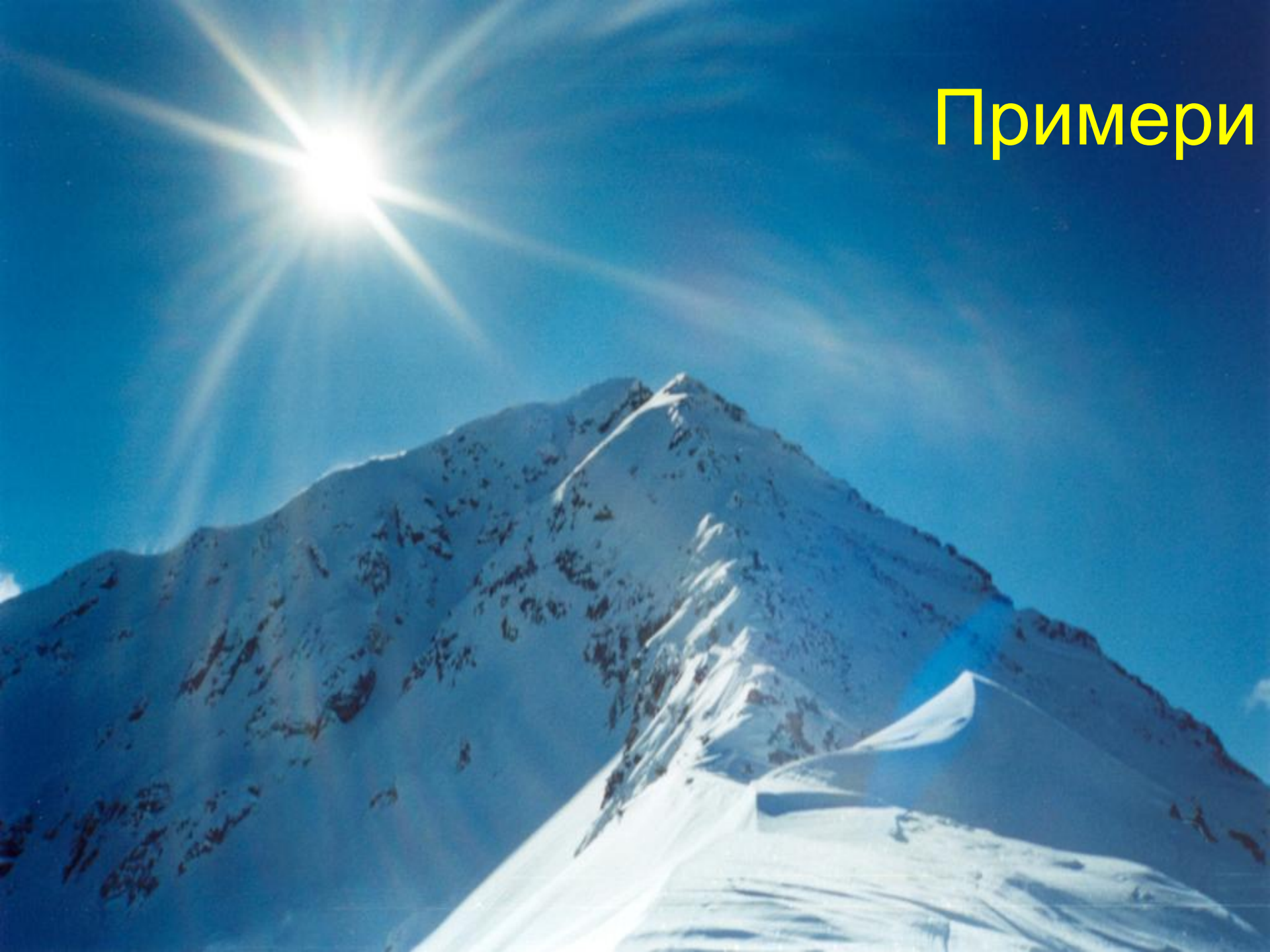
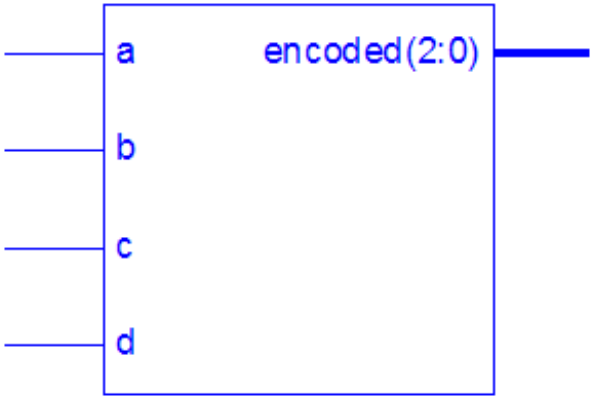
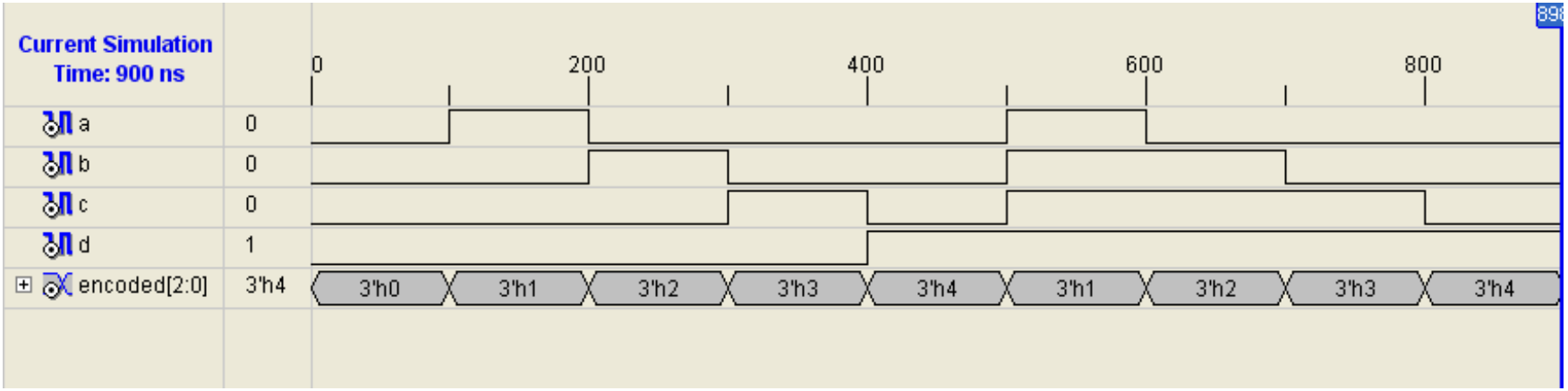
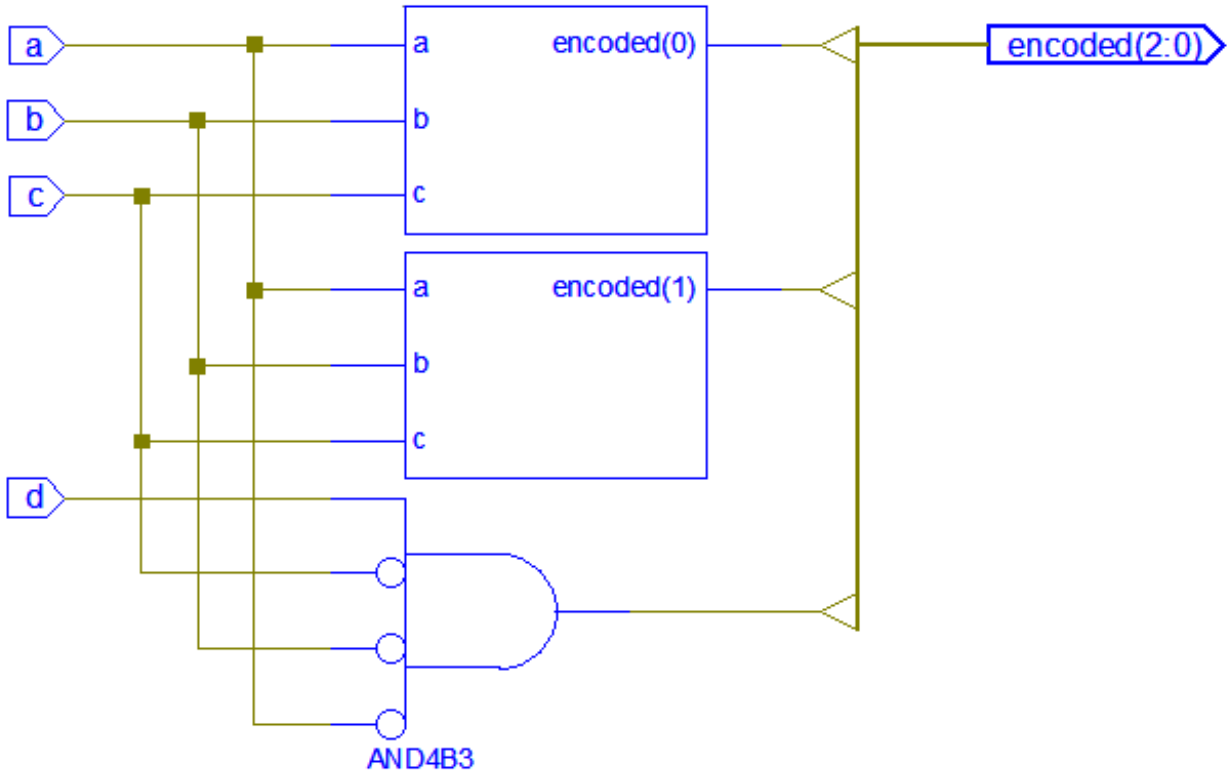

Примери

Приоритетен кодер

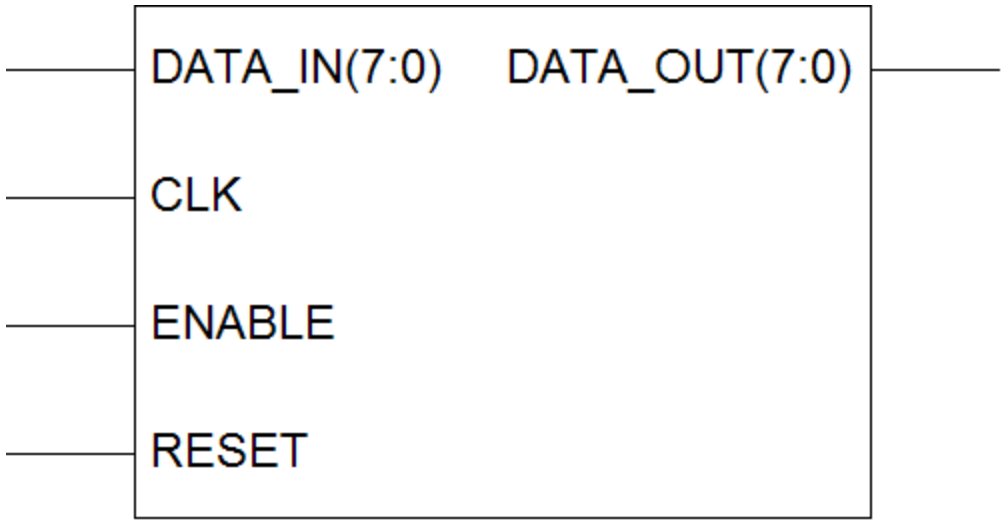


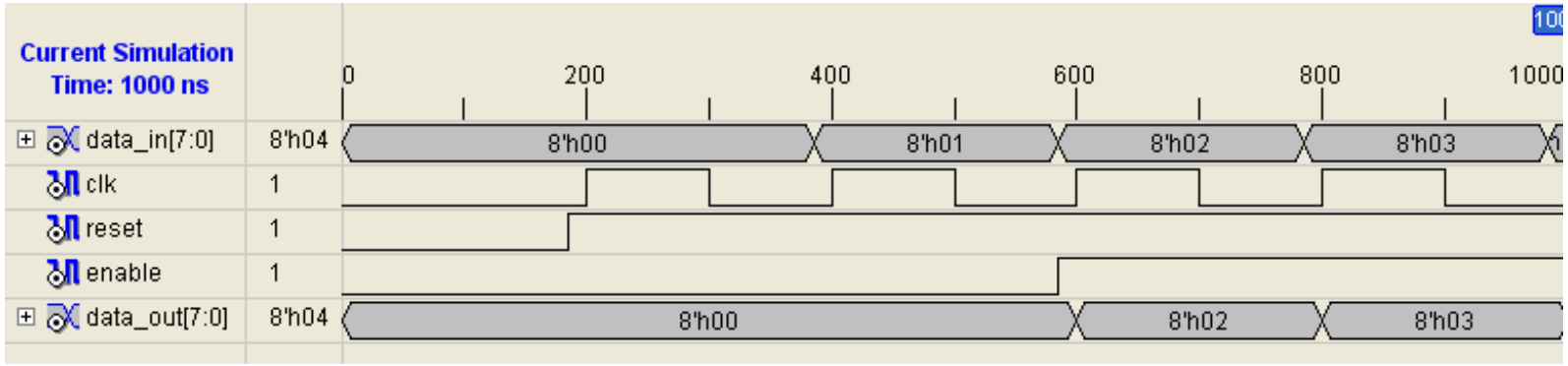
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity priority is
    Port ( a,b,c,d : in  STD_LOGIC;
           encoded : out STD_LOGIC_VECTOR(2 downto 0));
end priority;

architecture Behavioral of priority is

begin
process (a,b,c,d)
begin
    if a = '1' then
        encoded <= "001";
    elsif b = '1' then
        encoded <= "010";
    elsif c = '1' then
        encoded <= "011";
    elsif d = '1' then
        encoded <= "100";
    else
        encoded <= "000";
    end if;
end process;

end Behavioral;
```


Current Simulation Time: 900 ns
0
200
400
600
800
a
0
b
0
c
0
d
1
encoded[2:0]
3'h4
3'h0
3'h1
3'h2
3'h3
3'h4
3'h1
3'h2
3'h3
3'h4

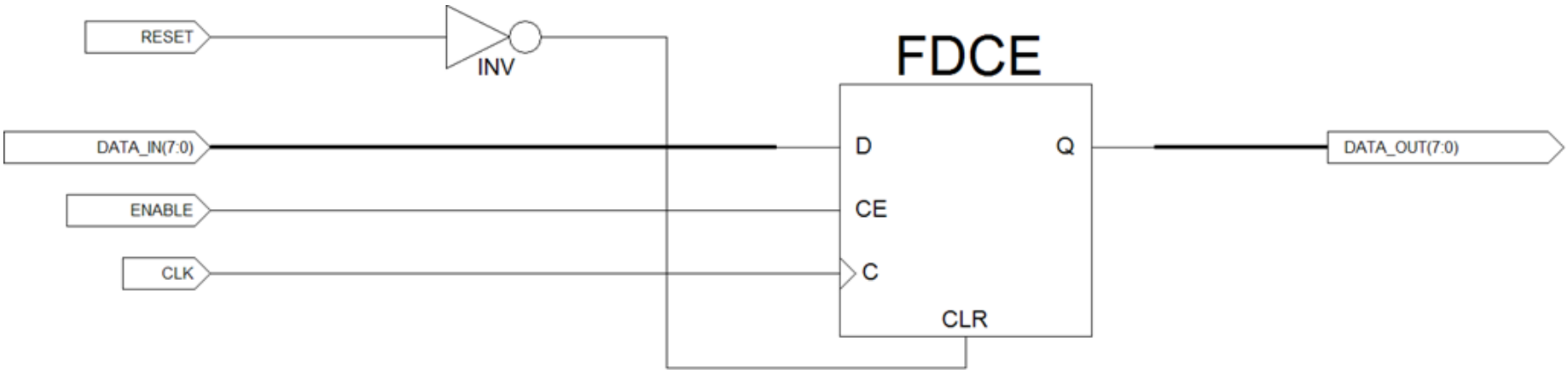

a
b
c
d
a
b
c
encoded(0)
a
b
c
encoded(1)
encoded(2:0)
AND4B3

Регистър (N-битов)

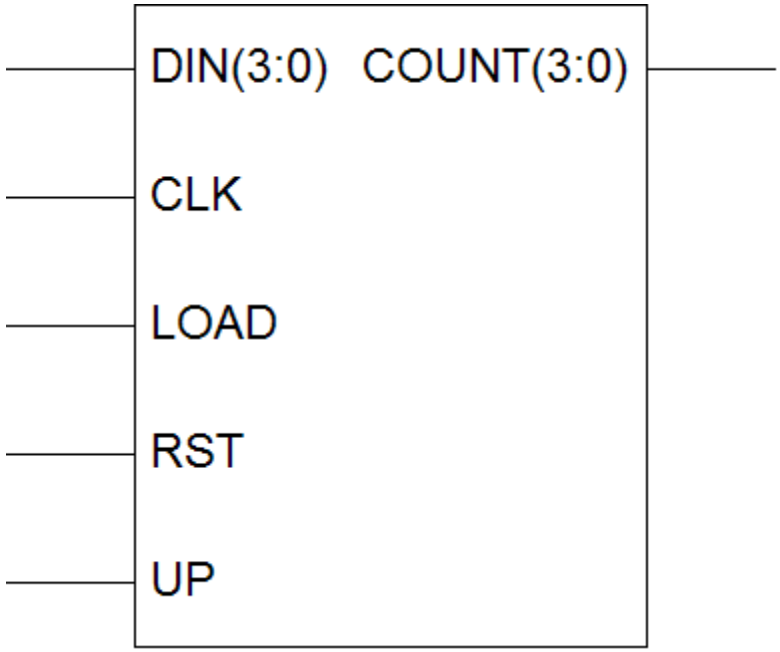


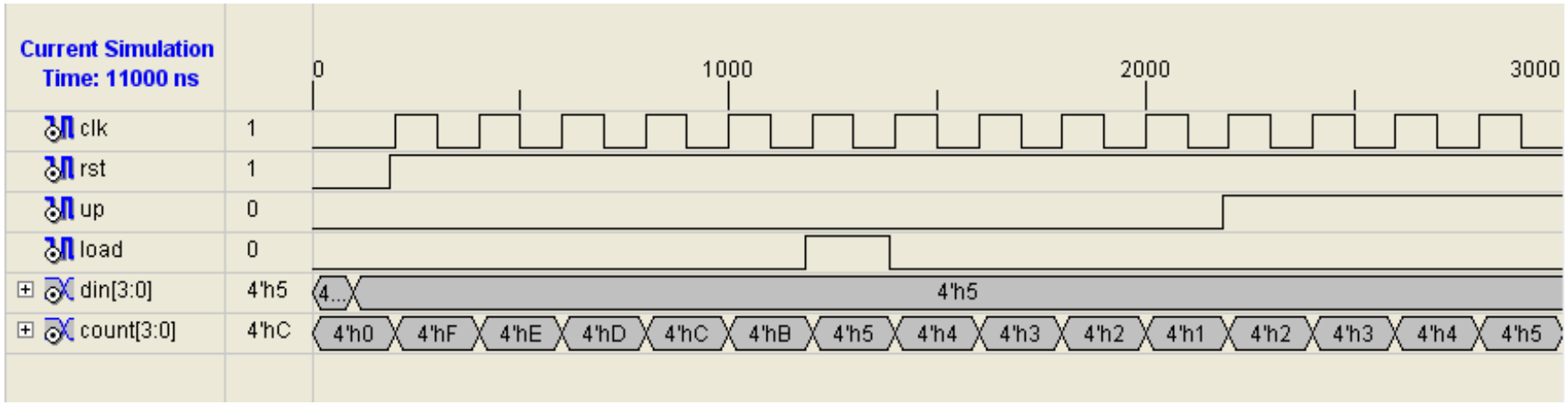

Current Simulation Time: 1000 ns
0
200
400
600
800
1000
data_in[7:0]
8'h04
8'h00
8'h01
8'h02
8'h03
clk
1
reset
1
enable
1
data_out[7:0]
8'h04
8'h00
8'h02
8'h03

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity reg is
Generic (N : integer := 8);
Port ( DATA_IN : in  STD_LOGIC_VECTOR (N-1 downto 0);
           CLK : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           ENABLE : in  STD_LOGIC;
           DATA_OUT : out  STD_LOGIC_VECTOR (N-1 downto 0));
end reg;

architecture Behavioral of reg is

begin

process (CLK, RESET) begin
     if RESET = '0' then
          DATA_OUT <= (others=>'0');
     elsif rising_edge(CLK) then
          if ENABLE = '1' then
               DATA_OUT <= DATA_IN;
          end if;
     end if;
end process;

end Behavioral;
```

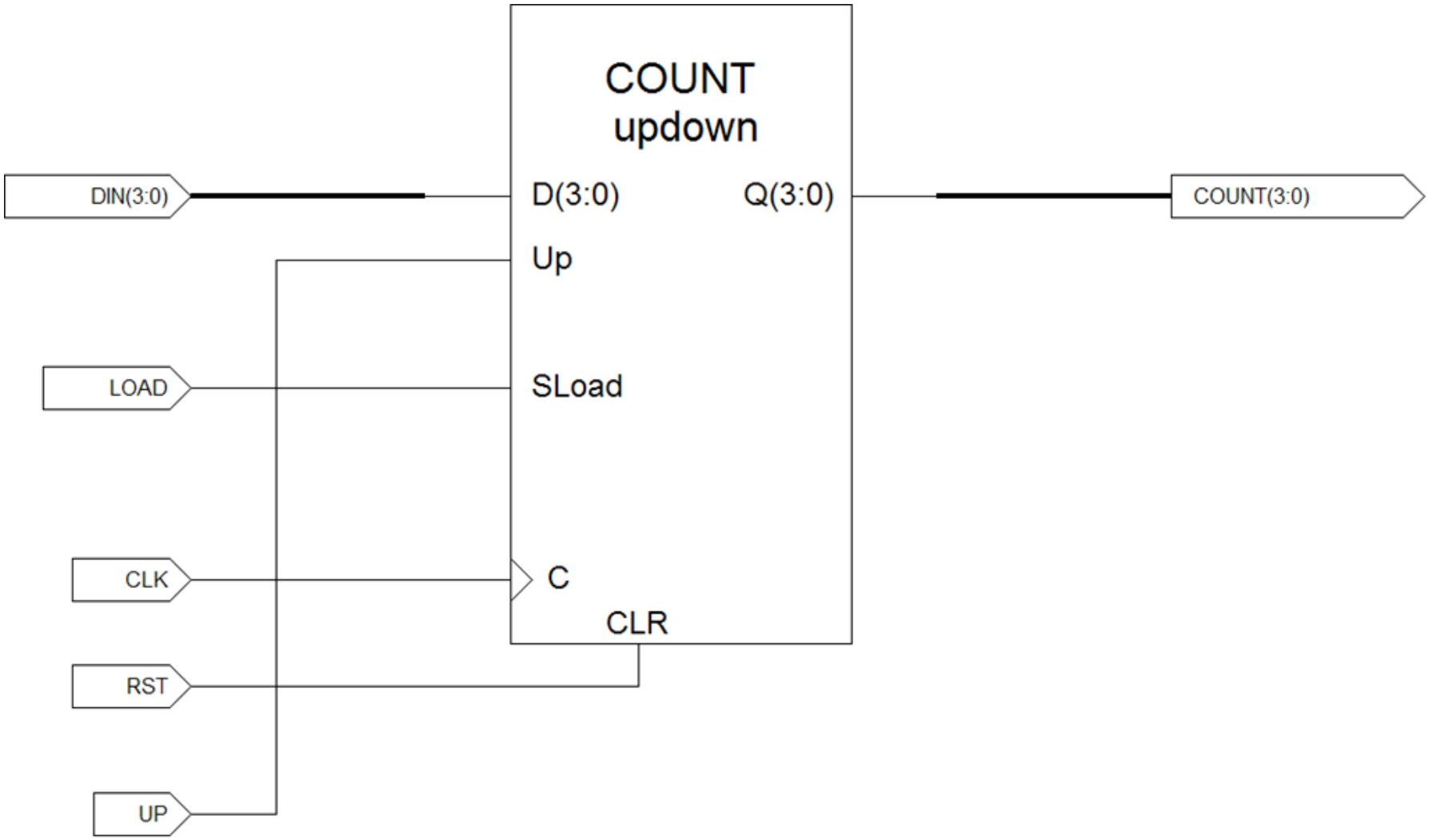

RESET
INV
FDCE
DATA_IN(7:0)
D
Q
DATA_OUT(7:0)
ENABLE
CE
CLK
C
CLR

Брояч (N-битов)

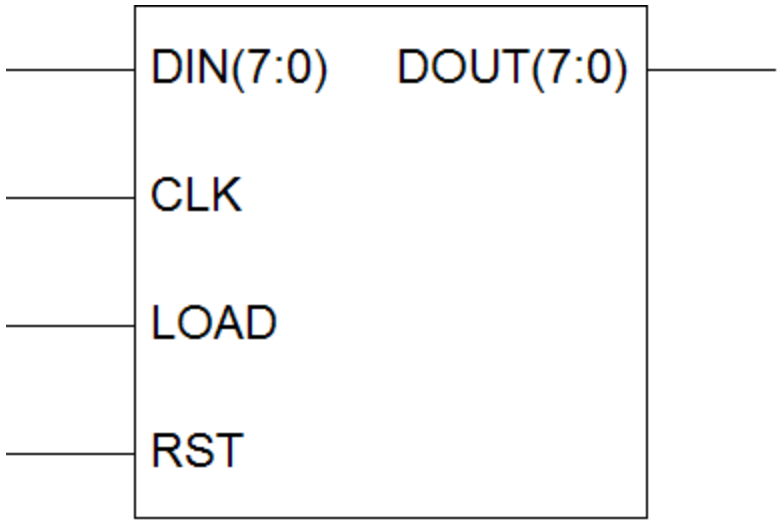



Current Simulation Time: 11000 ns
0
1000
2000
3000
clk
1
rst
1
up
0
load
0
din[3:0]
4'h5
4...
4'h5
count[3:0]
4'hC
4'h0
4'hF
4'hE
4'hD
4'hC
4'hB
4'h5
4'h4
4'h3
4'h2
4'h1
4'h2
4'h3
4'h4
4'h5

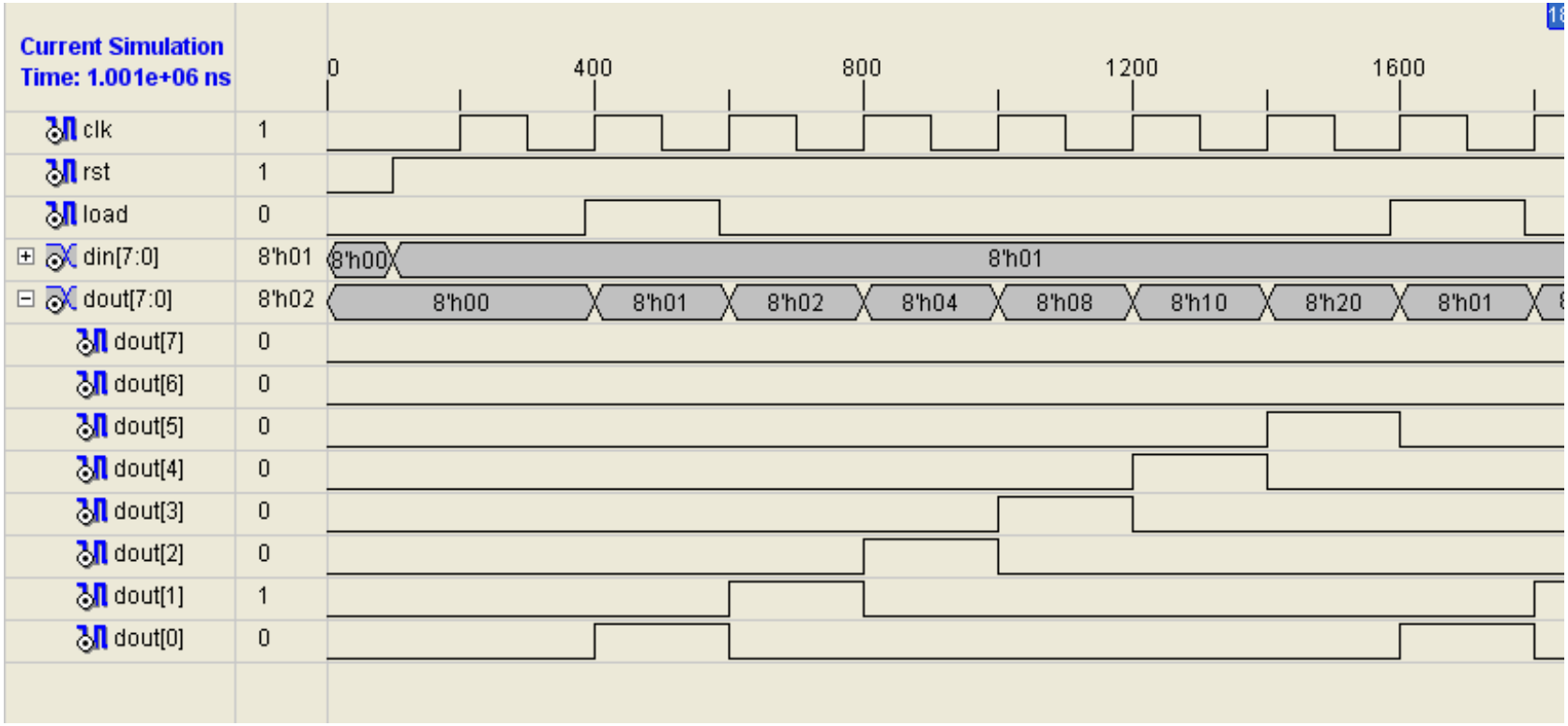
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity revcnt is
generic (
      N : integer := 4;
      RST_ACTIVE : std_logic := '0';
      LOAD_ACTIVE : std_logic := '1');
port (
      CLK, RST, UP, LOAD: in std_logic;
      DIN: in std_logic_vector(N-1 downto 0);
      COUNT: out std_logic_vector(N-1 downto 0));
end revcnt;

architecture Behavioral of revcnt is
signal TMP: std_logic_vector(N-1 downto 0);
begin

process (CLK,RST) begin
     if RST = RST_ACTIVE then
        TMP <= (others=>'0');
     elsif rising_edge(CLK) then
        if LOAD = LOAD_ACTIVE then
           TMP <= DIN;
        elsif UP = '1' then
           TMP <= TMP + 1;
        else
           TMP <= TMP - 1;
       end if;
     end if;
  end process;
  COUNT <= TMP;
end Behavioral;
```


COUNT
updown
DIN(3:0)
D(3:0)
Q(3:0)
COUNT(3:0)
Up
LOAD
SLoad
CLK
C
CLR
RST
UP

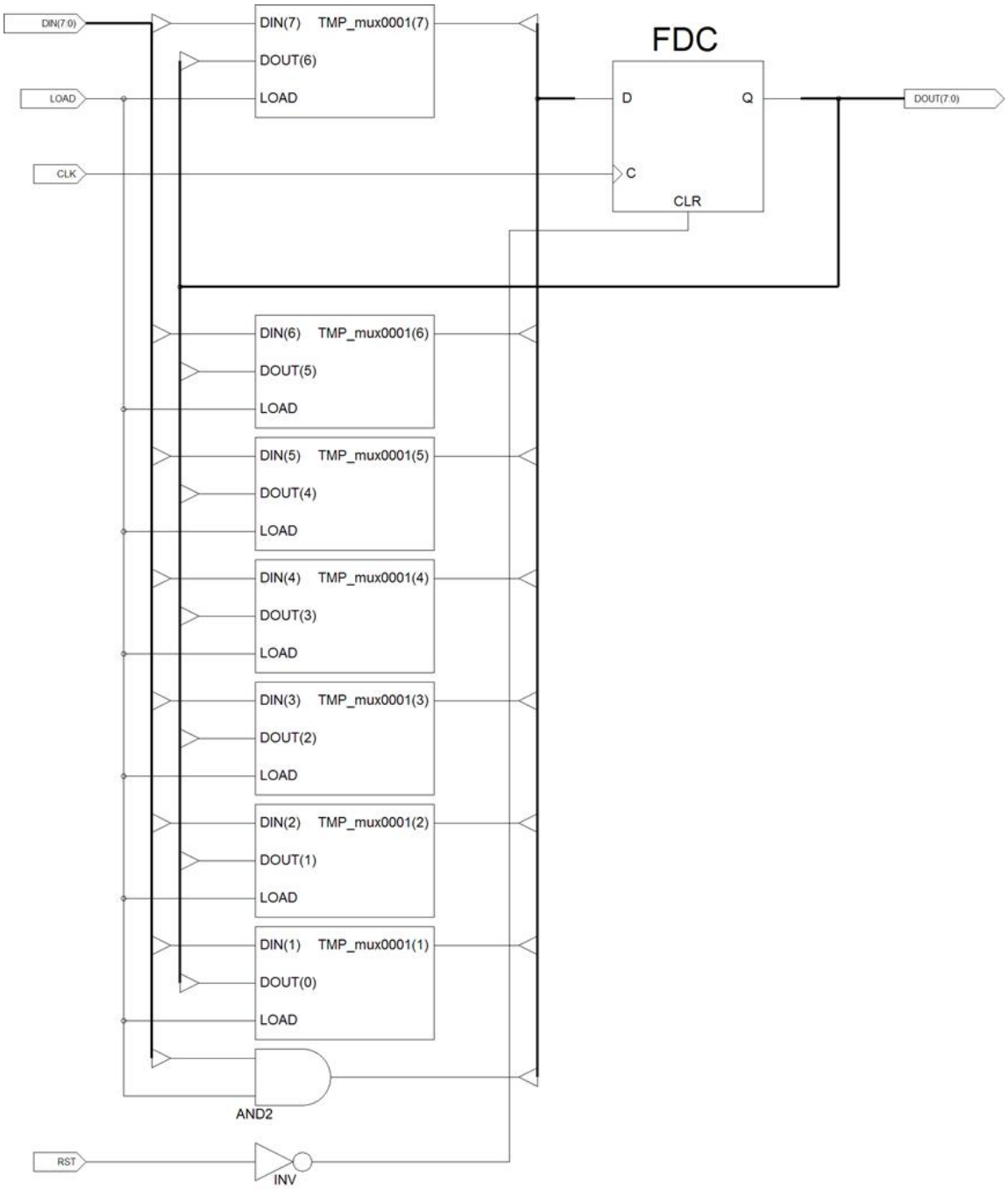
Премества щ регистър (N-битов)



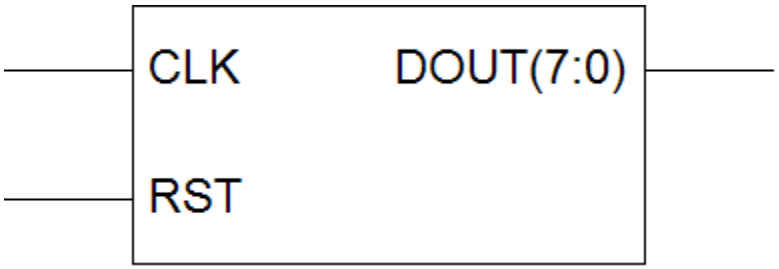
Преместване наляво и допълване с нули.


Current Simulation Time: 1.001e+06 ns
0
400
800
1200
1600
clk
1
rst
1
load
0
din[7:0]
8'h01
8'h00
8'h01
dout[7:0]
8'h02
8'h00
8'h01
8'h02
8'h04
8'h08
8'h10
8'h20
8'h01
dout[7]
0
dout[6]
0
dout[5]
0
dout[4]
0
dout[3]
0
dout[2]
0
dout[1]
1
dout[0]
0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shiftreg is
generic (
      N : integer := 8;
      RST_ACTIVE : std_logic := '0';
      LOAD_ACTIVE : std_logic := '1');
port (
      CLK, RST, LOAD: in std_logic;
      DIN: in std_logic_vector(N-1 downto 0);
      DOUT: out std_logic_vector(N-1 downto 0));
end shiftreg;

architecture Behavioral of shiftreg is
signal TMP: std_logic_vector(N-1 downto 0);
begin

process (CLK,RST) begin
     if RST = RST_ACTIVE then
        TMP <= (others=>'0');
     elsif rising_edge(CLK) then
        if LOAD = LOAD_ACTIVE then
           TMP <= DIN;
        else
           TMP <= TMP(N-2 downto 0) & '0'; -- shift left, zero extend
       end if;
     end if;
  end process;
  DOUT <= TMP;
end Behavioral;
```

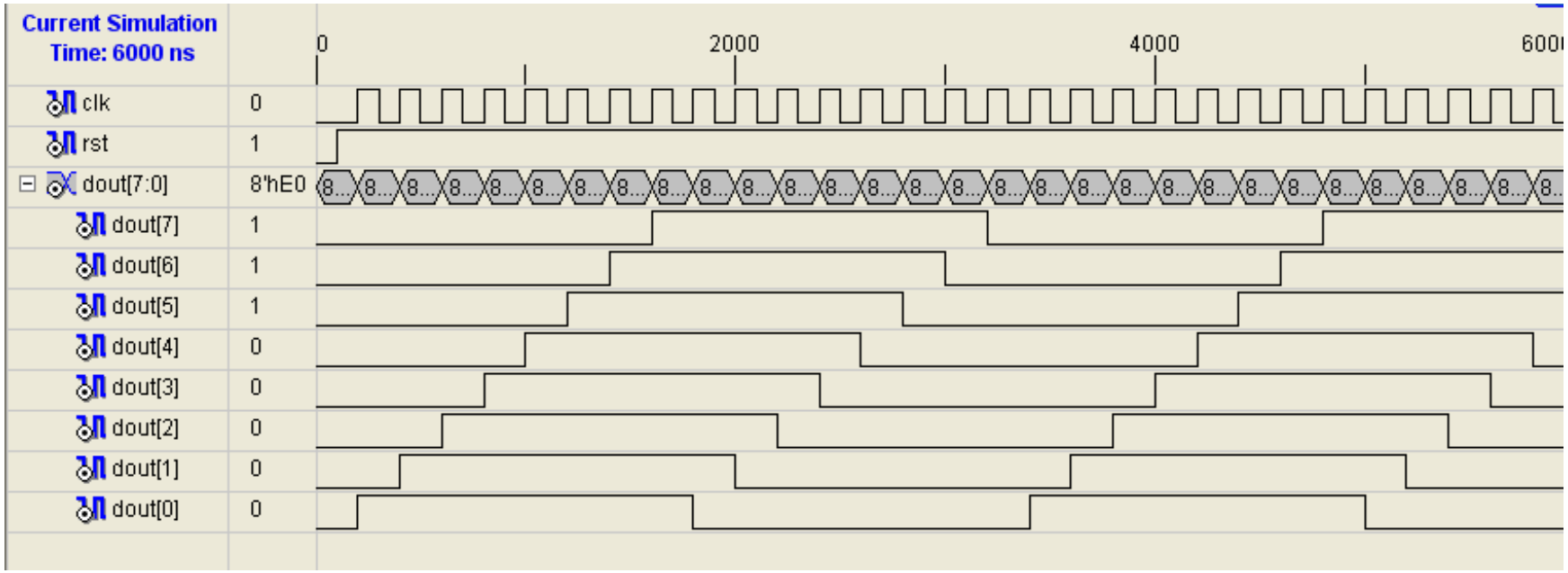

DIN(7:0)
LOAD
CLK
RST
DIN(7) TMP_mux0001(7)
DOUT(6)
LOAD
DIN(6) TMP_mux0001(6)
DOUT(5)
LOAD
DIN(5) TMP_mux0001(5)
DOUT(4)
LOAD
DIN(4) TMP_mux0001(4)
DOUT(3)
LOAD
DIN(3) TMP_mux0001(3)
DOUT(2)
LOAD
DIN(2) TMP_mux0001(2)
DOUT(1)
LOAD
DIN(1) TMP_mux0001(1)
DOUT(0)
LOAD
AND2
INV
FDC
D
Q
C
CLR
DOUT(7:0)

Брояч на Джонсън (N-битов)

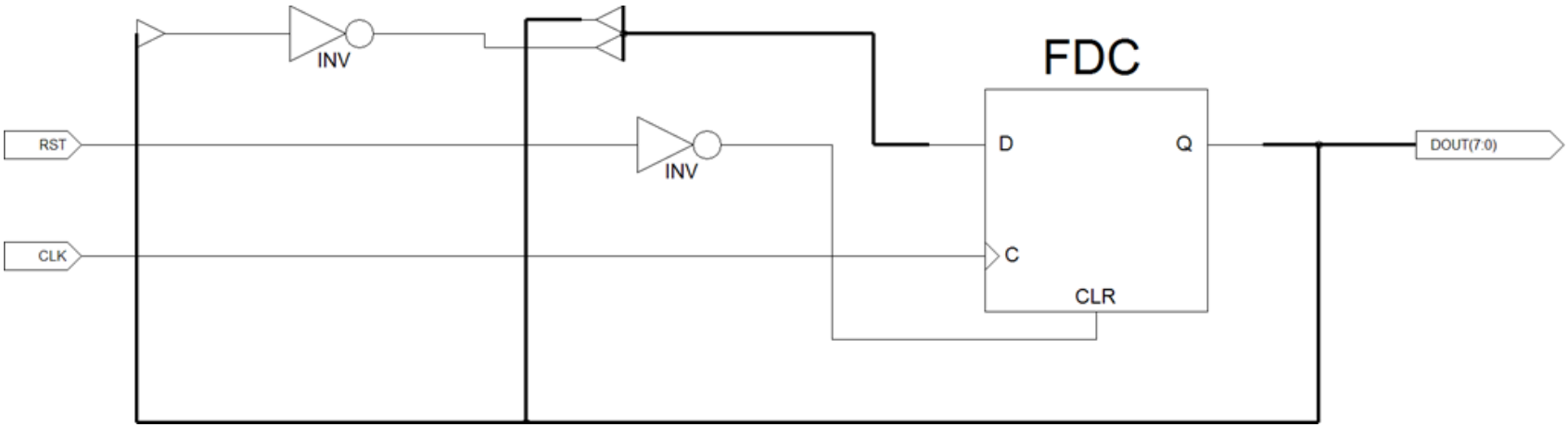


Кръгов преместващ регистър с инвертирана обратна връзка.

Брояч на Джонсън (N-битов)



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity johnson is
generic (
     N : integer := 8;
     RST_ACTIVE : std_logic := '0');
port (
     CLK, RST: in std_logic;
     DOUT: out std_logic_vector(N-1 downto 0));
end johnson;

architecture Behavioral of johnson is
signal TMP: std_logic_vector(N-1 downto 0);
begin

process (CLK,RST) begin
     if RST = RST_ACTIVE then
        TMP <= (others=>'0');
     elsif rising_edge(CLK) then
        TMP <= TMP(N-2 downto 0) & not TMP(N-1); -- cyclic shift left
end if;
  end process;
  DOUT <= TMP;
end Behavioral;
```


INV
INV
FDC
RST
CLK
D
Q
C
CLR
DOUT(7:0)

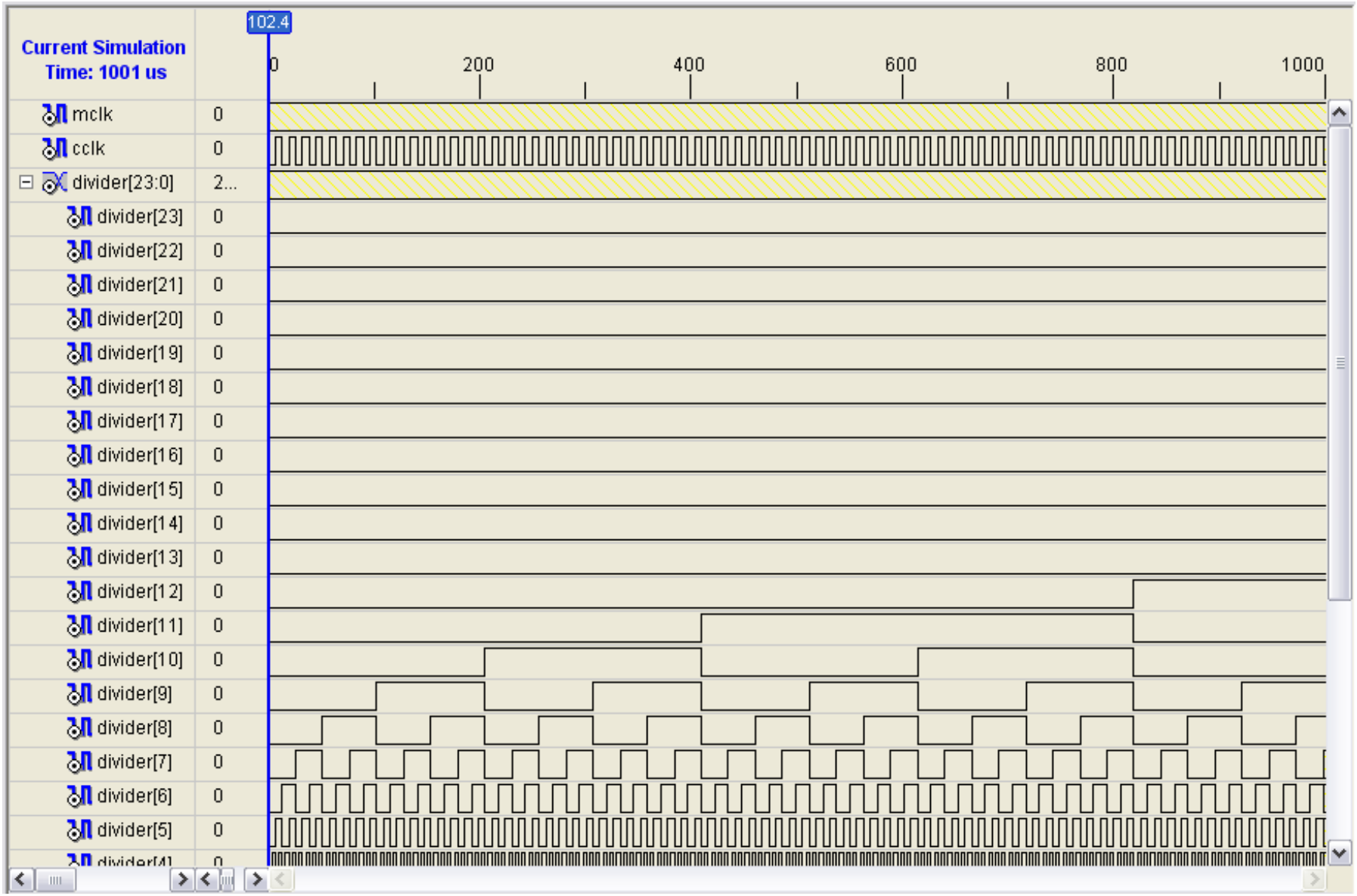
Делител на честота ($N=2^n$)

$f_{cclk} = f_{mclk} / N$

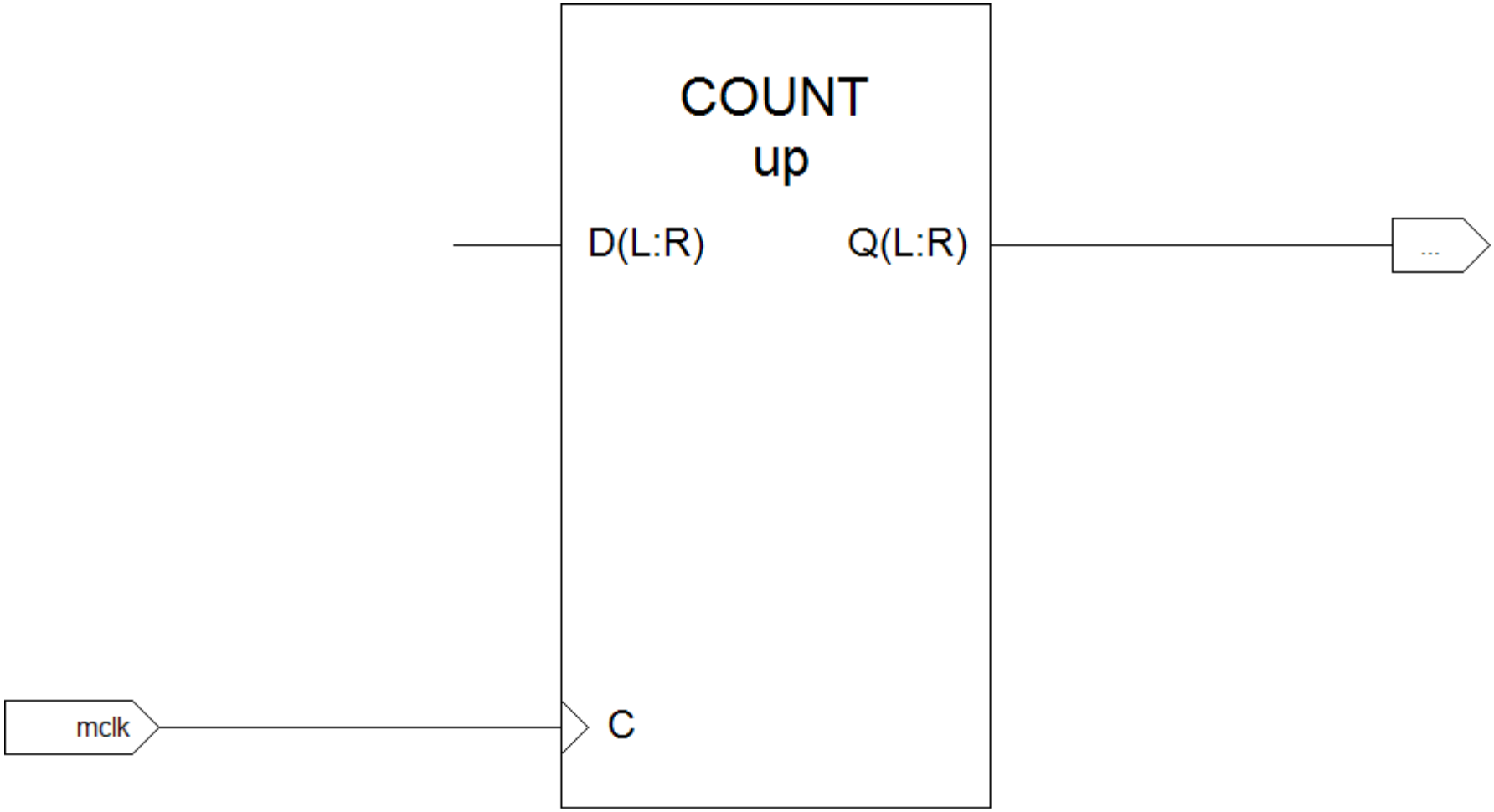
$N = 2^n$

Делител на честота (2^n)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clkdiv is
    Port (
        mclk    : in std_logic;
        cclk    : out std_logic);
end clkdiv;

architecture Behavioral of clkdiv is
signal divider  : std_logic_vector(26 downto 0);
begin

process (mclk)
     begin
          if rising_edge(mclk) then
               divider <= divider + 1;
          end if;
     end process;

cclk <= divider(25);

end Behavioral;
```



Забележка: за симулацията N= 2^5

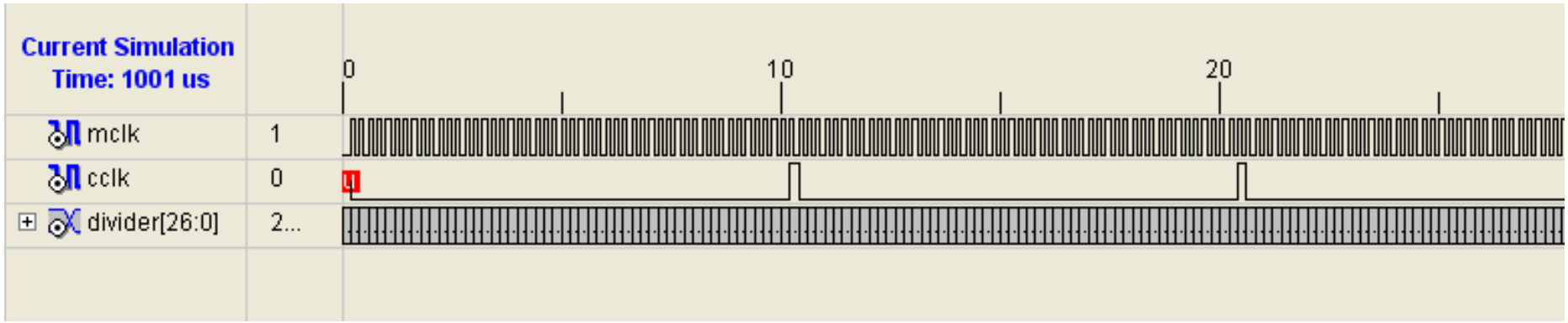

COUNT
up
D(L:R)
Q(L:R)
...
mclk
C

Делител на честота (N)

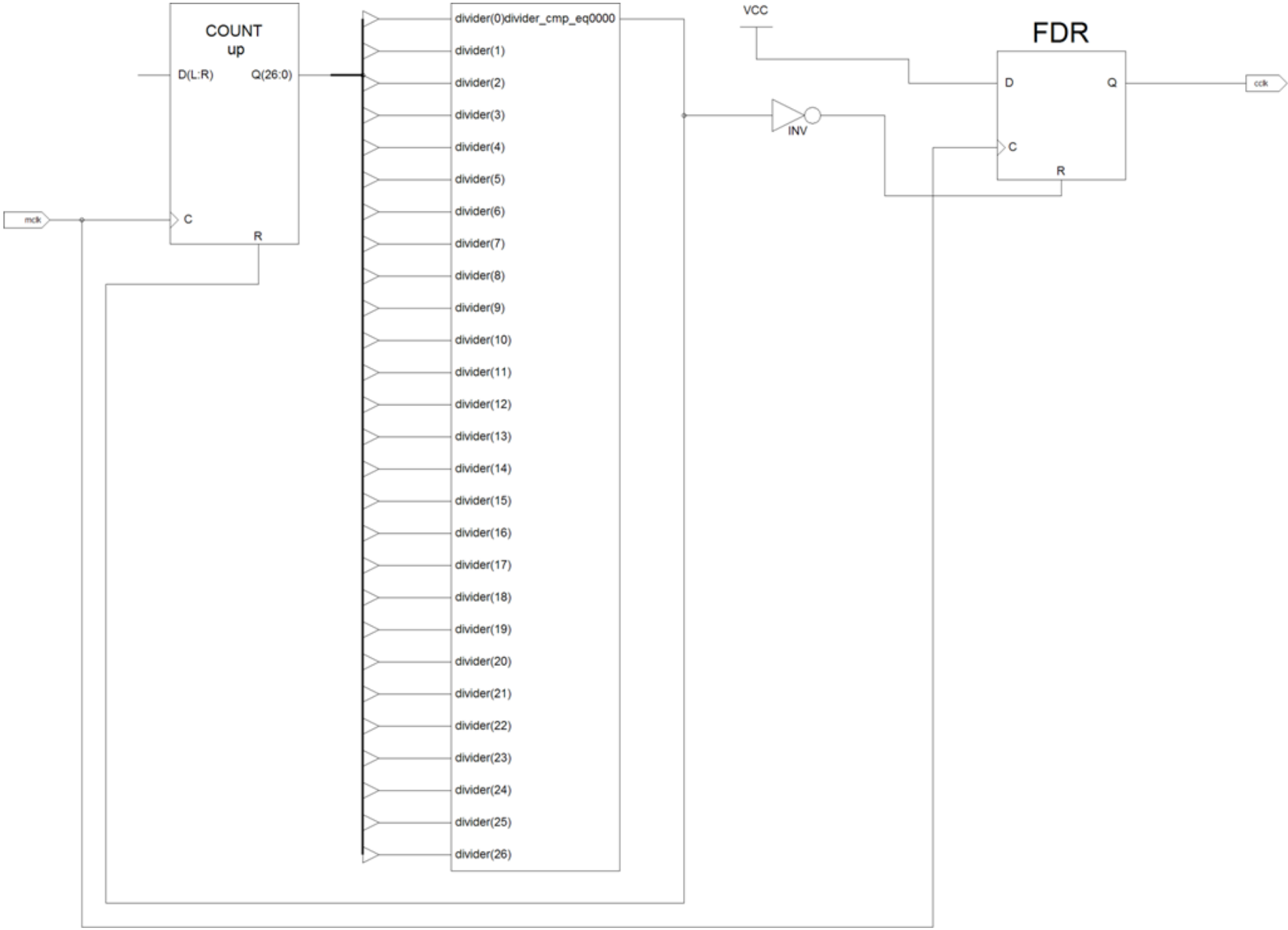
$$f_{cclk} = f_{mclk} / N$$

Делител на честота

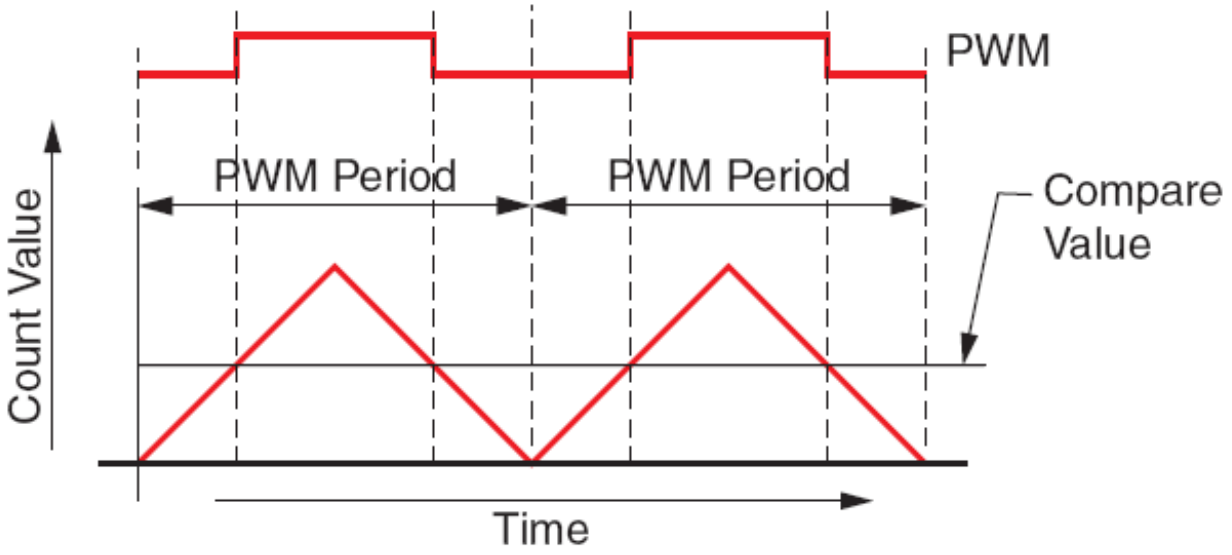
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clkdiv50m is
generic (N : integer := 50000000);
    Port (
        mclk    : in std_logic;
        cclk    : out std_logic);
end clkdiv50m;

architecture Behavioral of clkdiv50m is
signal divider  : std_logic_vector(26 downto 0)
begin
process (mclk)
      begin
            if rising_edge(mclk) then
                  if(divider=N)then
                        divider<=(others=>'0');
                        cclk<='1';
                  else
                        divider <= divider + 1;
                        cclk<='0';
                  end if;
            end if;
      end process;
end Behavioral;
```

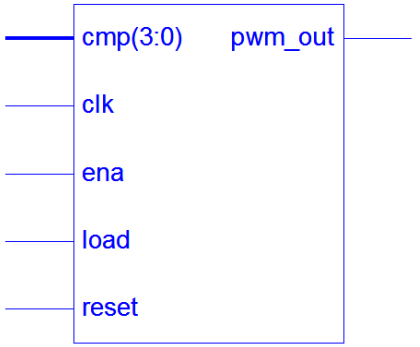


Забележка: за симулацията N=50


COUNT
up
D(L:R)
Q(26:0)
C
R
mclk
divider(0)divider_cmp_eq0000
divider(1)
divider(2)
divider(3)
divider(4)
divider(5)
divider(6)
divider(7)
divider(8)
divider(9)
divider(10)
divider(11)
divider(12)
divider(13)
divider(14)
divider(15)
divider(16)
divider(17)
divider(18)
divider(19)
divider(20)
divider(21)
divider(22)
divider(23)
divider(24)
divider(25)
divider(26)
VCC
INV
FDR
D
Q
C
R
cclk

PWM



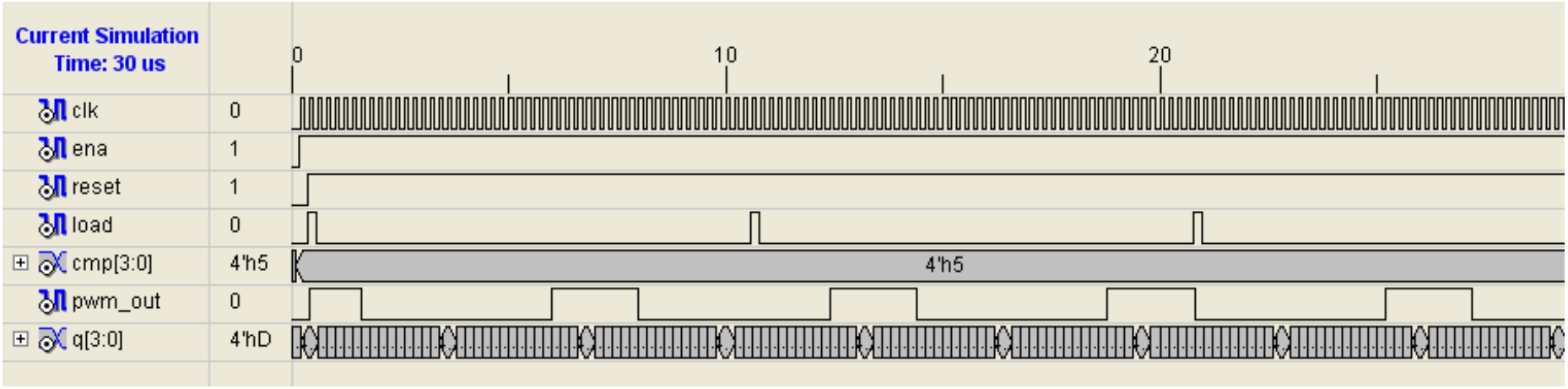


Ref: Xilinx XAPP448

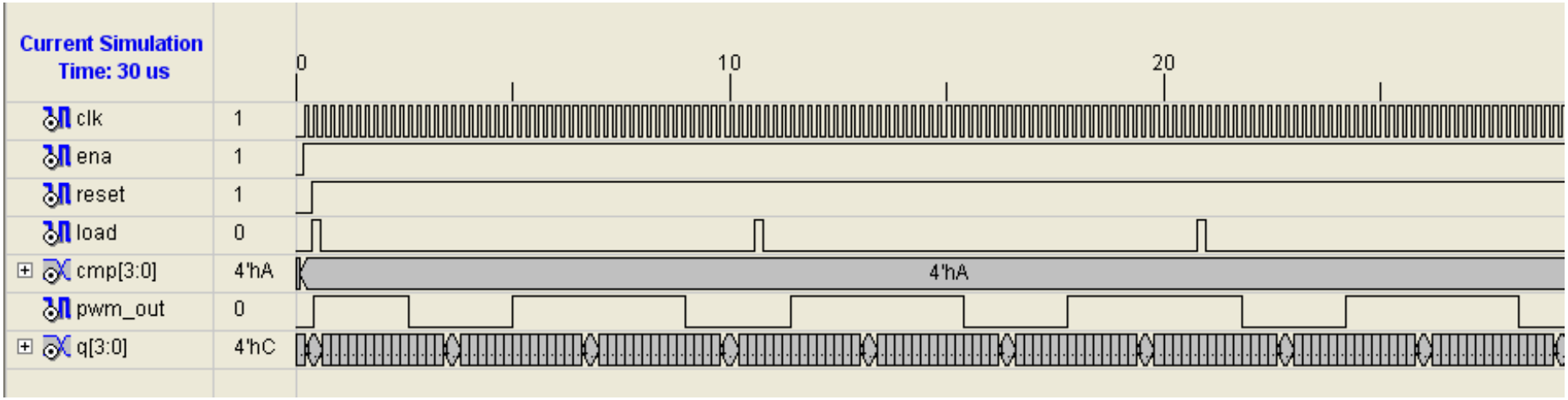
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity pwm is
Generic ( WIDTH : integer := 4 );
    Port (
    clk, ena, reset, load   : in std_logic;
    cmp : in std_logic_vector(WIDTH-1 downto 0);
    pwm_out : out std_logic);
end pwm;

architecture Behavioral of pwm is
constant UP_TC: std_logic_vector(WIDTH-1 downto 0 ) := (others => '1');
signal Q    : std_logic_vector(WIDTH-1 downto 0);
signal compare_value : std_logic_vector(WIDTH-1 downto 0);
signal up : std_logic;
begin
...
```

```
counter: process(clk) begin
    if rising_edge(clk) then
        if ( reset = '0' ) then
            Q <= (others => '0' );
            up <= '0';
        else
            if( ena = '1' ) then
                if ( up = '1' ) then
                    if( Q = UP_TC ) then
                        up <= '0';
                    else
                        Q <= Q + 1;
                    end if;
                else
                    if( Q = 0 ) then
                        up <= '1';
                    else
                        Q <= Q - 1;
                    end if;
                end if;
            end if;
        end if;
    end if;
end process;
```

```
comparator: process( clk ) begin
    if( reset = '0') then
        pwm_out <= '0';
        compare_value <= cmp;
    elsif rising_edge( clk ) then
        if( load = '1' ) then
            compare_value <= cmp;
        end if;
        if( Q < compare_value ) then
            pwm_out <= '1';
        else
            pwm_out <= '0';
        end if;
    end if;
end process;

end Behavioral;
```



cmp = 5, width = 4



cmp = A, width = 4

Масиви

```
type type_name is array (specification) of data_type;
signal signal_name: type_name [:= initial_value];
```

Пример

```
constant ADDR_WIDTH : integer := <num_addr_bits>;
constant DATA_WIDTH : integer := <data_width>;

type MY_RAM is array (2**ADDR_WIDTH-1 downto 0) of std_logic_vector
    (DATA_WIDTH-1 downto 0);
signal ram : MY_RAM;
```

RAM

```
entity a1 is
   Port ( address : in  STD_LOGIC_VECTOR (3 downto 0);
          data : inout  STD_LOGIC_VECTOR (7 downto 0);
          rw : in  STD_LOGIC;
          clock : in  STD_LOGIC);
end a1;
architecture Behavioral of a1 is
  type MY_RAM is array (15 downto 0) of std_logic_vector (7 downto 0);
  signal ram : MY_RAM;
begin
ram1 : process (clock) begin
  if rising_edge(clock) then
    if rw = '1' then
      data <= ram(conv_integer(address));
    else
      ram(conv_integer(address)) <= data;
    end if;
  end if;
end process;
end Behavioral;
```

```
type <rom_type> is array ((2**<ram_addr_bits>-1) downto 0) of
  std_logic_vector (<ram_width>-1) downto 0);
constant <ROM_NAME> : <rom_type>
  :=("0001","0010","0011","0100","0101",
                          "0110","0111","1000","1001","1010",
                          "1011","1100","1101","1110","1111",
                          "0001","0010","0011","0100","0101",
                          "0110","0111","1000","1001","1010",
                          "1011","1100","1101","1110","1111",
                          "1111","1111","<more_values_here>");

signal <rom_data> : std_logic_vector (<ram_width>-1) downto 0);
```