ازمایش هفتم

موضوع: دریافت سریال از Rx و تغییر نور led با توجه به عدد دریافت شده

تاریخ آزمایش: ۱۴۰۲/۹/۸

استاد: مهندس جوادی

جواد فرجی (۹۹۵۲۲۰۰۵)

محمد رحمانی (۹۷۵۲۱۲۸۸)

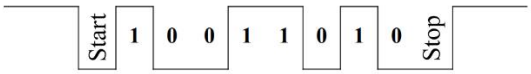
ورودی و خروجی:

کلاک که همیشه هست! ورودی سریال از پورت rx خوانده میشود و بر روی ۸ led به عنوان خروجی نمایش داده میشود.

```
entity a6 is
    port( GCLK : in std_logic;
          RX : in std_logic;
          LED : out std_logic_vector (7 downto 0));
end a6;
```

پروسس ها:

- **ساخت کلاک خروجی برای تشخیص توالی بیت ها:**



Bauds	Bit duration	Speed
9600 bauds	**104.167 µs**	**1200 bytes/s**
19200 bauds	52.083 µs	2400 bytes/s
28800 bauds	34.722 µs	3600 bytes/s
38400 bauds	26.042 µs	4800 bytes/s
57600 bauds	17.361 µs	7200 bytes/s
76800 bauds	13.021 µs	9600 bytes/s
115200 bauds	**8.681 µs**	**14400 bytes/s**

برای این که بتوانیم به ترتیب بیت ها را بخوانیم و تشخیص دهیم نیاز داریم که یک سری استاندارد ها را رعایت کنیم. یکی از این استاندارد ها اندازه کلاک است که در جلسه توضیح داده شده و باید کلاک خروجی جدیدی بسازیم. برای این کار نیاز به یک پروسس داریم که کلاک جدیدی بر اساس کلاک کنونی بسازد. در اینجا هر ۱۷۵ بار که کلاک **GCLK** از ۰ به ۱ تریگر میشود، یک بار مقدار کلاک **CLCK** تغییر میکند. که یک سیگنال در بدنه اصلی برنامه است. البته باید یک بار هم به مقدار نصف ۱۷۵ بار در اولین دریافت، کلاک را تغییر دهیم. این کار برای این است که ورودی به درستی دریافت شود.

```
process (GCLK)
    variable counter : integer range 0 to 200 := 0;
    variable first_time : std_logic := '0';

    begin
        if (falling_edge(GCLK)) then
            if started = '1' then
                if first_time = '1' then
                    if counter < 87 then
                        counter := counter + 1;
                    else
                        counter := 0;
                        first_time := '0';
                        CLCK <= not CLCK;
                    end if;
                elsif counter < 175 then
                    counter := counter + 1;
                else
                    counter := 0;
                    CLCK <= not CLCK;
                end if;
            else
                first_time := '1';
            end if;
        end if;
end process;
```

- **انتقال سریال بیت ها در یک پروسس**

در ابتدا باید بررسی کنیم که انتقال شروع شده است یا خیر. این شروع شدن با یک متغیر **started** انجام میشود. این متغیر وقتی که اولین بار به مقدار صفر در ورودی برسیم به حالت **started** در میاید.

بعد از آن ۹ بیت را میخوانیم و به ترتیب بر روی یک متغیر ذخیره میکنیم تا عدد دریافت شده توسط ورودی سریال را ذخیره داشته باشیم.

این کارها در یک **case when** انجام می شوند که مانند تصویر زیر است.

```
51    process(CLCK,started,RX)
52        variable counter: integer range 0 to 10 :=0 ;
53        variable density: std_logic_vector(7 downto 0);
54        variable density_int: integer range 0 to 255 :=0 ;
55        begin
56        if (RX = '0' and started = '0') then
57            started <= '1';
58        elsif (falling_edge(CLCK)) then
59            case counter is
60                    when 0 =>
61                        counter := counter + 1;
62                        density(0) := RX;
63                    when 1 =>···
66                    when 2 =>···
69                    when 3 =>···
72                    when 4 =>···
75                    when 5 =>···
78                    when 6 =>···
81                    when 7 =>···
84                    when 8 =>
85                        counter := counter + 1;
86                        started <= '0';
87                    when others =>
88                        counter := 0;
89                        started <= '0';
90                end case;
91        end if;
92        density1 <= density;
93    end process;
```

- **تغییر نور ال ای دی ها با توجه به عدد دریافت شده توسط RX**

برای این کار نیاز است نسبت به عدد دریافت شده نور ال ای دی ها را تغییر دهیم. فرض میکنیم که ۲۵۵ بزرگترین عدد قابل دریافت باشد و ورودی ۱۰۰ را داریم.

در این حالت باید از هر ۲۵۵ بار که کلاک تریگر میشود، ۱۰۰ بار led را روشن کنیم و بعد از آن خاموش کنیم. این کار با ۲ شرط انجام میشود که عدد دریافتی را با یک شمارنده مقایسه میکند و در هر

کلاک یک واحد به شمارنده افزوده میشود. حال اگر شمارنده بزرگتر از عدد دریافتی شود، ال ای دی را خاموش میکنیم. اینطوری میتوانیم نور ال ای دی را با توجه به عدد دریافتی تنظیم کنیم.

```
process(GCLK)
variable duty: std_logic_vector(7 downto 0) := "00000000";
begin
    if (falling_edge(GCLK)) then
        if (density1 > duty) then
            duty := duty + '1';
            LED <= "11111111";
        elsif (duty < "1111111") then
            LED <= "00000000";
            duty := duty + '1';
        else
            duty := "00000000";
            LED <= "00000000";
        end if;
    end if;

end process;
```

مپ کردن خروجی ها به روی fpga:

برای مپ کردن روی برد های fpga، با استفاده از داکیومنت موجود، این خطوط را داخل فایل ucf قرار میدهیم:

```
NET "GCLK" CLOCK_DEDICATED_ROUTE = FALSE;
NET "GCLK" LOC = P184;
NET "RX" LOC = P37;

NET "LED[0]" LOC = P61;
NET "LED[1]" LOC = P62;
NET "LED[2]" LOC = P63;
NET "LED[3]" LOC = P64;
NET "LED[4]" LOC = P65;
NET "LED[5]" LOC = P67;
NET "LED[6]" LOC = P68;
NET "LED[7]" LOC = P71;
```

اجرای برنامه بر روی برد fpga:

1. Synthesize
2. Implement design
3. Generate programming

در این سه مرحله گزینه run را میزنیم و در صورتی که مشکل خاصی در برنامه وجود نداشته باشد و به باگ نخوریم به مرحله بعد می‌رویم.

4. Impact

با استفاده از این برنامه، فایل باینری ساخته شده را به programmer انتقال میدهیم و programmer این برنامه را روی بردهای fpga اجرا میکند.