ساختارهای ترتیبی و همروند

Sequential and Concurrent Structures

ساختارهای ترتیبی

- **ساختارهای ترتیبی:**
  - فرایند (process)
  - روال (procedure)
  - تابع (function)

ساختارهای ترتیبی

- انتساب سیگنال:

```
MYSIG <= (A or B) xor C after 10 ns;
```

```
CLK <= not CLK after PERIOD/2;
```

```
process (...)
begin
 ⋮
S1 <= '0', '1' after 5 ns, '0' after 7 ns, '1' after 12 ns, ...;
 ⋮
end process;
```

❑ انتساب مقدار اولیه: با نماد :=

```
signal S1: bit := '0';
```

ساختارهای ترتیبی

- **انتساب متغیر:**

```
MYVAR := (A or B) xor C;
```

- بدون زمان

ساختارهای ترتیبی

- If-then-else:

```
if CONDITION then
   -- sequential statements
end if;
```

```
 if CONDITION then
   -- sequential statements
else
   -- sequential statements
end if;
```

```
if CONDITION then
   -- sequential statements
elsif CONDITION then
   -- sequential statements
. . .
else
   -- sequential statements
end if;
```

ساختارهای ترتیبی

- Case-when:

```
case  EXPRESSION  is

   when  VALUE_1  =>
     -- sequential statements

   when  VALUE_2 | VALUE_3   =>
     -- sequential statements

   when  VALUE_4 to VALUE_N  =>
     -- sequential statements

   when   others =>
     -- sequential statements

end   case ;
```

- سازگاری نوع دادهٔ مقادیر و عبارت

ساختارهای ترتیبی

```
entity ...
   port (SELECT: in std_logic_vector (1 downto 0);
         D0, D1, D2, D3: in std_logic_vector (7 downto 0);
         DOUT: in std_logic_vector (7 downto 0));
end entity;

architecture ...
begin
:
process (SELECT, D0, D1, D2, D3)
begin
   case SELECT is
      when  "00" => DOUT <= D0;
      when  "01" => DOUT <= D1;
      when  "10" => DOUT <= D2;
      when  "11" => DOUT <= D3;
      when  others => DOUT <= "XXXXXXXX";
   end case ;
end process;
:
end architecture;
```

- همهٔ حالت‌ها باید پوشش داده شوند.
- `when 2 to 5` صحیح اما غلط

```
when "00" to "11"
```

- همهٔ حالت‌ها باید ناهمپوشان باشند:

```
when 2 => …
when 1 to 4 =>
```

ساختارهای ترتیبی

- loop:
  - for loop

```
entity FOR_LOOP is
   port (A : in    integer range 0 to 3;
         Z  : out bit_vector (3 downto 0));
end FOR_LOOP;

architecture EXAMPLE of FOR_LOOP is
begin
   process (A)
   begin
      Z <= "0000";
      for  I  in  0 to 3  loop
        if (A = I) then
           Z(I) <= `1`;
        end if;
      end loop;
   end process;
end EXAMPLE;
```

- مولد توازن در کتاب

ساختارهای ترتیبی

```
entity CONV_INT is
      port (VECTOR: in   bit_vector(7
downto 0);
               RESULT:  out integer);
end CONV_INT;
architecture C of CONV_INT is
begin
   process(VECTOR)
      variable TMP: integer;
      variable I       : integer;
   begin
      TMP := 0;
      I := VECTOR'high;
      while (I >= VECTOR'low) loop
         if (VECTOR(I)='1') then
            TMP := TMP + 2**I;
         end if;
         I := I - 1;
      end loop;
      RESULT <= TMP;
   end process;
end C;
```

- loop:
  - while loop

ساختارهای ترتیبی

- wait:
  - Wait for ❑

```
process
begin
    S1 <= '1';
    wait for 10 ns;
    S1 <= '0';
    wait for 5 ns;
    S1 <= '1';
    wait for 12 ns;
    S1 <= '0';
    ...
end process;
```

ساختارهای ترتیبی

- wait:
  - Wait on ❑

```
wait on SIG1, SIG2, ...;
```

```
process
begin
...
   wait on CLK;
   if (CLK = '1') then
      REG <= DATA;
...
end process;
```

ساختارهای ترتیبی

- **wait:**
  - Wait until
    - نیاز به تغییر هم دارد.

```
wait until CONDITION;
```

```
process
begin
...
   wait until CLK = '1';
      REG <= DATA;
...
end process;
```

- **wait:**
  - Wait نامحدود

```
wait;
```

ساختارهای ترتیبی

```
assert DESIRABLE_CONDITION
report STRING
severity LEVEL;
```

```
architecture BEHAVIORAL of D_SR_FLIPFLOP is
signal STATE : bit := '0';
begin
   DFF: process (RST, SET, CLK)
   begin
      assert
         (not (SET = '1' and RST = '1'))
      report
         "set and rst are both 1"
      severity note;
      if SET = '1' then
         STATE <= '1';
      elsif RST = '1' then
         STATE <= '0';
      elsif CLK = '1' and clk'EVENT then
         STATE <= D;
      end if;
   end process dff;
   Q <= STATE;
   QB <= not STATE;
end BEHAVIORAL;
```

- assert
- report
- severity
  - note
  - warning
  - error
  - failure

ساختارهای ترتیبی

- **فرایند (process)**

۱. با لیست حساسیت:

- مجموعه‌ای از سیگنال‌ها
- یک بار اجرا
- تکرار در صورت تغییر در هر یک از سیگنال‌های لیست

```
architecture BEHAVIORAL of STH is
begin
   L1: process (S1, S2, S3)
      variable V1: bit;
   begin
      V1 := '0';
      ⋮
      if (S1 = V1) then ...
      ⋮
   end process L1;
end BEHAVIORAL;
```

ساختارهای ترتیبی

- فرایند (process)

۲. با wait

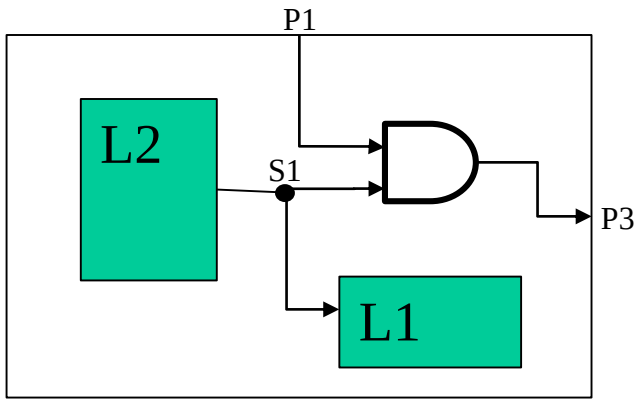
- فقط یکی از این دو حالت

```
architecture BEHAVIORAL of STH is
begin
   L1: process
      variable V1: integer;
   begin
      V1 := S1;
      ⋮
      wait for 5 ns;
      ⋮
   end process L1;
end BEHAVIORAL;
```

ساختارهای ترتیبی

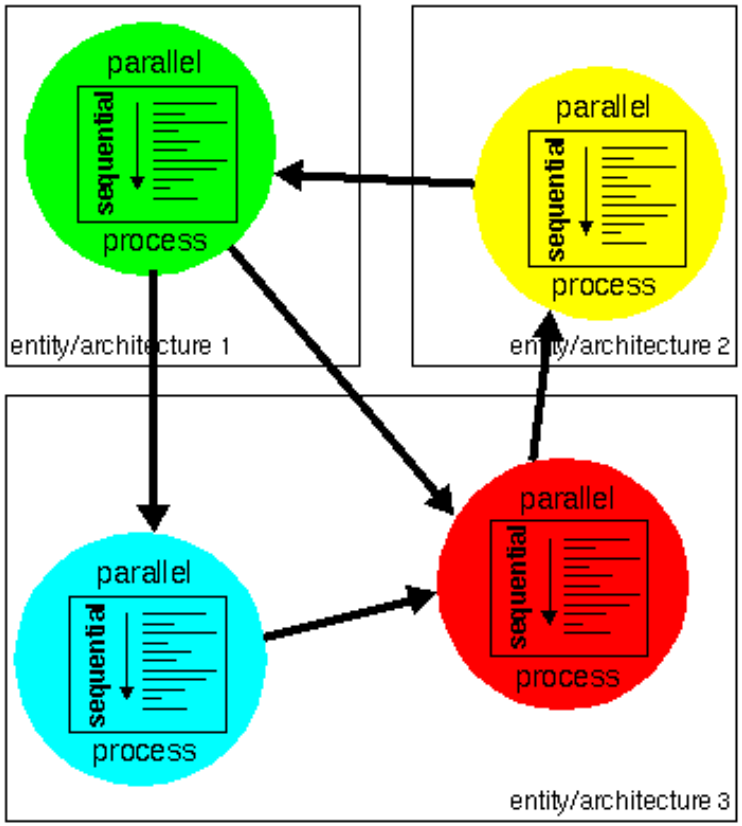
- فرایند (process)
  - فرایند یک دستور **همروند** است

```
architecture P_ARCH of P_ENT is
   signal S1: std_logic;
begin
   G1:ANDGATE port map (P1, S1, P3);
   :
   L1: process (S1)
   begin
      :
      if (S1 = '0') then ...
      :
   end process L1;
   L2: process
   begin
      :
      S1 <= '0';
      :
   end process L1;
   :
end architecture P_ARCH;
```



  - ارتباط فرایندها با سیگنال
  - ارتباط architectureها با درگاه

VHDL Communication Model



multi_process_exec2.avi

ساختارهای ترتیبی

- فرایند (process)
  - توصیف FF با wait

```
entity DFF is
      port (D, CLK: in std_logic:
            Q, QBAR: out std_logic);
end entity DFF;
```

```
architecture RTL of DFF is
begin
   L:process
   begin
      wait on CLK;
      if (CLK = '1') then
         Q <= D;
      end if;
   end process L;

   QBAR <= not Q;
end architecture RTL;
```

- شبیه‌ساز در هر پریود دو بار فرایند را اجرا می‌کند.
- اجرای دستور انتساب به QBAR؟

ساختارهای ترتیبی

- فرایند (process)
  - توصیف FF با لیست حساسیت

```
entity DFF is
      port (D, CLK: in std_logic:
            Q, QBAR: out std_logic);
end entity DFF;
```

```
architecture RTL of DFF is
begin
   L:process (CLK)
   begin
      if (CLK = '1') then
         Q <= D;
      end if;
   end process L;

   QBAR <= not Q;
end architecture RTL;
```

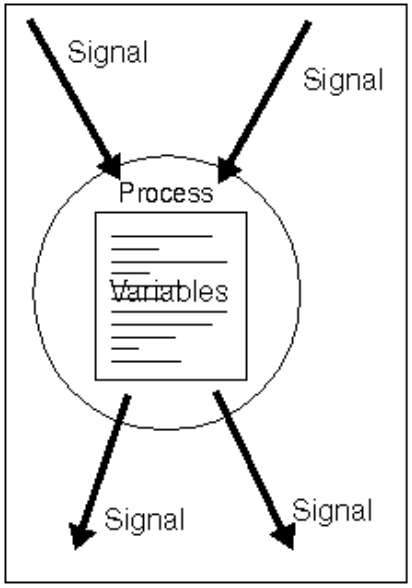
متغیر و سیگنال

- **متغیر و سیگنال در فرایند:**
  - متغیر: برای ذخیرهٔ مقادیر موقت و استفاده از مقدار آن در دستورهای بعدی
  - سیگنال: حامل سیگنال سخت‌افزاری
  - متغیر: انتساب فوری
  - سیگنال: تعویق انتساب تا زمان تعلیق فرایند
    - حتی بدون after
    - علت: عدم وابستگی به زمان رسیدن به دستورپس از تغییر S2
  - مدت اجرای فرایند از یک تعلیق به تعلیق دیگر: صفر

```
architecture ...
   signal S1, S2: std_logic;
begin
   P1:process (S2, ...)
   begin
      if (S1 = '1') then ...
      :
      end if;
      :
   end process P1;

   P2:process (S2, ...)
   begin
      :
      S1 <= '1';
      :
   end process P2;
end architecture RTL;
```

متغیر و سیگنال



- **کاربرد عملی متغیر و سیگنال در فرایند:**
  - متغیر: ابتدای فرایند انتقال مقدار سیگنال در متغیر
  - محاسبات با متغیر
  - انتهای فرایند: انتقال نتیجه به سیگنال
  - عدم دسترسی به متغیر در بیرون از فرایند

```
architecture ...
   signal S1: std_logic;
begin
   P1:process (S1, ...)
      variable V1: integer;
   begin
      V1 <= S1;
      :
      arithmetic/logic operations on V1;
      :
      S1 <= V1;
   end process P1;

end architecture RTL;
```

متغیر و سیگنال

Guess what values will have the A, B and E signals after D changes its value from 1 to 2. Verify your answer by clicking the button.

```
process(C,D)
begin
   A <= 2;
   B <= A + C;
   A <= D + 1;
   E <= 1 * 2;
end process;
```

A = 1
B = 1
C = 1
D = 1
E = 1

Change D to 2

انتساب سیگنال و فرایند

- شباهت انتساب به سیگنال در بدنهٔ همروند و در اجرای فرایند

```
architecture ...
   signal S1, S2, S3: std_logic;
begin
   ⋮
   S3 <= S1 xor S2;
   ⋮
end architecture RTL;
```

```
architecture ...
   signal S1, S2, S3: std_logic;
begin
   P1:process (S1, S2)
begin
      S3 <= S1 xor S2;
end process P1;

end architecture RTL;
```

زیربرنامه‌ها

- **تابع:**
  - دریافت مقادیر با پارامتر
  - انجام محاسبات
  - بازگرداندن یک نتیجه

```
architecture BEHAVIORAL of STH is
begin
   L1: process (...)
   begin
      ⋮
      if (FCOUT (A, B, CIN) = '0' then ...
      ⋮
   end process L1;

   RES <= FCOUT (A, B, CIN) or FLAG;

end BEHAVIORAL;
```

  - فراخوانی:
    - به جای عبارت
    - در بدنهٔ ترتیبی
    - در بدنهٔ همروند
  - دستورهای ترتیبی غیر از wait

زیربرنامه‌ها

- تابع:

```
function COUNT1 (X: std_logic_vector (7 downto 0)) return integer is
   variable NoOfOnes : integer := 0;
begin
   for I in X'range loop
      if X(I) = '1' then
         NoOfOnes := NoOfOnes + 1;
      end if;
   end loop;
   return NoOfOnes;
end COUNT1;
```

```
L1: process (...)
   variable A: std_logic_vector (0 to 3);
   variable A: std_logic_vector (15 downto 0);
   begin
      if COUNT1 (A) > 2 then ...
      ⋮
      for I in 1 to COUNT1(B) loop
      ⋮
   end process L1;
end BEHAVIORAL;
```

زیربرنامه‌ها

- **روال (procedure):**
  - دریافت مقادیر با پارامتر
  - انجام محاسبات و عملیات (همهٔ دستورهای ترتیبی)
  - فراخوانی:
    - به جای دستور
    - در بدنهٔ ترتیبی
    - در بدنهٔ همروند
  - پارامترها:
    - ورودی (in)
    - خروجی (برای بازگرداندن یک یا چند نتیجه) (out)
    - ورودی–خروجی (inout)

زیربرنامه‌ها

- روال (procedure):

```
procedure MINMAX (signal S1, S2: in integer;
                  signal MIN, MAX: out integer) is
begin
   if S1 > S2 then
      MAX <= S1;
      MIN <= S2;
   else
      MAX <= S2;
      MIN <= S1;
   end if;
end procedure MINMAX;
```

زیربرنامه‌ها

- روال (procedure):
  - فراخوانی در بدنهٔ ترتیبی:
    - زمان رسیدن به آن
  - فراخوانی در بدنهٔ همروند:
    - هر بار پارامتر ورودی سیگنال تغییر کند
  - همهٔ دستورهای ترتیبی مجازند
  - Wait در صورتی که تابعی آن را فراخوانی نکرده باشد!

زیربرنامه‌ها

- **حالت و کلاس پارامترها در تابع و روال:**

حالت‌ها و کلاس‌های مجاز

	تابع	روال	
حالت	in	in	out inout
کلاس	signal constant	signal variable constant	signal variable
بدون حالت	file		

❑ تابع:

- constant
- signal
- file

❑ روال:

- constant
- signal
- file
- variable

❑ کلاس پیش‌فرض:

- برای in: constant
- برای out: variable
- برای inout: variable

زیربرنامه‌ها

- **گرانبار کردن (overloading):**
  - چند زیربرنامه با نام یکسان:
    - تعداد و/یا نوع پارامترها متفاوت
  - نام یکتای تابع:
    - نام تابع + نوع دادهٔ هر پارامتر + نوع مقدار بازگردانده شده
    - نام پارامتر و کلاس پارامترها جزو نام یکتا نیست.

```
function FUNC1 (integer I: bit B) return bit;
function FUNC1 (real r: bit B) return bit;
```

زیربرنامه‌ها

- پکیج textio:
  - ❑ استخراج داده‌ها با نوع داده‌های گوناگون از یک line:

```
procedure READLINE (file F: TEXT; L: inout LINE);
```

```
procedure READ (L          : inout line;
                VALUE : out    character );

procedure READ ( L         : inout line;
                 VALUE : out    character;
                 GOOD : out     boolean );

procedure READ ( L         : inout line;
                 VALUE : out integer );

procedure READ ( L         : inout line;
                 VALUE : out    integer;
                 GOOD : out    boolean );
```

عملگرها

- عملگر
  - مانند تابع با یک یا دو عملوند
  - طراح می‌تواند گرانبار کند
    - نه برای انواع داده‌ای که عملگر برای آن تعریف شده
    - نه برای نمادهای غیر عملگر (مثل @)
    - تعداد و/یا نوع پارامترها متفاوت

عملگرها

```
package P_BIT_ARITH is
  function "+" (L: bit_vector; R: bit_vector) return bit_vector; -- 1
  function "+" (L: integer; R: bit_vector) return bit_vector;     -- 2
  function "+" (L: bit_vector; R: integer) return bit_vector;     -- 3
  function "+" (L: bit_vector; R: bit_vector) return integer;     -- 4
end P_BIT_ARITH;

use work.P_BIT_ARITH.all;

entity OVERLOADED is
  port(A_VEC, B_VEC: in bit_vector(3 downto 0);
       A_INT, B_INT: in integer range 0 to 15;
       Q_VEC: out bit_vector(3 downto 0);
       Q_INT: out integer range 0 to 15);
end OVERLOADED;

architecture EXAMPLE of OVERLOADED is
begin
  Q_VEC <= A_VEC + B_VEC; -- a
  Q_VEC <= A_INT + B_VEC; -- b
  Q_VEC <= A_VEC + B_INT; -- c
  Q_VEC <= A_INT + B_INT; -- d  → Error: INT+INT returns INT
  Q_INT <= A_VEC + B_VEC; -- e
  Q_INT <= A_INT + B_INT; -- f
end EXAMPLE;
```

ساختارهای همروند

Concurrent Structures

دستورهای همروند

- **ساختار همروند:**
  - پیش‌فرض
- **دستورهای همروند:**
  - انتساب سیگنال
  - ایجاد نمونه
  - فراخوانی روال
  - فرایند
  - انتساب سیگنال شرطی
    - when-else
  - انتساب سیگنال انتخابی
    - with-select-when

Conditional Signal Assignment

```
TARGET <= VALUE_1
when CONDITION_1
else
        VALUE_2
when CONDITION_2
else
        . . .
        VALUE_n;
```

- **Condition is a Boolean expression**
- **Mandatory else path, unless unconditional assignment**
  - conditions may overlap
  - priority
- **Similar to if ..., elsif ..., else constructs**

- **هرگاه تغييري روي سيگنالهاي سمت راست رخ دهد اين انتساب بار ديگر ارزيابي مي شود (هميشه فعال است)**

Conditional Signal Assignment

```
entity CONDITIONAL_ASSIGNMENT is
  port (A, B, C, X : in   bit_vector (3 downto 0);
        Z_CONC : out bit_vector (3 downto 0);
        Z_SEQ  : out bit_vector (3 downto 0));
end CONDITIONAL_ASSIGNMENT;

architecture EXAMPLE of CONDITIONAL_ASSIGNMENT is
begin
  -- Concurrent version of conditional signal assignment
  Z_CONC <= B when X = "1111" else
            C when X > "1000" else
            A;
 -- Equivalent sequential statements
  process (A, B, C, X)
  begin
    if  (X = "1111")  then
      Z_SEQ <= B
    elsif  (X > "1000")  then
      Z_SEQ <= C;
    else
      Z_SEQ <= A;
    end if;
  end process;
end EXAMPLE;
```

- توجه: در پروسس، همهٔ سیگنالهای سمت راست انتساب در لیست حساسیت آمده اند.

Selected Signal Assignment

```
with EXPRESSION select

   TARGET <= VALUE_1 when
   CHOICE_1,

                VALUE_2 when
   CHOICE_2 | CHOICE_3,

                VALUE_3 when
   CHOICE_4 to CHOICE_5,

                . . .

                VALUE_n when
   others;
```

- **Choice options must not overlap**
- **All choice options have to be covered**
  - single values
  - value range
  - selection of values ("|" means "or")
  - "when others" covers all remaining choice options
- **Similar to case ..., when ... constructs**

Selected Signal Assignment

```
entity SELECTED_ASSIGNMENT is
    port (A, B, C, X : in   integer range 0 to 15;
          Z_CONC : out integer range 0 to 15;
          Z_SEQ    : out integer range 0 to 15);
  end SELECTED_ASSIGNMENT;

  architecture EXAMPLE of SELECTED_ASSIGNMENT is
  begin
    -- Concurrent version of selected signal assignment
    with X select
      Z_CONC <= A when 0,
                B when 7 | 9,
                C when 1 to 5,
                0 when others;
    -- Equivalent sequential statements
    process (A, B, C, X)
    begin
      case X is
        when 0        => Z_SEQ <= A;
        when 7 | 9    => Z_SEQ <= B;
        when 1 to 5   => Z_SEQ <= C;
        when others => Z_SEQ <= 0;
      end case;
    end process;
  end EXAMPLE;
```

Selected Signal Assignment

```
entity MUX4 is
    port (DIN : in std_logic_vector (3 downto 0);
          SEL : in std_logic_vector (1 downto 0);: in integer range 0 to 15;
          DOUT : out std_logic);
 end MUX2;

 architecture ARCHMUX of MUX4 is
 begin
    with SEL select
      DOUT <= DIN (3) when “11”,
                DIN (2) when “10”,
                DIN (1) when “01”,
                DIN (0) when others;
 end ARCHMUX;
```