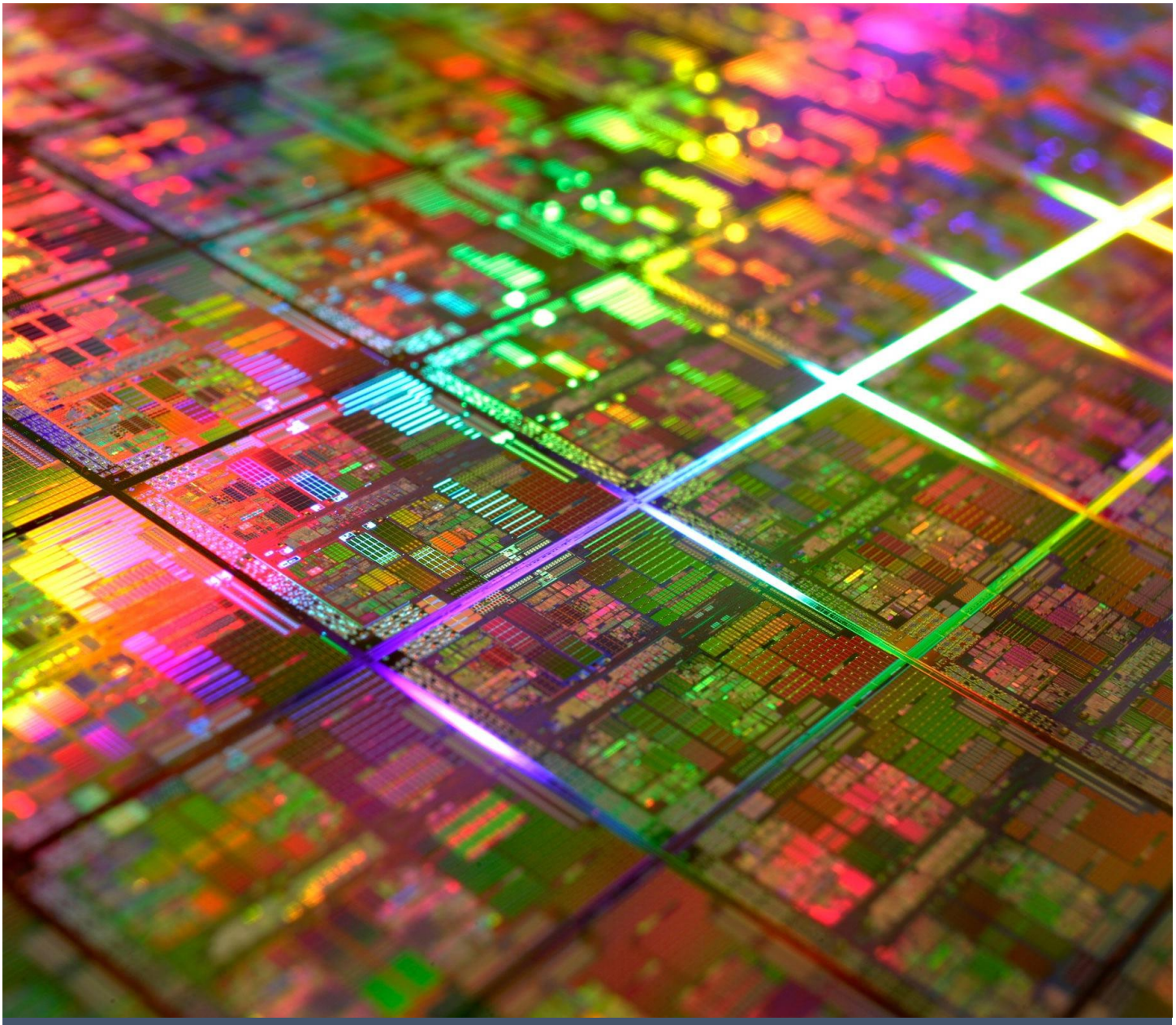


MH

Masoud Heidary

think outside the box

CA Lab – EX5

MasoudHeidaryMH@gmail.com

برای ساخت رجیستر 8 بیتی به این صورت عمل میکنیم که یک variable 8 bit در داخل process تعریف کرده، سپس با توجه به کلاک بالارونده و وضعیت پایه ها در آن زمان، خروجی را تعیین میکنیم.

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

entity Register8Bit is
    Port ( Din : in  STD_LOGIC_VECTOR (7 downto 0);
           Load : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Clk : in  STD_LOGIC;
           Dout : out  STD_LOGIC_VECTOR (7 downto 0));
end Register8Bit;

architecture Behavioral of Register8Bit is
-- signal reg: STD_LOGIC_VECTOR (7 downto 0) := "00000000";
begin

process (Clk)
    variable reg: STD_LOGIC_VECTOR (7 downto 0) := "00000000";
begin
    if (Clk'Event and Clk = '1') then
        if Reset = '1' then
            reg := "00000000";
        elsif Load = '1' then
            reg := Din;
        end if;
    end if;

    Dout <= reg;
end process;


end Behavioral;
```

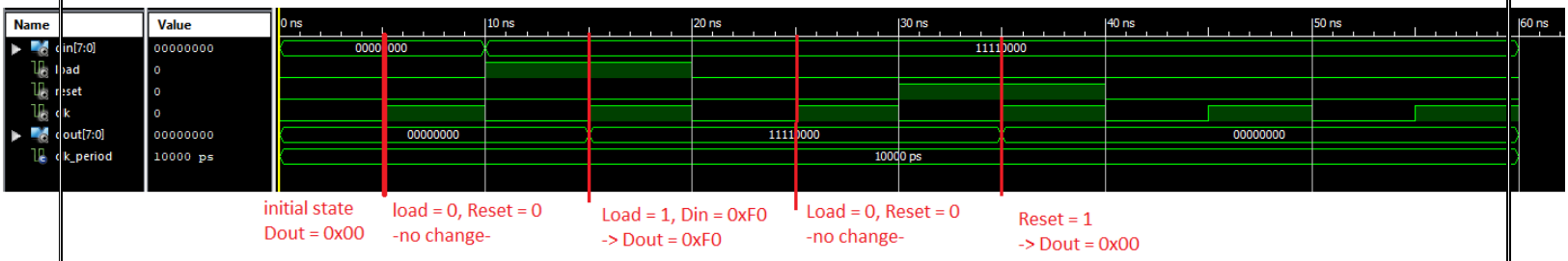
test bench:

تست این ماژول به راحتی طراحی آن است و به صورت زیر عمل میکنیم

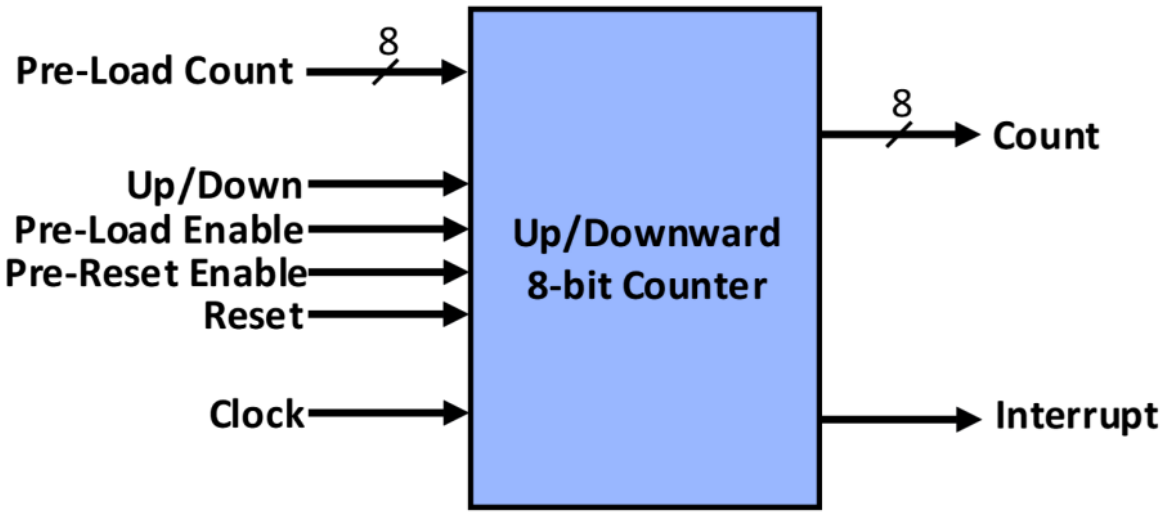
❋ قابل توجه است به کدهای auto generate شده دست نمیزنیم ❋

```
stim_proc: process
   begin
      Load <= '0';
      Reset <= '0';
      wait for Clk_period;

      Din <= "11110000";
      Load <= '1';
      wait for Clk_period;

      Load <= '0';
      wait for Clk_period;

      Reset <= '1';
      wait for Clk_period;

      Reset <= '0';
      wait for Clk_period;

      wait;
   end process;
```

output:



طراحی counter



ایده ها:

برای شمارش، ابتدا یک متغیر تعریف میکنیم

اگر در حالت بالارونده باشیم، با آمدن هر کلاک، به آن یک واحد اضافه میکنیم

اگر PreResetEn = 1 باشد، با رسیدن متغیر به عدد ورودی، متغیر را 0 میکنیم

اگر PreResetEn = 0 باشد، هر زمانی که متغیر بزرگتر از 255 شد، باید آن را ریست کنیم، که خود دو حالت دارد

اگر PreLoadEn = 0 باشد، متغیر را 0 خواهیم کرد، در غیر این صورت برابر عدد ورودی قرار خواهیم داد.

برای قسمت پایین رونده نیز ایده به صورتی ثابت است،

با آمدن هر کلاک، متغیر را یک واحد کوچکتر میکنیم،

اگر PreResetEn = 1 باشد، با رسیدن متغیر به عدد ورودی، متغیر را به 255 ریست میکنیم

اگر PreResetEn = 0 باشد، هر زمانی که متغیر کوچکتر از 0 شد باید آن را ریست کنیم، که خود دو حالت دارد

اگر PreLoadEn = 0، متغیر را 255 خواهیم کرد، در غیر این صورت برابر عدد ورودی قرار خواهیم داد

برای قسمت های Reset, Interrupt، ایده ها ساده هستند، و از توضیح آنها صرف نظر میکنم.

Code

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

entity Counter is
    Port ( PreInput : in  STD_LOGIC_VECTOR (7 downto 0);
           Count : out  STD_LOGIC_VECTOR (7 downto 0);
           UpDown : in  STD_LOGIC;
           PreLoadEn : in  STD_LOGIC;
           PreResetEn : in  STD_LOGIC;
           Reset: in STD_LOGIC;
           Clk : in  STD_LOGIC;
           Interrupt : out  STD_LOGIC);
end Counter;

architecture Behavioral of Counter is

begin

process (Clk, Reset)
    variable reg: integer := 0;
begin

    if Reset = '1' then
        if PreLoadEn = '1' then
            reg := to_integer(unsigned(PreInput));
        elsif UpDown = '0' then
            reg := 0;
        else
            reg := 255;
        end if;
    end if;

    -- instead of ___ if (Clk'Event and Clk = '1') then
    if rising_edge(Clk) and Reset = '0' then
        Interrupt <= '0';

        -- up routine
        if UpDown = '0' then
```

```
            reg := reg + 1;

            -- natural reset
            if reg > 255 then
                if PreLoadEn = '1' then
                    reg := to_integer(unsigned(PreInput));
                else
                    reg := 0;
                end if ;

                Interrupt <= '1';

            -- pre reset en
            elsif PreResetEn = '1' then
                if reg = to_integer(unsigned(PreInput)) then
                    reg := 0;
                    Interrupt <= '1';
                end if ;
            end if ;

        -- down routine
        else
            reg := reg - 1;

            -- natural reset
            if reg < 0 then
                if PreLoadEn = '1' then
                    reg := to_integer(unsigned(PreInput));
                else
                    reg := 255;
                end if ;

                Interrupt <= '1';

            -- pre reset en
            elsif PreResetEn = '1' then
                if reg = to_integer(unsigned(PreInput)) then
                    reg := 255;
                    Interrupt <= '1';
                end if ;
            end if ;

        end if;

    end if ;
```

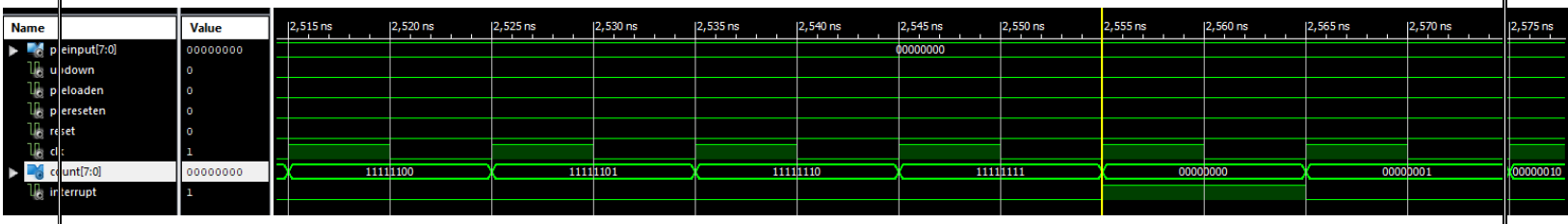
```
    -- set output
    Count <= std_logic_vector(to_unsigned(reg, 8));
end process;

end Behavioral;
```

Test Bench:

ابتدا ماژول را در حالت های مختلف برای بالارونده چک میکنیم

Normal:

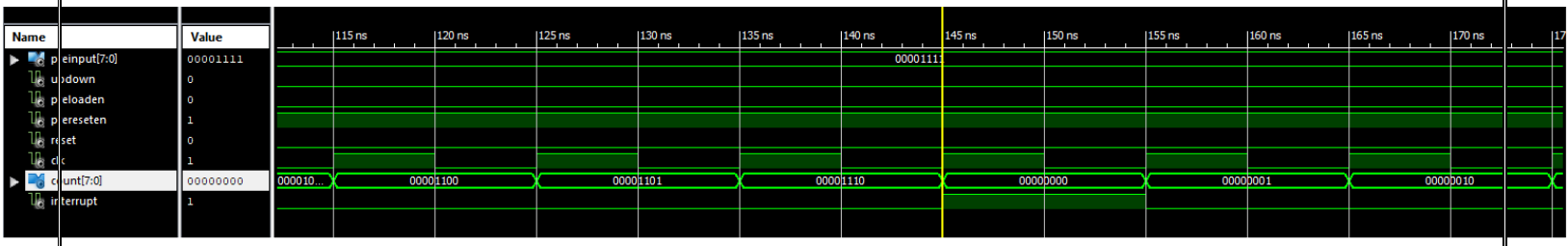
```
        UpDown <= '0';
```



Reset to 0x00
Interrupt = 1 for one clock

PreResetEn:

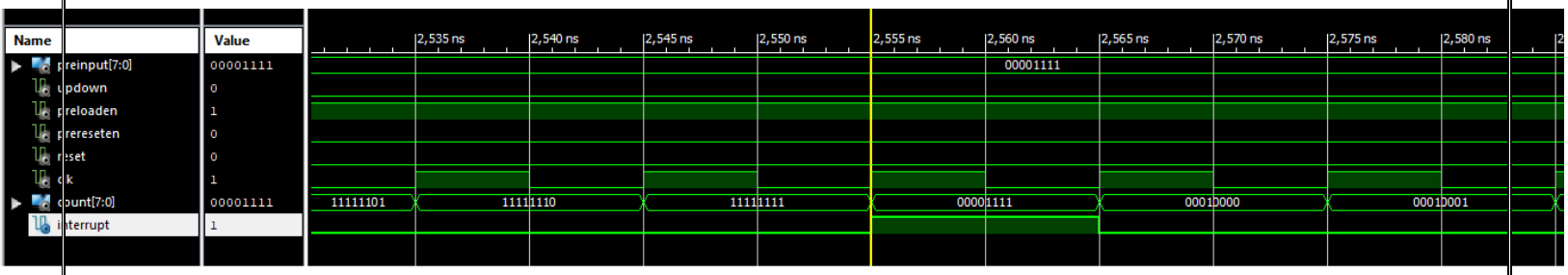
```
        UpDown <= '0';
       PreInput <= "00001111";
       PreResetEn <= '1';
```



PreInput = 0x0F, PreResetEn = 1
counter reset on 0x0F, Interrupt = 1 for one clock

PreLoadEn:

```
UpDown <= '0';
PreInput <= "00001111";
PreLoadEn <= '1';
```

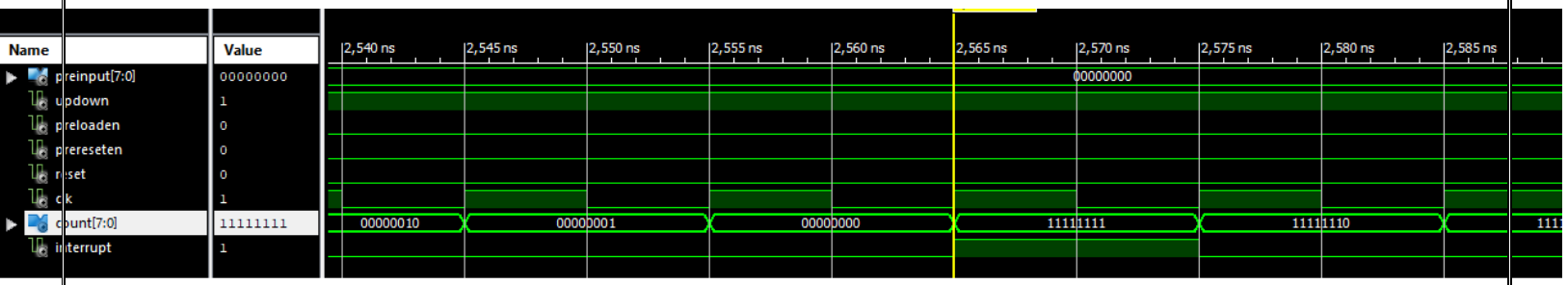


PreInput = 0x0F, PreLoadEn = 1
Reset and start from 0x0F, interrupt = 1 for one clock

حال ماژول را به صورت پایین رونده نیز تست میکنیم

Normal:

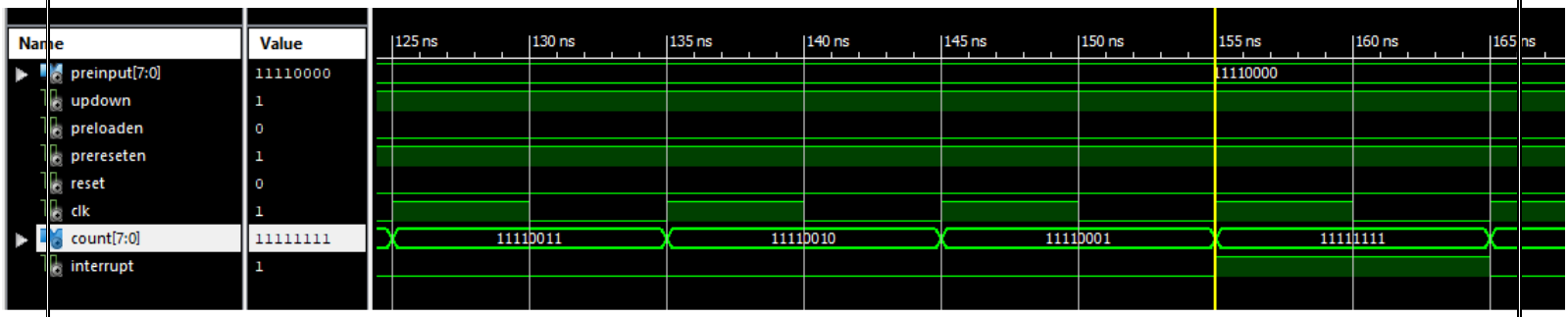
```
UpDown <= '1';
```



reset to 0xFF
interrupt for one clock

PreResetEn:

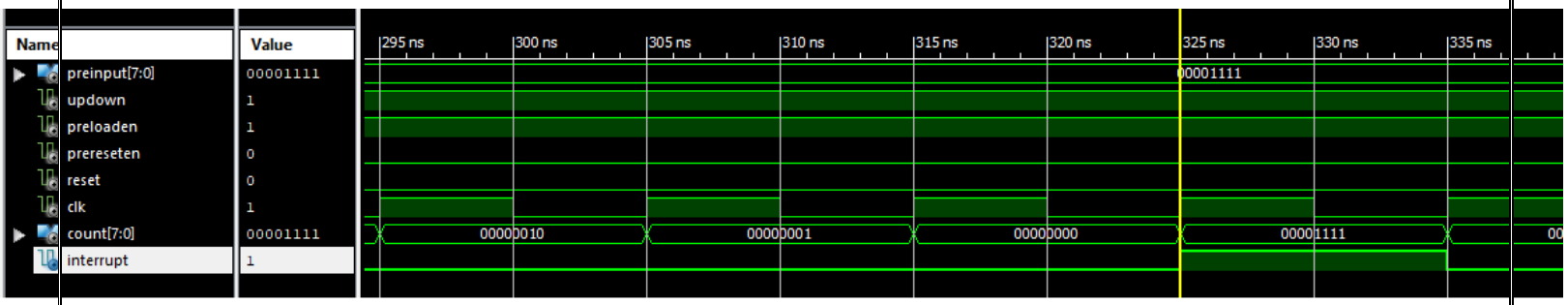
```
UpDown <= '1';
PreInput <= "11110000";
PreResetEn <= '1';
```



UpDown = 1, PreResetEn = 1, PreInput = 0xF0
reset to 0xFF on reaching to 0xF0

PreLoadEn:

```
UpDown <= '1';
PreInput <= "00001111";
PreLoadEn <= '1';
```

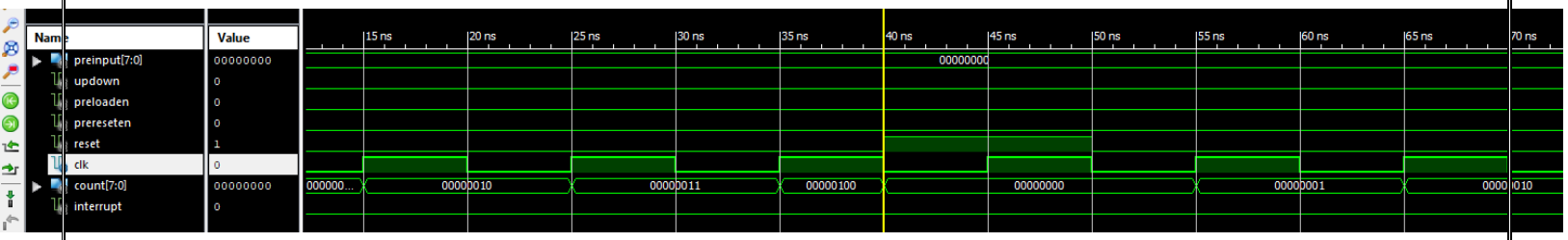


UpDown = 1, PreLoadEn = 1, PreInput = 0x0F
Load on 0x0F on reset

حال که تمامی حالت ها برای قسمت های بالارونده و پایین رونده تست شدند، باید پایه Reset را نیز تست کنیم، دقت کنید که پایه Reset به صورت aSync طراحی شده است.

Upward, normal

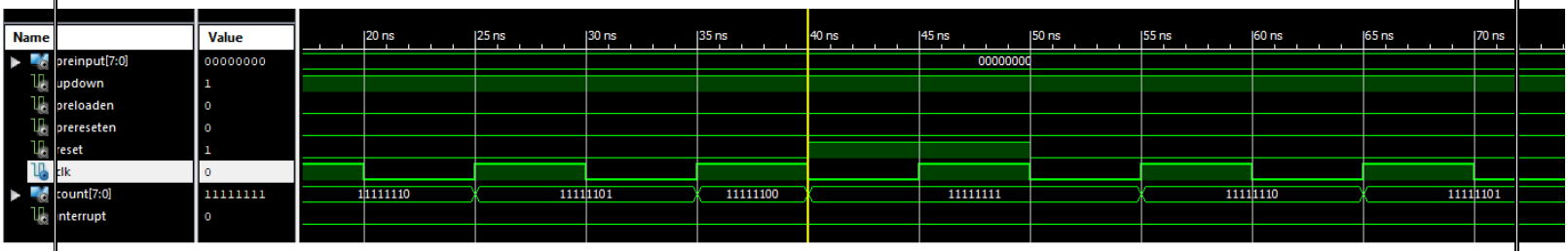
```
    UpDown <= '0';
    wait for Clk_period;
    wait for Clk_period;
    wait for Clk_period;
    wait for Clk_period;

    Reset <= '1';
    wait for Clk_period;

    Reset <= '0';
    wait for Clk_period;
```



Reset = 1 , UpDown = 0, PreLoadEn = 0
reset output to

Downward, normal

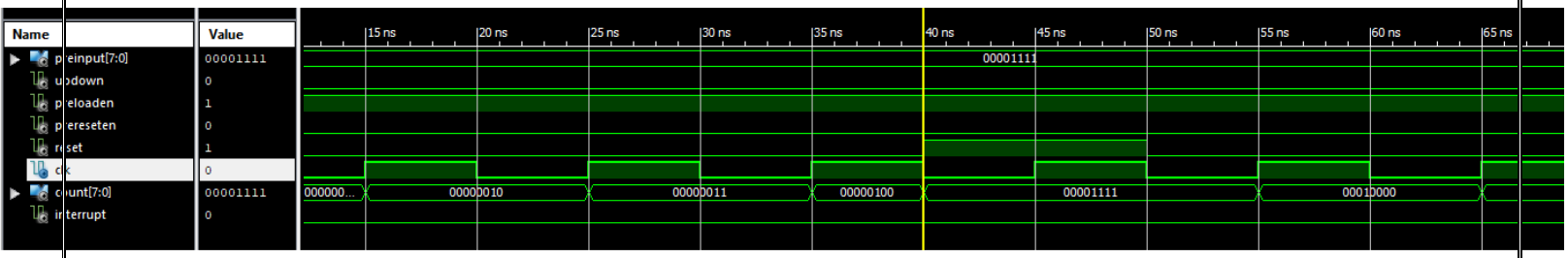
```
    UpDown <= '1';
    wait for Clk_period;
    wait for Clk_period;
    wait for Clk_period;
    wait for Clk_period;

    Reset <= '1';
    wait for Clk_period;

    Reset <= '0';
    wait for Clk_period;
```



Reset = 1, UpDown = 1, PreLoadEn = 0
Reset output to maximum number (0xFF)

PreLoadEn (upward OR downward)

```
UpDown <= '0';
PreInput <= "00001111";
PreLoadEn <= '1';
wait for Clk_period;
wait for Clk_period;
wait for Clk_period;
wait for Clk_period;

Reset <= '1';
wait for Clk_period;

Reset <= '0';
wait for Clk_period;
```



Reset = 1, PreLoad = 0x0F, PreLoadEn = 1
Reset output to 0x0F