درس طراحی کامپیوتری سیستم‌های دیجیتال

گزارش شماره‌ی: 18

اعضای گروه: نرگس قاسمی ، تارا برقیان

چکیده:

در این تمرین، هدف پیاده سازی یک شمارنده 4کاره بود، که بتواند به بالا،پایین بشمارد. همچین گام شمارش و نقطه شروع قابل تنظیم باشد.

با استفاده از Generic و مقدار پیش فرض 8، این شمارنده را مدل سازی کردیم.

ورودی های این شمارنده به ترتیب شامل کلاک، ریسِت سنکرون، نقطه آغاز،گام شمارش و استیت مد نظر کاربر است و خروجی در هر اجرای کلاک، شماره بعدی خواهد بود.

با کمک 0و1 کردن دو سیگنال load_in و load_step میتوانیم نقطه شروع و گام شمارش را تنظیم کنیم.

فهرست مطالب

1) بخش ENTITY

```
ENTITY counter IS
    Generic ( n : integer := 8 );
    PORT (
        clock       : IN  std_logic;
        nreset      : IN  std_logic;
        load_in     : IN std_logic;
        load_step   : IN std_logic;
        mode        : IN  std_logic;
        output      : OUT std_logic_vector(n-1 DOWNTO 0);
        data_in     : IN std_logic_vector(n-1 DOWNTO 0);
        data_step   : IN std_logic_vector(n-1 DOWNTO 0)
    );
END counter;
```

در این بخش ورودی و خروجی های شمارنده مشخص شده اند. با کمک Generic میتوانیم، تعداد بیت های شمارنده را تغییر دهیم برای مثال 8، 16 و ....

یک ریسِت low active داریم، میدانیم بهتر است در طراحی ها از این reset استفاده کنیم تا بتوان گیت ها را به nand تبدیل کرد.

دو سیگنال کنترلی load_in و load_step از این جهت قرار داده شده تا بدانیم کاربر میخواهد نقطه شروع و گام را خودش تعیین کند یا خیر. (در ادامه مفصل توضیح داده خواهد شد.)

سیگنال کنترلی mode برای مشخص کردن شمارش بالا یا پایین است که در ادامه میبینیم با 0 شدن، گام 1+ است و در غیر این صورت 1- .

بقیه ی ورودی ها و خروجی مدار، ارایه هایی little-endian هستند که طول n دارند.

2) بخش ARCHITECTURE

در پیاده سازی معماری شمارنده، به 2 سیگنال کمکی احتیاج داشتیم، تا برای نگه داری مقادیر وارد شده توسط کاربر از انها استفاده کنیم. (در ادامه کاربرد انهارا خواهیم دید.)

```
SIGNAL temp: std_logic_vector (n-1 DOWNTO 0) ; -- temporary vector to hold user input
SIGNAL temp_step: std_logic_vector (n-1 DOWNTO 0):=(OTHERS => '0'); -- temporary vector to hold user step input
```

در ابتدا چک میشود که کاربر مدار را ریسِت کرده یا خیر. اگر مقدار سیگنال ریسِت 0 بود، سیگنال کمکی temp تماما با 0 مقدار دهی خواهد شد.

در غیر این صورت،در هر لبه ی بالا رونده کلاک،اگر کاربر بخواهد ورودی وارد کند، این کار را با"1" کردن load_step نشان میدهد.در غیر این صورت، شمارنده با 01...00 ، مقدار دهی میشود.

```
IF (nreset = '0') THEN
            temp <= (OTHERS => '0' );
ELSIF (clock'EVENT AND clock='1' ) THEN
    IF (load_step = '1') THEN
                temp_step <= data_step; -- load step parallel data
    ELSE
        temp_step <= (0=> '1' , OTHERS => '0');
    END IF;
```

مقدار شروع هم اگر کاربر بخواهد با "1" کردن load_in تنظیم میکند. در این حالت متغیر temp، مقدار ورودی را دریافت میکند. در غیر این صورت مقدار پیشین را حفظ میکند.

```
    --temp
    IF (load_in = '1') THEN
        temp <= data_in; -- load input parallel data
```

در پایان با کمک سیگنال کمکی mode، بالارونده یا پایین رونده بودن تعیین میشود.

در هر مرحله، متغیر temp با Step جمع (کم) شده و مجدد داخل temp نگه داری میشود.

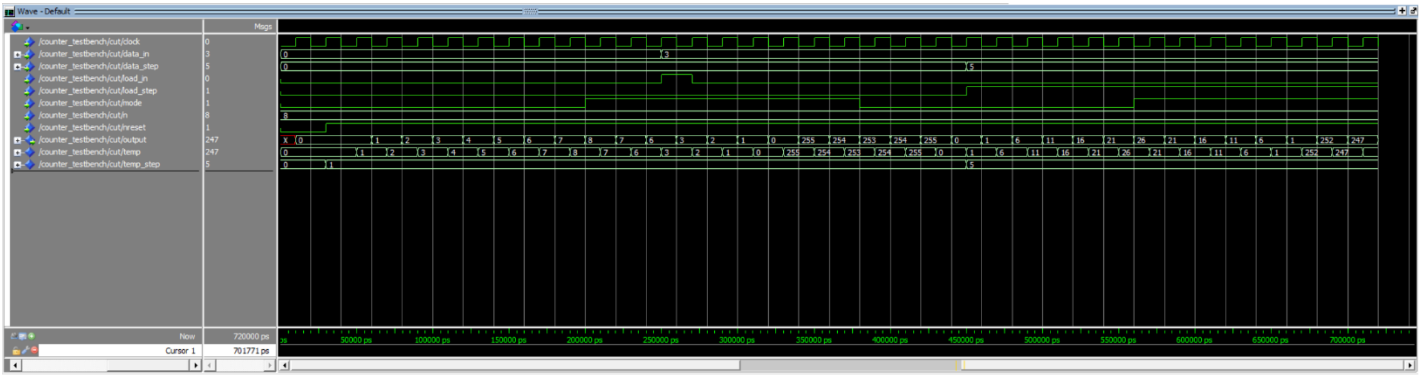
```
ELSIF(mode = '0' ) THEN
    temp <= STD_LOGIC_VECTOR (unsigned(temp) + unsigned(temp_step));
ELSE
    temp <= STD_LOGIC_VECTOR (unsigned(temp) - unsigned(temp_step));
END IF;
```

در آخر output، مقدار temp را برمیگرداند.

```
output <= temp; -- change output to current temp signal value
```

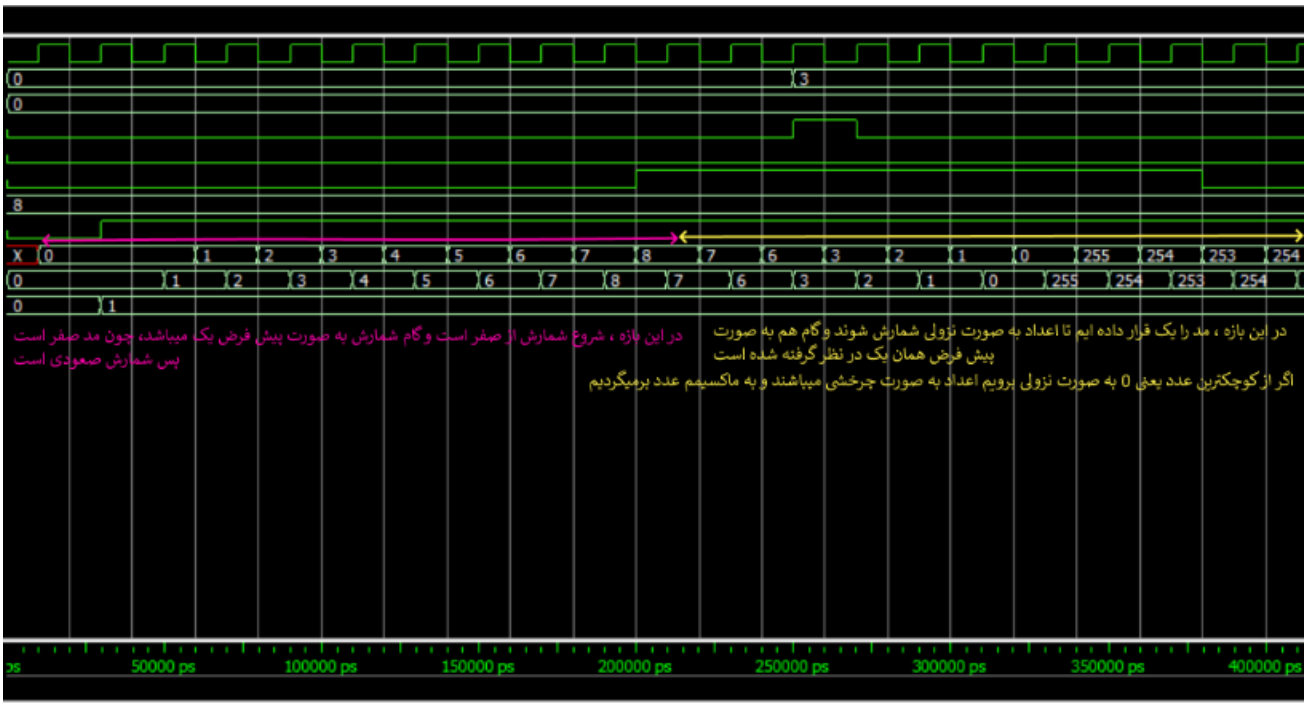
3) بخش testbench

برای درک بهتر این بخش، بهتر است به شکل خروجی رجوع کنیم.

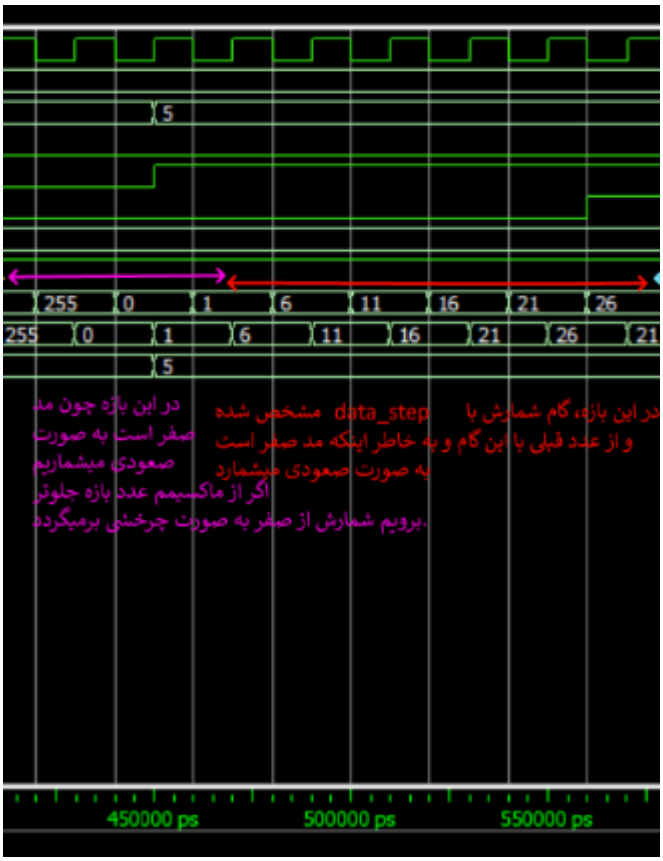


تصویر 1/4

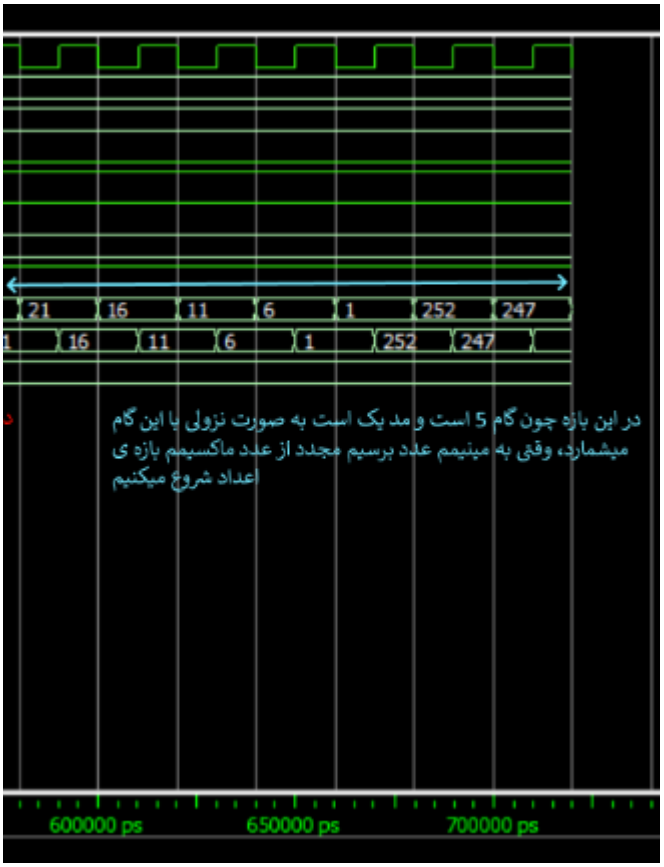
تصویر 2/4



تصویر 3/4



تصویر 4/4



4) جمع بندی و نتیجه گیری

با توجه به جست وجوهایی که در اینترنت داشتیم متوجه شدیم، این مدار را به شیوه های گوناگون میتوان پیاده سازی کرد. که ما این روش را انتخاب کرده و خودمان پیاده سازی کردیم.

همچنین متوجه شدیم در بیشتر مدار ها reset را activelow میگیرند تا هنگام پیاده سازی راحت تر بتوان از گیت nand استفاده کرد.

همچنین حتما باید توجه کرد هنگا پیاده سازی همه ی حالات در دسته بندی ها در نظر گرفته شود. یکی از بهترین راه ها استفاده از ELSE یا DEFUALT است.

5) پیوست

- https://startingelectronics.org/software/VHDL-CPLD-course/tut19-up-down-counter/