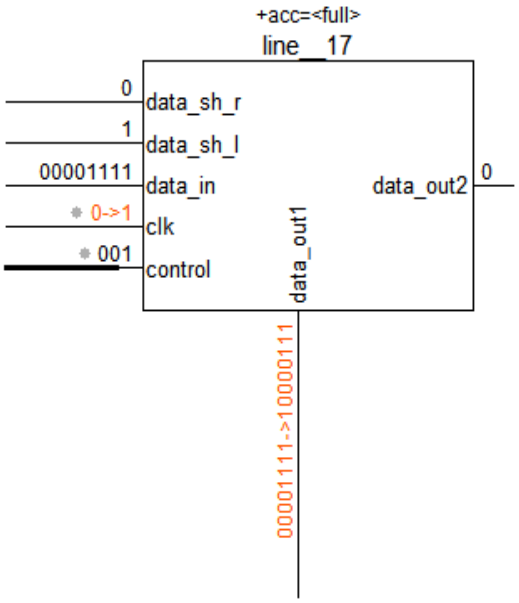
1) Entity declaration
   - Generic (It gives us ability to extend our register to more than 8 bits)
   - Ports
2) Architecture
   - Process
     a) Variable
        - Tmp (we use it to save the last or first bit in rotating mode)
     b) constants
        - Zero_all (we use it to compare between it and data_out having same size)
        - One_all (we use it to compare between it and data_out having same size)
     c) Process body
        - Case statements
          i) Loading data
          ii) Right shifting using for loop to pass each bit to its right adjacent one
          iii) Left shifting using for loop to pass each bit to its left adjacent one
          iv) Left rotating using for loop like left shifting but without inserting external bit
          v) Right rotating using for loop like right shifting but without inserting external bit
          vi) Storing data by passing output again to the output using buffer
          vii) Counting up by simply adding one to the output state
          viii) Counting down by simply subtracting one from the output state

        - If statement
          Used to assign to the second output 1 if the first output is 11111111 or
          00000000 else it assigns it with 0

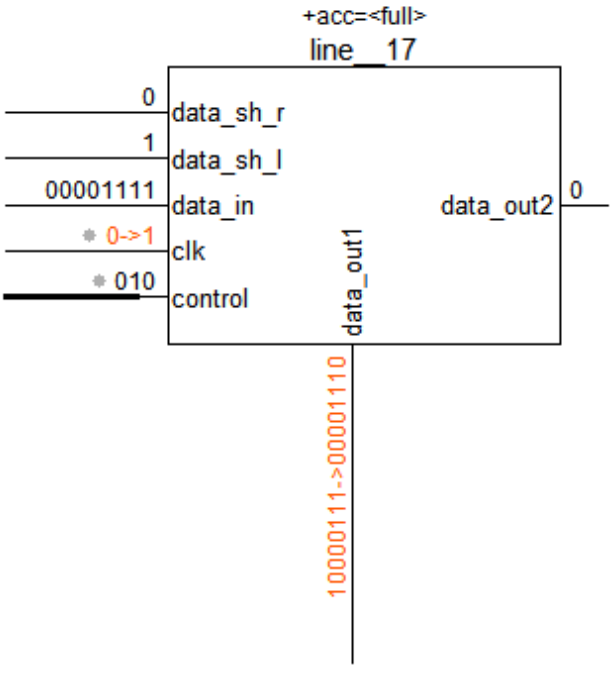
4th simulations



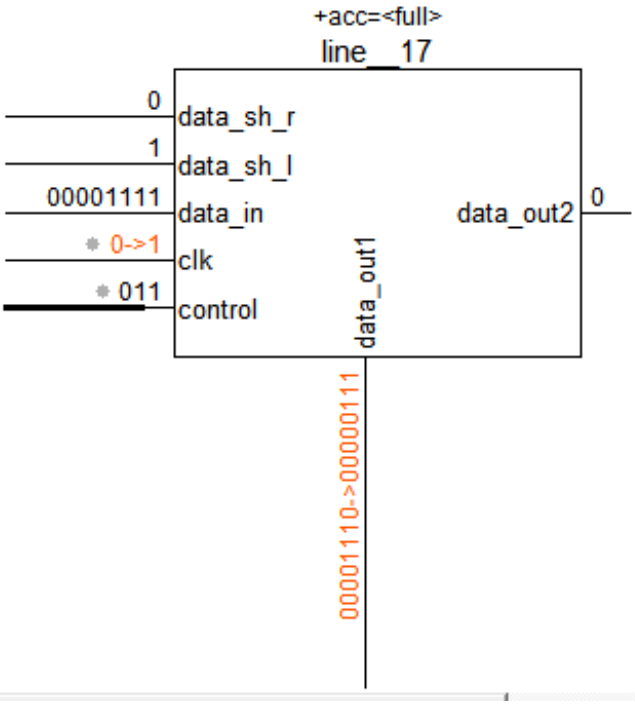
Insert data



Shift right with 1

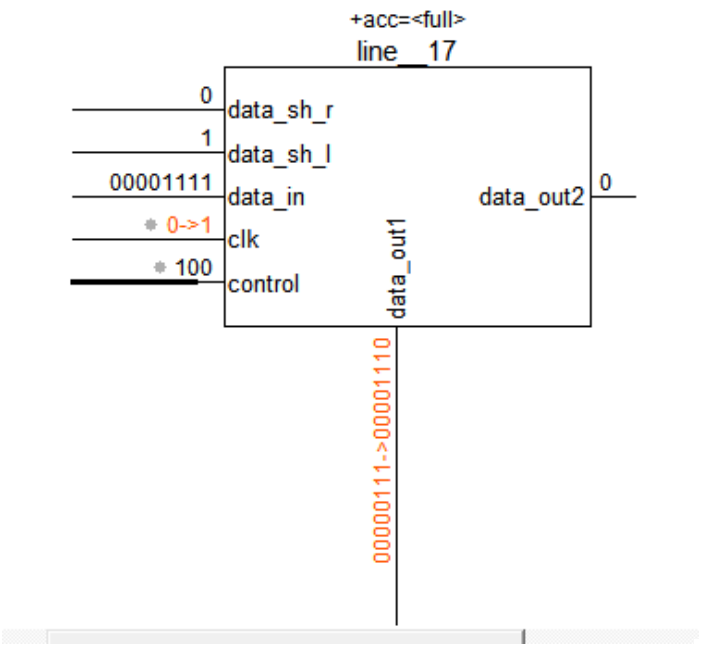


Shift left with 0

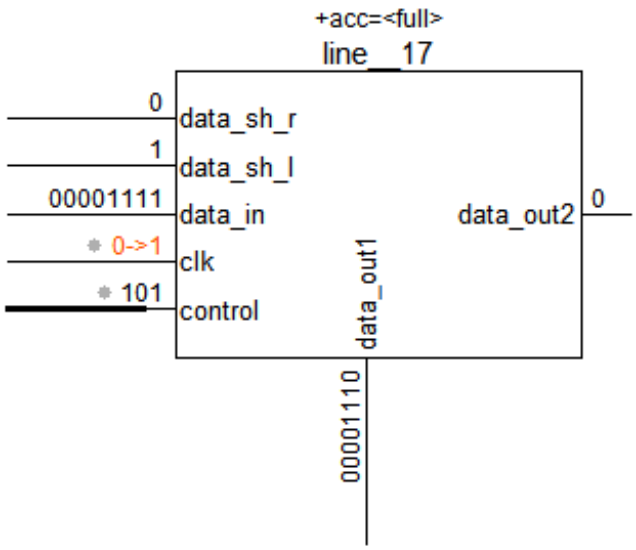


Rotate right

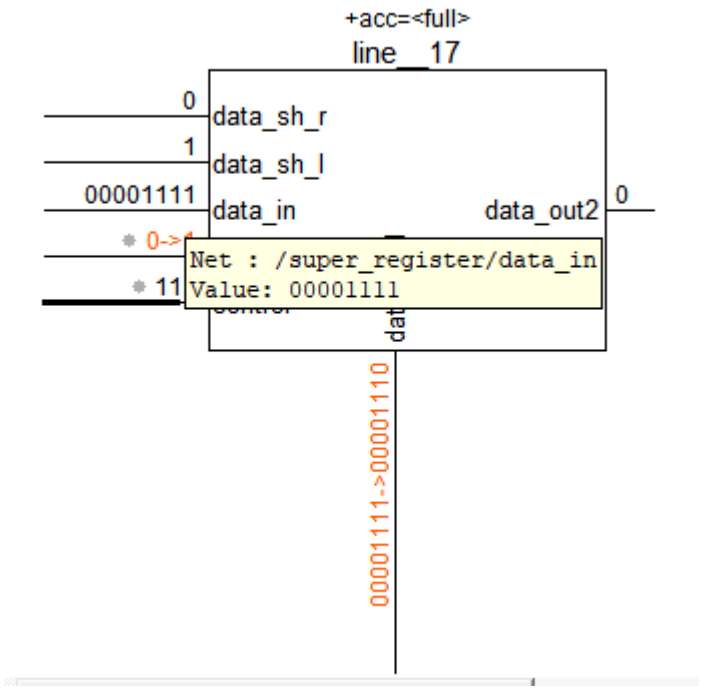
SUPER REGISTER



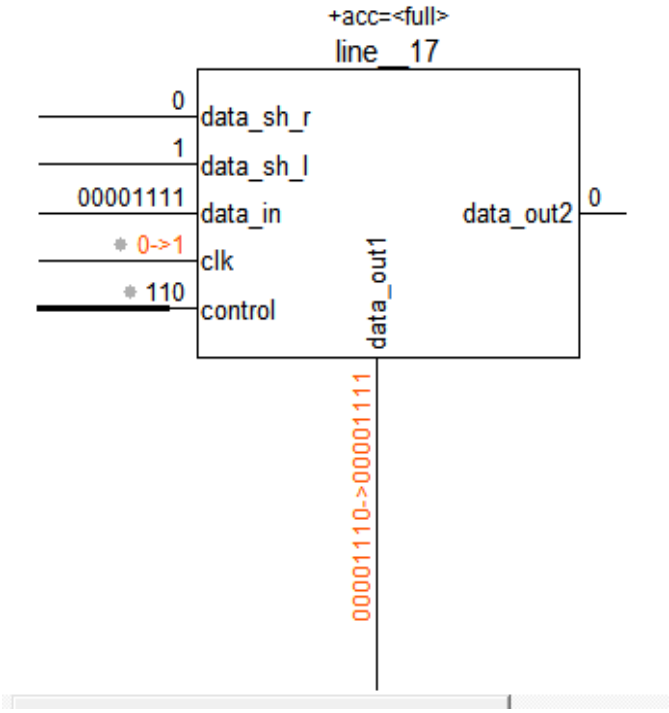
Rotate right



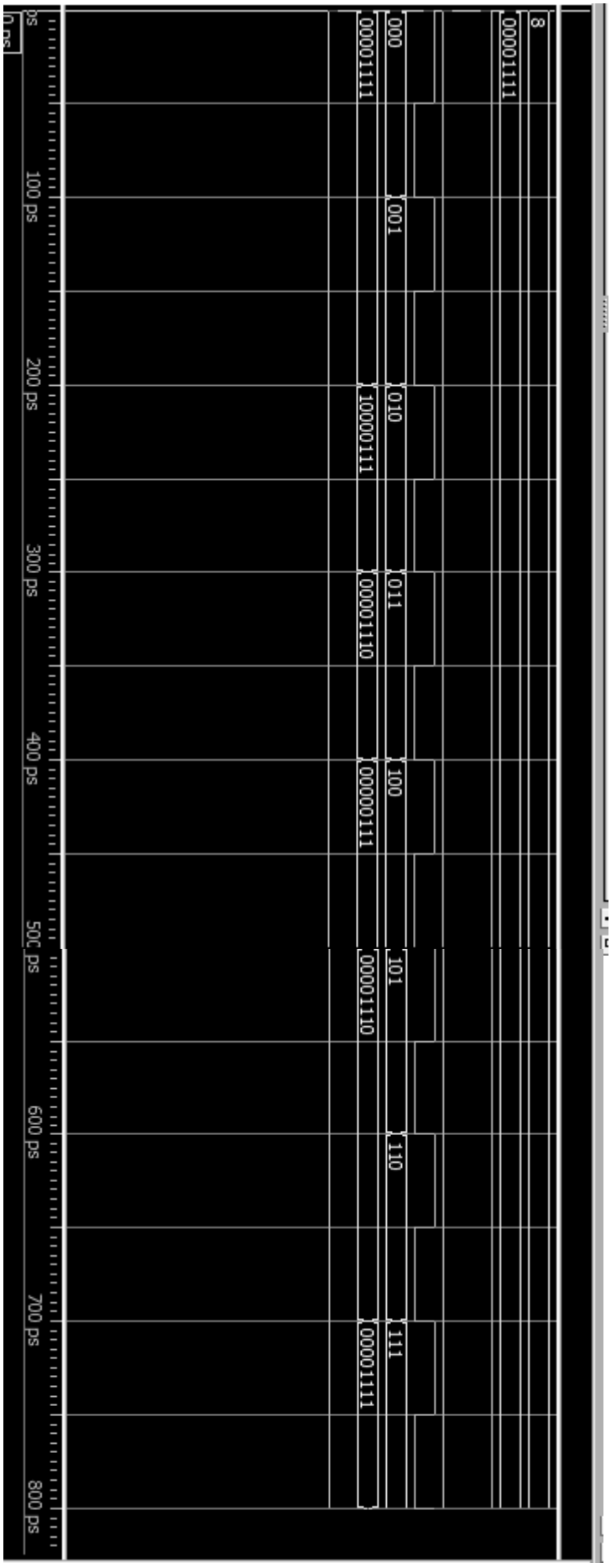
Store data



Count down



Count up


8
00001111
000
001
010
011
100
101
110
111
00001111
10000111
00001110
00000111
00001110
00001111
0 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
700 ps
800 ps