Sequential Processing

Istruzioni sequenziali: comandi eseguiti in maniera seriale, uno dopo l'altro (comune ai linguaggi di programmazione)

TUTTE le istruzioni in una `architecture` sono "concorrenti" (vengono eseguite contemporaneamente)
MA i `process` contengono sotto-istruzioni sequenziali

NB: i `process`, nel loro insieme, sono istruzioni concorrenti ma definiscono una regione della `architecture` in cui tutte le sotto-istruzioni sono sequenziali

Processi

```
ENTITY flag_gen IS
PORT(
    clock,reset: IN std_logic;
    flag: OUT  std_logic;
END flag_gen;

ARCHITECTURE behavior OF flag_gen IS

BEGIN

  PROCESS(clock,reset)-- SENSITIVITY LIST

  variable count:std_logic_vector(7 downto 0); -- "oggetti" interni(variabili,costanti...)

  BEGIN -- INIZIO DEL PROCESSO
    IF reset='1' then
        count:="00000000";
        flag<='0';
    ELSIF clock='1' and clock'event then
        count:=count+1;
        IF count="11111111" THEN
                flag<='1';
        ELSE
                flag<='0';
        END IF;
    END IF;
  END PROCESS;  -- FINE DEL PROCESSO

END behavior;
```

ISTRUZIONI ESEGUITE IN MODO SEQUENZIALE

Processi per descrivere logica sequenziale

I processi che descrivono logica sequenziale hanno nella sensitivity list SOLO il CLOCK ed eventualmente il RESET ASINCRONO

```
ENTITY reg IS
PORT(
   clock,reset: IN std_logic;
   din:  IN  std_logic_vector(7 downto 0);
   dout: OUT std_logic_vector(7 downto 0));
END reg;

ARCHITECTURE behavior OF reg IS
BEGIN
  PROCESS(clock,reset)
  BEGIN
       IF reset='1' then
              dout<=(OTHERS=>'0');
       ELSIF clock='1' and clock'event then
              dout<=din;
       END IF;
    END PROCESS;
END behavior;
```

Processi per descrivere logica combinatoria

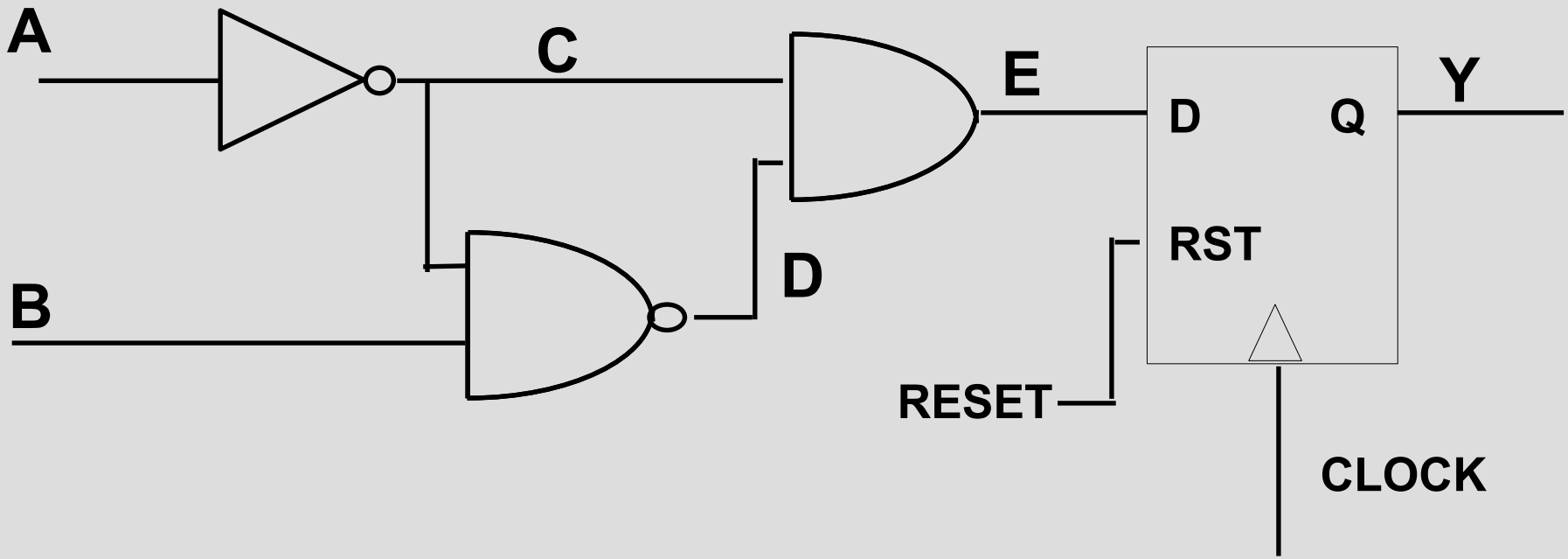
I processi che descrivono logica combinatoria hanno nella sensitivity list TUTTI gli ingressi

```
ENTITY nand_2 IS
PORT(

   a,b:  IN  std_logic;
   c: OUT std_logic);
END nand_2;

ARCHITECTURE behavior OF nand_2 IS
BEGIN
  PROCESS(a,b)-- tutti gli ingressi sono nella sensit. list
  BEGIN
       c<= NOT (a and b);
    END PROCESS;
END behavior;
```

ESEMPIO 1



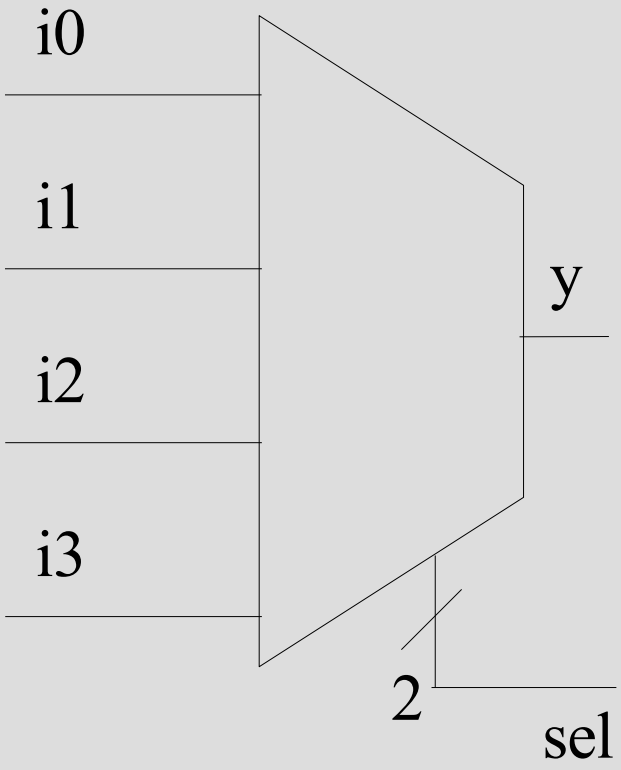

```
ENTITY comb_seq IS
PORT(
   a,b,clock,reset:  IN  std_logic;
   y: OUT std_logic);
END comb_seq;
```

ESEMPIO 1 (cont.)

```
ARCHITECTURE arch OF comb_seq IS
SIGNAL c,d,e: std_logic;
BEGIN
    c<=NOT(a);
    d<=NOT(c AND b);
    e<=c AND d;

  PROCESS(clock,reset)
  BEGIN
    IF reset='1' THEN
        y<='0';
    ELSIF clock='1' AND clock'EVENT THEN
        y<=d;
    END IF;
  END PROCESS;
END arch;
```

ESEMPIO 2




```
ENTITY mux_4x1 IS
PORT(
    i0,i1,i2,i3:  IN  std_logic;
    sel: IN std_logic_vector(1 downto 0);
    y: OUT std_logic);
END mux_4x1;

ARCHITECTURE arch OF mux_4x1 IS
BEGIN
    PROCESS(i0,i1,i2,i3,sel)
    BEGIN
        CASE sel IS
                WHEN "00" => y<=i0;
                WHEN "01" => y<=i1;
                WHEN "10" => y<=i2;
                WHEN "11" => y<=i3;
        END CASE;
    END PROCESS;
END arch;
```

Segnali e variabili (1)

Le variabili, a differenza dei segnali non sono soggette allo “schedule” degli eventi basato sui *delta* ma vengono aggiornate IMMEDIATAMENTE

```
ARCHITECTURE test1 OF mux IS
    SIGNAL x : BIT := '1';
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, x, y)
      BEGIN
        x <= in_sig XOR y;
        y <= in_sig XOR x;
      END PROCESS;
END test1;
```

```
ARCHITECTURE test2 OF mux IS
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, y)
      VARIABLE x : BIT := '1';
    BEGIN
      x := in_sig XOR y;
      y <= in_sig XOR x;
    END PROCESS;
END test2;
```

Supponendo che `in_sig` passi da '1' a '0', come varia y nei 2 casi?

Segnali e Variabili (2)

```
ARCHITECTURE sig_ex OF test IS
BEGIN
signal out_1,out_2:std_logic;
   PROCESS (a,b,c,out_1)
      BEGIN
         out_1 <= a NAND b;
         out_2 <= out_1 XOR c;
     END PROCESS;
END sig_ex;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

```
ARCHITECTURE sig_ex OF test IS
BEGIN
signal out_4:std_logic;
   PROCESS (a,b,c)
   variable out_3:std_logic;
      BEGIN
         out_3:= a NAND b;
         out_4 <= out_3 XOR c;
     END PROCESS;
END sig_ex;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	0
1+d	1	1	1	0	1

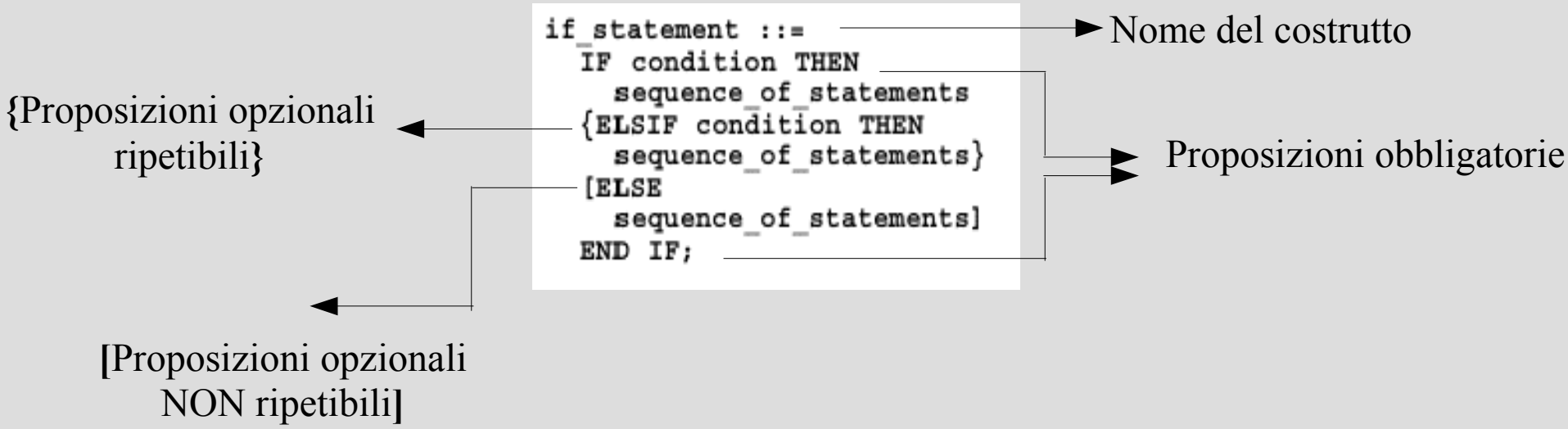
Istruzioni sequenziali

Le istruzioni sequenziali (o *sequential statements*) possono essere inserite nei process e nei sub-programs.
I principali *sequential statements* sono:

- WAIT
- IF
- LOOP
- CASE
- ASSERT

Bachus-Naur Format (BNF)

Il BNF formalizza i costrutti del VHDL. Uitle per capire il Language Reference Manual (LRM). Consente di distinguire i costrutti *opzionali* da quelli *obbligatori* e quelli i *non-ripetibili* da quelli *ripetibili*



Bachus-Naur Format (cont.)

```
if_statement ::=
  IF condition THEN
    sequence_of_statements
  {ELSIF condition THEN
    sequence_of_statements}
  [ELSE
    sequence_of_statements]
  END IF;
```

```
sequence_of_statements ::=
  {sequential_statement}
```

```
condition ::= boolean_expression
```

```
expression ::=
  relation {and relation}
  |relation {or relation}
  |relation {xor relation}
  | relation [nand relation]
  |relation [nor relation]
```

```
sequential_statement ::=
  wait_statement
  | assertion_statement
  | signal_assignment_statement
  | variable_assignment_statement
  | procedure_call_statement
  | if_statement
  | case_statement
  | loop_statement
  | next_statement
  | exit_statement
  | return_statement
  | null_statement
```

NOTA: il *corsivo* indica il tipo diespressione (*boolean*) ma tutte le espressioni condividono la stessa sintassi

WAIT

L'espressione WAIT determina la sospensione di un processo (o di una procedura)

wait [sensitivity_clause] [condition_clause] [timeout_clause];

sensitivity_clause ::= ON signal_name { , signal_name }

```
WAIT ON clock;
```

condition_clause ::= UNTIL boolean_expression

```
WAIT UNTIL clock = '1';
```

timeout_clause ::= FOR time_expression

```
WAIT FOR 150 ns;
```

Processi senza sensitivity lsit

```
Somma:  PROCESS
  BEGIN
     Sum <= A XOR B XOR Cin;
        WAIT ON A, B, Cin;
END PROCESS Summation;
```

```
Somma:
PROCESS( A, B, Cin)
  BEGIN
     Sum <= A XOR B XOR Cin;
END PROCESS Summation;
```

I due processi sono equivalenti.

ATTENZIONE

Un processo con sensitivity list NON può contenere espressioni WAIT!

Wait Until e Wait for

```
Somma:   PROCESS
  BEGIN
     Sum <= A XOR B XOR Cin;
     WAIT UNTIL A = '1';
END PROCESS Summation;
```

Dopo la prima esecuzione iniziale si aspetta che A valga '1' prima di riattivare il processo.
Se A non vale mai '1', il processo rimane sospeso all'infinito

```
Somma:   PROCESS
  BEGIN
     Sum <= A XOR B XOR Cin;
   WAIT FOR 100 ns;
END PROCESS Summation;
```

Dopo la prima esecuzioneil processo si riattiva dopo 100 ns all'infinito, anche se A, B, Cin non cambiano