جلسه‌ی پیش، تمرین در کلاس حل شد.

ادامه‌ی درس.

نحوه‌ی کار کردن با فایل‌ها:

```
TYPE myfile IS FILE OF character  --> در type تایپ پشتیبانی
                                      در VHDL
FILE fp : my_file;                --> در این حالت فایل txt داریم.
...
FILE_OPEN(fp, "input.txt", read_mode)
                    --> write-mode, append-mode
...
read(fp, c);   --> کاراکتر خوانده شده
FILE_CLOSE(fp);
```

معمولاً در خواندن از فایل، در بازه‌های زمانی مختلف (interval مشخص)، اطلاعات خوانده می‌شود و به سیگنال اعمال می‌شود.

برای مثال می‌خواهیم فایلی به فرمت زیر را بخواهیم که برای این کار تابع روبرو را تعریف می‌کنیم:

RST	OPCODE
0	000
0	000
1	000
1	001

```
PROCEDURE read_test_from_file(
        SIGNAL it : IN TIME;
        SIGNAL nr : OUT STD_LOGIC;
        SIGNAL op : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)) IS
    TYPE myfile IS FILE OF character;
    FILE fp : myfile;
    VARIABLE c : character;
    VARIABLE current_time : TIME := 0 ns;
    VARIABLE line_number : integer := 1;
BEGIN
    FILE_OPEN(fp, "file.txt", read_mode);
    --ignore line one (header)
    FOR i IN 0 TO 13 LOOP
        READ(fp, c);
    END LOOP;
```

نکته: این مثال، ادامه‌ی کد تمرین جلسه‌ی قبل است.

نکته: در انتهای فایل ورودی یک enter گذاشته شده است تا با فرمت کلی تابع نوشته شده، به درستی خوانده شود.

نکته: کد این جلسه و جلسه‌ی قبل در courseware گذاشته شده است.

```
WHILE (NOT ENDFILE(fp)) LOOP
    --READ nr
    READ(fp,c)

    IF c='0' THEN

        nr <= TRANSPORT '0' AFTER current_time;

    ELSE

        nr <= TRANSPORT '1' AFTER current_time;

    END IF;
    ...
    READ(fp,c) --read carriage return:
    READ(fp,c) -- read line feed
    current_time := current_time + it;

    line_number := line_number+1;

END LOOP;

FILE_CLOSE(fp);

END PROCEDURE read_test_from_file;
```

نکته: این تابع اگر در کد صدا زده شود، چون ورودی هایش تغییر نمی‌کند، تنها یک بار در زمان صفر شبیه سازی اجرا می‌شود.

به این علت، انتهای فایل مربوط به هست یک enter اضافه گذاشتیم.

ادامه‌ی درس

مثال نوشتن نتیجه در فایل (مقادیر رجیسترها):

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;    USE std.textio.ALL;
USE ieee.numeric_std.ALL;
PROCEDURE write_results (

        SIGNAL r1,r2,dr,ac:IN std_logic_vector(15 DOWNTO 0) IS

    TYPE text_file IS FILE OF string;

    FILE fp : text_file; --or better FILE fp: TEXT;

    VARIABLE str : string(16 DOWNTO 1);
BEGIN

    FILE_OPEN(fp,"mini_core_results.txt", APPEND_MODE);

    lv_to_str(r1,str);
    write(fp,str);

    write(fp," ");
    ...
    write(fp,newline);

    FILE_CLOSE(fp);

END write_results;
```

→ from textio package

↳ بازه رشته می‌تواند شامل عدد منفی باشد.

یک تابع که خودمان ساخته‌ایم و کار آن ساخت رشته برای std_logic_vector است. در نوشتن تابع دقت شود که اندیس های رشته باید مثبت باشند.

```
VARIABLE linefeed : STRING(1 DOWNTO 1);
linefeed(1) := lf;
```

↳ در کتابخانه‌ی استاندارد تعریف شده است.

PAPCO

برای این که مقادیر در مواقع تغییر مقدار سیگنال‌ها در فایل چاپ شوند، بهتر است تابع در بدنه‌ی اصلی architecture صدا زده شود تا تنها در دفعاتی که تغییر داریم، صدا زده شود. (طبق اصل کدهای concurrent)

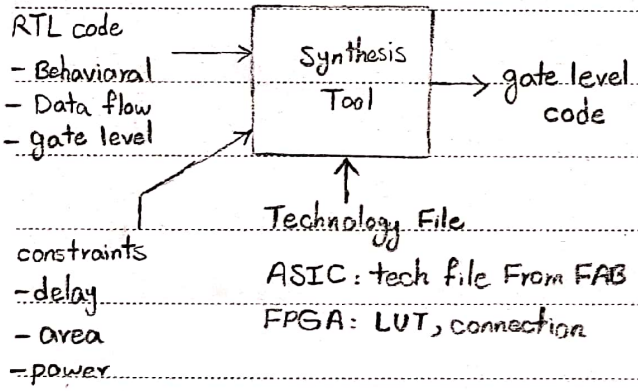
نکته: variable‌ها در درون توابع و procedure‌ها در هر بار صدا زده شدن تابع از نوع مقداردهی می‌شوند.

به همین دلیل برای این که بار اول فایل را باز کند، یک پارامتر به نام mode به تابع اضافه می‌کنیم که براساس آن write یا append می‌کند. درعوض مقدار این mode در خارج از تابع و در پیاده‌سازی architecture یا بار اول write بوده و دفعه‌ی بعدی همواره append است. مثلا به روش زیر:

```
mode <= WRI WHEN nrst = '0' ELSE APP;
```

سنتز logic:

در مورد constraint‌ها، ما یک مورد را انتخاب می‌کنیم و به احتمال زیاد دو مورد دیگر، زیاد می‌شوند.



مجموعه‌های قابل سنتز:

Rule 1: TIME cannot be synthesized: delay, wait, now (دستوری که زمان کنونی را برمی‌گرداند)

مثال: y <= NOT a AFTER 10ns;

مثال: mode <= WRI WHEN now = 0 ELSE APP;

Rule 2: In combinational PROCESS

- all inputs should be appear in PROCESS sensitivity list
- in one execution of a process, all output should be determined
- all states of inputs should be checked in conditions

Rule3 : loop can be synthesized if the iteration number of them is fixed

مثال: FOR i IN 0 TO x LOOP -- not good

نکته: در مورد Rule7 ، WAIT ON و WAIT UNTIL می‌تواند با وجود شرایطی سنتز شوند.

Rule4: Just one edge of clock can be used in a latch.

نکته: کد رو برو سنتز می‌شود چون در دیاگرام واقعی هیچ FF وجود ندارد که بتواند بر روی هر دو لبه کلاک کار کند.

```
PROCESS (clk) BEGIN
    IF clk='1' THEN
    ...
    ELSIF clk='0' THEN
    ...
    END IF;
END PROCESS;
```

ادامه‌ی درس

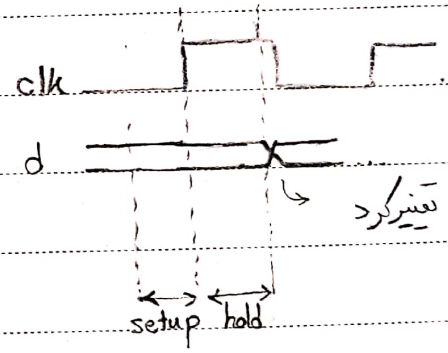
ادامه‌ی قواعد سنتز:

Rule5: Setup time and hold time

Hold time: a time interval after edge clock in which data should be remained stable.

Setup time: a time interval before clock edge in which data should be stable.

so: clock period > delay(critical path) + T_{setup}



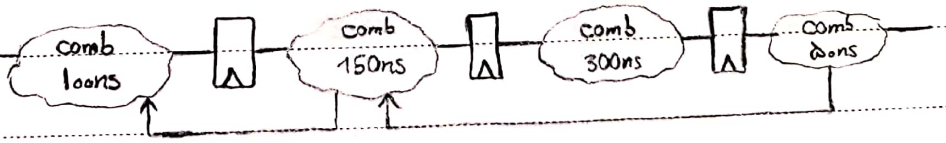
Rule6: Reseting

Two way:

- IF clk='1' THEN
  IF nrst='0' THEN
  } ریست سنکرون

- IF nrst='0' THEN
  ELSIF clk='1' THEN
  } ریست آسنکرون

سخت افزار فرق دارد.



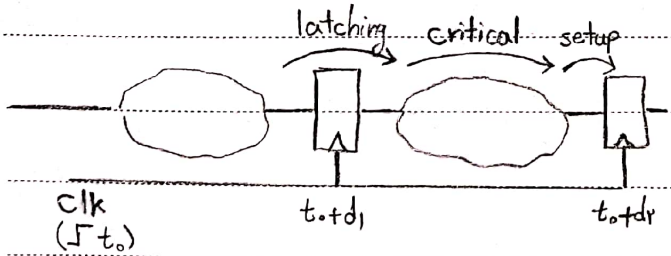
نکته: اگر قرار باشد کل FF های مدار را reset کنیم و به همه سیگنال ریست را وصل کنیم، مشکلی نخواهیم داشت. اما گاهی تعداد FF ها بسیار زیاد است (مثلا وجود cache) گاهی وقت‌ها ضرورت ندارد که کل FF های مدار ریست شود مثلاً اگر مدار صفحه‌ی قبل دارای فیدبک باشد می‌توانیم FF ورودی را reset کنیم و تا چند کلاک صبر کنیم تا کل مدار ریست شود. در درس ما نیاز به چنین بهینه سازی نیست و برای همه‌ی FF ها reset گذاشته شود.

نکته: ریست کردن اولیه‌ی مدار دارای نکته‌های زیادی می‌باشد. از طرفی ریست اولیه‌ی مدار اهمیت بسیار زیادی دارد چون ابتدای شروع کار مدار است. معمولاً ریست‌ها به صورت active low هستند چون در آن صورت پیاده سازی FF بوسیله NAND خواهد بود و سریع‌تر بوده و مقدار delay و power آن نسبت به AND کمتر است.

در مدار اکثر سیم‌ها به جاهای کمی وصل هستند و به همین دلیل fan out آن‌ها کم است. اما دو مورد clock و reset دارای fan out زیادی هستند. معمولاً یک پایه‌ی جداگانه برای clock گذاشته می‌شود و عملیات routing آن با تجهیزاتی (FPGA, ...) خاص انجام می‌شود تا به درستی به همه برسد.

اما در مورد ریست، در FPGA ها معمولاً پایه‌ی جداگانه نداریم. به همین دلیل تا جای ممکن باید fan out سیم ریست کم باشد. (معمولاً کمتر از ۲۰ یا ۳۰) (برای clock بیشتر از ۱۰۰۰ تا هم می‌رود چون FPGA آن را بافر می‌کند)

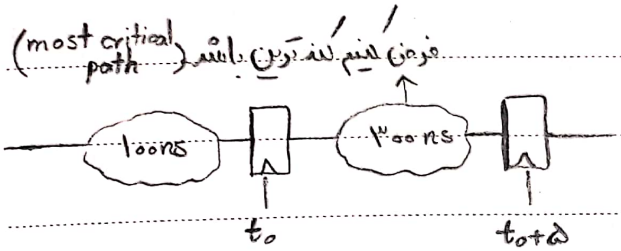
مفهومی به نام power-on-reset وجود دارد. ما وقتی سیستم روشن می‌شود، reset را فعال می‌کنیم. اما در ابتدای کار که سیستم روشن می‌شود ممکن است برای چند لحظه مقدار آن غلط باشد و سیستم وارد وضعیت اشتباه شود. برای همین سیم ریست را در حالت active low با یک مقاومت به زمین وصل می‌کنند تا اگر در لحظه‌ی اول هنوز data آن نیومده، مقدار صفر داشته باشد.



نکته: ممکن است همه‌ی کلاک‌ها، هم‌زمان به مدار نرسند.

clock skew: the time difference between time of soonest clock edge and the time of the latest clock edge

مثلا: input clock edge: 100 ns
soonest clock edge: 101 ns
latest edge: 108 ns
} clock skew = 108 - 101 = 7 ns



نکته: گاهی clock skew می‌تواند طول کلاک را کاهش دهد.

به دلیل ۵ns تاخیر FF دوم، می‌توانیم طول کلاک را ۵ تا کمتر بگیریم (مثلا ۲۹۵ واحد زمانی) و مشکلی پیش نیاید. دقت شود که شرایط باید خاص باشد (مثلا delay بعد از most critical path باشد و قبل آن delay نباشد خیلی خاص است.)

به همین دلیل ما clock skew scheduling داریم که راحت نیست.