ادامه‌ی درس:

یادآوری:

1) during the process execution, time is stoped.

2) all of the assignment, are not aplied to end of the process.

3) at end line of the process, all of the assignment will be done.

results:

1) each signal, will be changed during the process at most one time

2) all of process seems as a concurrent multiple assignment.

مثال:

یادآوری مثال جمع کننده‌ی جلسه‌ی قبل:

```
PROCESS (a,b,cin) BEGIN
    c(0) <=cin;
    FOR i IN 0 TO 31 LOOP
        sum <= a(i) XOR b(i) XOR c(i);
        c(i+1) <= ...
    END LOOP;
    cout <= c(32);
END PROCESS;
```

اگر به شکل رو به رو آن را بنویسیم اشتباه است:

چون c(i+1) ما به مقدار c(i) نیاز دارند

اما مقدار جدید c(i) تا انتهای پروسس، کی

خواهد شد. ما نیاز داریم که این مشکل را رفع

کنیم.

نکته:

```
SIGNAL c : std_logic_vector(0 TO 32);
```

Name of object type

→ class of object (SIGNAL, VARIABLE, FILE, ACCESS)

تفاوت variable با signal:

۱) یک variable تنها در بین process و begin و end آن بین process های مختلف share نمی شود.

۲) برای انتساب مقدار به variable از := استفاده می شود.

۳) انتساب یک مقدار به variable، هر زمان صورت بگیرد همان لحظه اعمال می شود.

مثال DFF:

```
ENTITY dff_async IS PORT (nrst,clk,din : IN std_logic;   -- nrst: not reset
                          q,nq         : OUT std_logic);
END dff_async;
ARCHITECTURE behave OF dff_async IS
BEGIN
P1: PROCESS (nrst, clk)
    VARIABLE t: std_logic;
  BEGIN
    IF nrst = '0' THEN
       t := '0';
    ELSIF clk'event AND clk='1' THEN
       t := din;
    END IF;
    q <= t;
    nq <= NOT t;
  END PROCESS P1;
END behave;
```

حالت آسنکرون

```
P1: PROCESS (clk)
  VARIABLE t: std_logic;
BEGIN
  IF clk='1' THEN
    IF nrst='0' THEN
      t := '0';
    ELSE
      t := din;
    END IF;
  END IF;
  q <= t;
  nq <= t;
END PROCESS P1;
```

نکته: متغیر (variable) تعریف شده در قسمت process، مقدار خود را در اجراهای مختلف process حفظ می کند.

نکته: اگر بخواهیم به جای dff، یک رجیستر N بیتی تعریف کنیم، که مشابه مختص قبل می باشد و موارد زیر تغییر می کند:

۱) در قسمت تعریف entity بخش generic اضافه می شود:

```
Generic (n : integer := 8)
```

۲) قسمت port ها نیز مشابها تغییر کند:

```
PORT(..., din : IN std_logic_vector(n-1 DOWNTO 0);
         q : OUT std_logic_vector(n-1 DOWNTO 0));
```

۳) تعریف متغیر process به صورت مشابه تغییر می کند:

```
VARIABLE t : std_logic_vector(n-1 DOWNTO 0);
```

۴) برای مقداردهی t با مقدار صفر در بخش nrst از دستور روبرو استفاده می شود:

```
t := (OTHERS => '0');
```

مثال: از مزیت های به خاطر سپرده شدن مقدار variable در دفعات مختلف اجرای process، استفاده های مختلفی می توان داشت. در مثال زیر که یک شمارنده نشان داده شده است:

```
ENTITY counter IS
    GENERIC (n : integer := 8);
    PORT ( nrst, clk : IN std_logic ; q : OUT std_logic_vector(n-1 DOWNTO 0));
END counter;
ARCHITECTURE behave of counter IS BEGIN
    P1: PROCESS (nrst, clk)
        VARIABLE t: std_logic_vector(n-1 DOWNTO 0);
    BEGIN
        IF nrst = '0' THEN
            t := (OTHERS => '0');
        ELSIF clk'event AND clk = '1' THEN
            t := t + 1;
        END IF;
        q <= t;
    END PROCESS P1;
END behave;
```

وقتی یک signal یا variable در لیستی یک سیگنال دیگر، مقداردهی شود، آن سیگنال یا متغیر به صورت register خواهد بود.

(برای این کد باید کتاب خانه ی مورد نیاز اضافه شود: std_logic_unsigned)

PaPCO

مثال پیاده‌سازی latch:

قسمت تعریف entity آن مشابه dff است اما پیاده‌سازی آن به صورت زیر است:

```
P1: PROCESS (nrst, clk, din)
    VARIABLE t : std_logic_vector (n-1 DOWNTO 0);
BEGIN
    IF nrst = '0' THEN
        t := (OTHERS => '0');
    EISIF clk = '1' THEN
        t := din;
    END IF;
    q <= t;
END PROCESS P1;
```

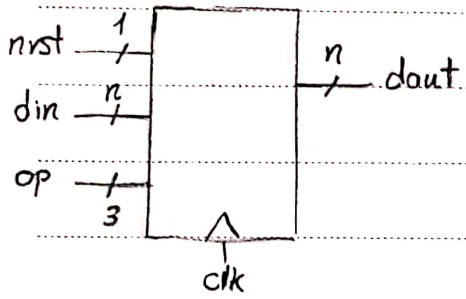
نکته: هنوز هم با وجود variable، اجرای process در مرتبه دوم قرار است. به عنوان مثال:

```
PROCESS ()
    VARIABLE c :...
BEGIN
    a <= '1';
    b <= a; ⇝ a مقدار قبلی
    c := a; ⇝ باز هم مقدار قبلی
    x <= c; ⇝ x <= a معادل
    ...
END
```

طبق گفته هنوز همه چیز در مرتبه دوم محاسبه و خروجی داده می‌شود.

نکته: اگر در هر جای از یک process، سیگنال یا متغیری بیش از یکبار مقدار بگیرد، process به درستی اجرا نخواهد شد (در ادامه‌ی توضیح استاد: مقدار آخر آن استفاده خواهد شد!!)

تمرین: طراحی universal counter



OP	function
000	no operation
001	dout <= din
010	count up
011	count down
100	logical right shift
101	logical left shift
110	circular right shift
111	circular left shift

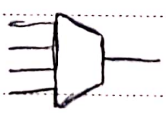
ادامه‌ی دستورات

ساختار IF در process:

```
IF condition THEN
...
ELSIF condition THEN
...
ELSE
...
END IF;
```

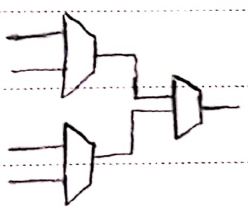
مثال پیاده‌سازی mux4x1:

```
PROCESS (sel) BEGIN
    IF sel = "00" THEN
        z<=a;
    ELSIF sel = "01" THEN
        z<=b;
    ...
    END IF;
END PROCESS;
```



=

```
IF sel(1)='0' THEN
    IF sel(0)='0' THEN
        z<=a
    EISE
        z<=b
    ENDIF;
ELSE
...
END IF;
```



PAPCO

ساختار case برای مثال قبل:

```
PROCESS (sensitivity list) BEGIN
    CASE sel IS
        WHEN "00" => Z<=a;
        WHEN "01" => Z<=b;
        ...
        WHEN OTHERS => Z<=d;
    END CASE;
END PROCESS;
```

ساختار حلقه:

(i ← به صورت ضمنی تعریف شده)

```
FOR i IN 0 TO 31 LOOP
    ...
END LOOP;
```

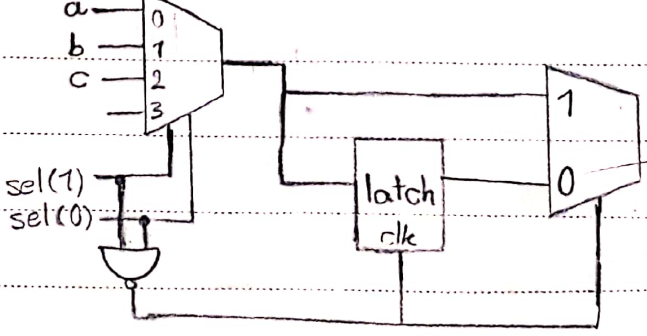
(cntr ← می تواند variable یا signal باشد.)

```
WHILE cntr <100 LOOP
    ...
END LOOP;
```

نکته:

اگر تمامی حالات در case بیان نشود یا برخی از سیگنال ها مقدار نگیرند، این باعث می شود که مدار به همراه latch پیاده سازی شود. به عنوان مثال کد روبرو در صورت حساس بودن به همه ی سیگنال ها به صورت زیر پیاده سازی می شود.

```
CASE sel IS
    WHEN "00" => Z<=a;
    WHEN "01" => Z<=b;
    WHEN "10" => Z<=c;
END CASE;
```



نکته: process بدون لیست حساسیت مثل یک حلقه‌ی بی نهایت است.

هرجمله‌ای مثل process یکبار در زمان شروع اجرا می‌شود و در دفعات بعدی تنها در تغییر انجام می‌شود.

پس process با لیست حساسیت یکبار در زمان صفر برای گرفتن مقدارهای اولیه اجرا می‌شود و در دفعات بعدی تنها با تغییر در لیست حساسیت اجرا می‌شد.

اما process بدون لیست حساسیت دوام اجرا می‌شود.

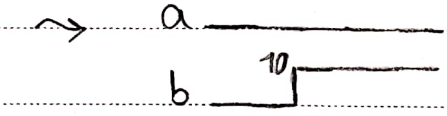
برای خارج شدن از آن معادل زبان های دیگر نیازمند exit هستیم. برای این کار در زبان VHDL از دستور wait برای خارج شدن استفاده می‌شود.

```
PROCESS BEGIN
    a<='0';
    WAIT;   -- End process forever
END PROCESS;
```

اما wait حالات دیگری هم دارد:

```
PROCESS BEGIN
    a<='0';
    WAIT FOR 10 ns;
    a<='1';
    WAIT;
END PROCESS;
```

در این حالت process در صفر نانو ثانیه اجرا می‌شود و به همین دلیل قابل سنتز است. (معادل هیچ hw نیست)



نکته: process بدون لیست حساسیت حتماً باید دارای wait باشد در آن صورت دیگر قابل سنتز نیست. این خود یک مزیت محسوب می‌شود چون درست دهها نیاز به چنین مواردی داریم.

نکته: process با لیست حساسیت نمی‌تواند دارای wait باشد و خطای compile دارد. (برای مدل کردن hw است)