ازمایش پنجم

موضوع: انتقال سریال جمع دو عدد از دیپ سوییچ به tx به صورت ascii

تاریخ آزمایش: ۱۴۰۲/۸/۹

استاد: مهندس جوادی

جواد فرجی (۹۹۵۲۲۰۰۵)

محمد رحمانی (۹۷۵۲۱۲۸۸)

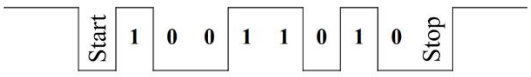
ورودی و خروجی:

ورودی و خروجی ها دقیقا مانند آزمایش قبلی هستند.

```
entity a4 is
    port( GCLK : in std_logic;
          TX : out std_logic;
          DIP : in std_logic_vector (7 downto 0));
end a4;
```

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.

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

    begin
        if (rising_edge(GCLK)) then
            if counter < 175 then
                counter := counter + 1;
            else
                counter := 0;
                CLCK <= not CLCK;
            end if;
        end if;
end process;
```

- **انتقال سریال بیت ها در یک پروسس**

برای این کار باید مطابق استاندارد های انتقال سریال، یک بار بیت خروجی را ۰ کنیم که نشان دهنده استارت است. بعد از آن باید ۸ بیت به خروجی بدهیم. خروجی ۹ ام میتواند استاپ باشد که در این صورت باید ۰ باشد و یا میتواند بیت **parity** باشد که در اینجا استفاده نکردیم.

اما در این آزمایش نیاز است که این کار را سه بار انجام دهیم. دو بار برای کد اسکی عدد های به دست آماده و عدد دیگر برای فرستادن مقدار **10** که به عنوان **new line** عمل کند.

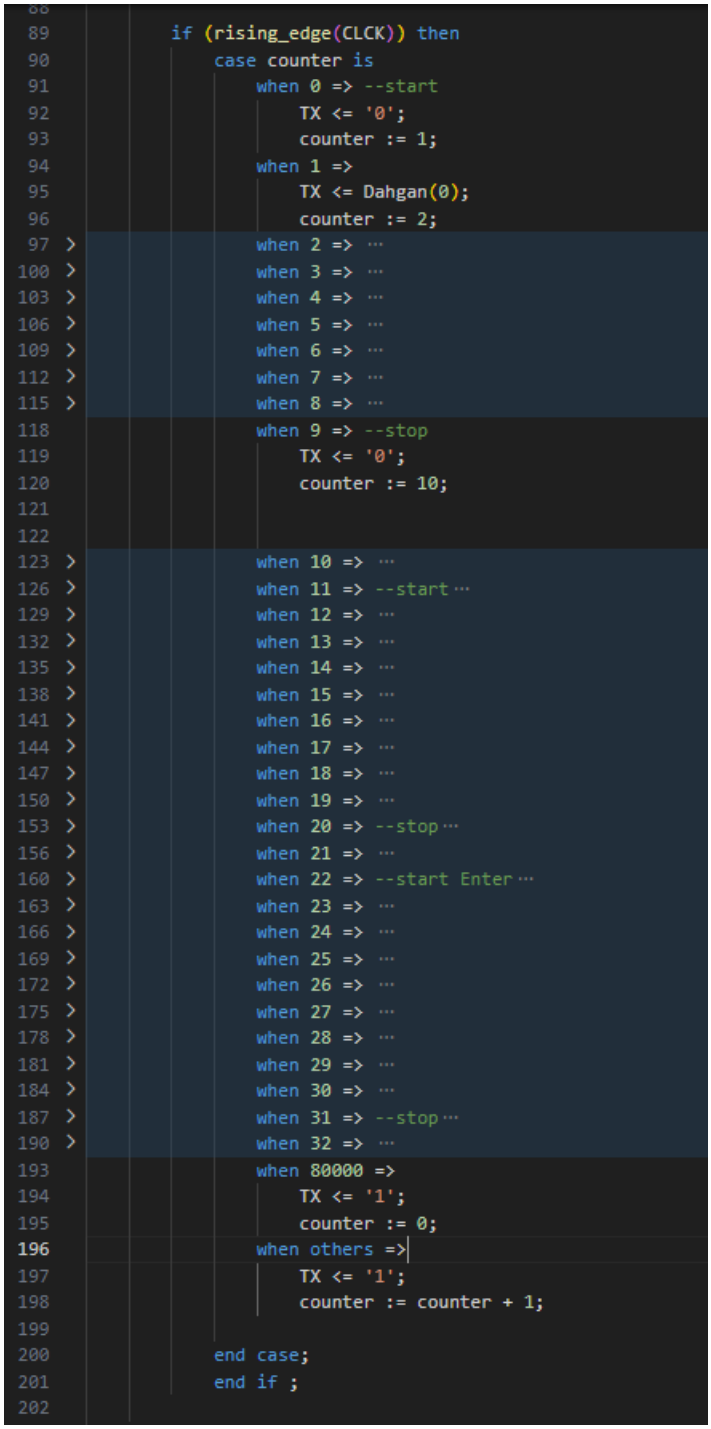
نکته: در حالت عادی بیت خروجی باید ۱ باشد. به غیر از زمان هایی که نیاز به انتقال سریال داشته باشیم. استارت و استاپ نشان دهنده اغاز و پایان انتقال اطلاعات هستند.

در این پروسس هم از این روش استفاده شده. با استفاده از سوییچ کیس بر روی متغیر **counter** چند حالت را بررسی میکنیم.

در حالت ۰ و ۹ استارت و استاپ داریم. و بعد از آن دوباره از ۱۰ تا ۲۱ و بعد از آن از ۲۲ تا ۳۳ میشماریم و عدد های مورد نیاز را به خروجی میدهیم.

در دو حالت اخر نیز شمارنده تا ۸۰۰۰۰ می شمارد که هر یک ثانیه یک بار اطلاعات منتقل شوند.

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



- **به دست آوردن ضرب و رقم یکان و دهگان اعداد خروجی:**

در اینجا یک بار ورودی را در متغیر میریزیم و بعد از تبدیل به `integer` و ضرب کردن، بر اساس مقدار عدد به دست آمده از ضرب دو عدد ورودی، خروجی یکان و دهگان را مشخص میکنیم. این کار به این خاطر انجام شده که نمیتوانستیم از `mod` و تقسیم استفاده کنیم.

a4.vhd

```
33      variable counter : integer range  0 to 1000000:= 0 ;
34      variable A : std_logic_vector(3 downto 0);
35      variable B : std_logic_vector(3 downto 0);
36      variable A_Integer :integer range 0 to 9;
37      variable B_Integer :integer range 0 to 9;
38      variable C1 :integer range 0 to 60;
39      variable C2 :integer range 0 to 60;
40      variable Yekan : std_logic_vector(7 downto 0);
41      variable Dahgan : std_logic_vector(7 downto 0);
42      variable C :integer range 0 to 81;
43
44      begin
45      A := DIP(7 downto 4);
46      B := DIP(3 downto 0);
47
48      A_integer := to_integer(unsigned(A));
49
50      B_integer := to_integer(unsigned(B));
51
52      C := A_Integer * B_Integer;
53      if (c > 79) then
54          C2 := 8;
55          C1 := C - 80;
56  >   elsif(c > 69) then ···
59  >   elsif(c > 59) then ···
62  >   elsif(c > 49) then ···
65  >   elsif(c > 39) then ···
68  >   elsif(c > 29) then ···
71  >   elsif(c > 19) then ···
74  >   elsif(c > 9) then ···
77  >   else ···
80      end if;
81      C1 := C1 + 48;
82      C2 := C2 + 48;
83
84      Yekan := std_logic_vector(to_unsigned(C1, 8));
85      Dahgan := std_logic_vector(to_unsigned(C2, 8));
```

- **نمایش اطلاعات خروجی:**

در اینجا نیاز داریم اطلاعات سریالی که در خروجی **tx** آمده را نمایش دهیم. برای این کار از یک مبدل استفاده شده که خروجی را به **usb** منتقل کنیم تا بتوانیم با استفاده از نرم افزار **docklight** این خروجی سریال را مشاهده کنیم.

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 "TX" LOC = P40;

NET "DIP[0]" LOC = P171;
NET "DIP[1]" LOC = P169;
NET "DIP[2]" LOC = P168;
NET "DIP[3]" LOC = P167;
NET "DIP[4]" LOC = P166;
NET "DIP[5]" LOC = P165;
NET "DIP[6]" LOC = P162;
NET "DIP[7]" LOC = P161;
```

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.