Modello sequenziale: i Process

Modello sequenziale

```
ENTITY ffsr IS
   Port ( s,r: IN std_logic;
          q, qn: OUT std_logic);
    END ffsr;

ARCHITECTURE seq OF ffsr IS
BEGIN
 PROCESS(s, r)
  BEGIN
   IF s = '1' AND r = '0' THEN
   q <= '1' AFTER 2 ns;
   qn <= '0' AFTER 4 ns;
   ELSIF s = '0' AND r = '1' THEN
   qn <= '1' AFTER 2 ns;
   q <= '0' AFTER 4 ns;
   ELSIF s = '1' AND r = '1' THEN
    q <= '0' AFTER 2 ns;
    qn <= '0' AFTER 2 ns;;
   END IF;
END PROCESS;
END seq;
```

Process

```
[label:] PROCESS (sensitivity list)
         sezione_dichiarativa_process
      BEGIN
      sequenza_istruzioni
      END PROCESS [label];
```

- **Un process contiene istruzioni eseguite in modo sequenziale.**
- **E' definito all'interno dell'ARCHITECTURE.**
- **Più process vengono eseguiti in modo concorrente.**
- **L'esecuzione del process è determinata da un evento su uno dei segnali della sensitivity list.**

Process

- I process comunicano tra loro attraverso i signal.

```
ARCHITECTURE Behavioral OF Esempio IS
signal c : std_logic;

pa: PROCESS (a,b)
       BEGIN
         c <= a  AND b;
    END PROCESS  pa;

pb:  PROCESS (a,c)
       BEGIN
         d <= NOT c AND a;
       END PROCESS  pb;
END Behavioral
```

Variabili

```
VARIABLE nome_variabile :tipo_variabile [:=valore_iniziale];
```

Es.

```
VARIABLE a : integer range 0 to 15;
```

Le variabili possono essere definite in un process e utilizzate solo nello stesso process.

Assegnamento ad una variabile:

```
a := espressione;
```

A differenza dei signal, l'assegnamento del nuovo valore ad una variabile avviene immediatamente.

Le variabili conservano il loro valore fino alla successiva esecuzione del process.

Istruzione IF

```
IF condizione THEN
   sequenza_di_istruzioni
[ELSIF  condizione THEN
   sequenza_di_istruzioni]
....
[ELSE
   sequenza_di_istruzioni]
END IF;
```

ELSIF può essere ripetuto più di una volta.

ELSE può apparire una sola volta.

Condizione è in tutti i casi un'espressione booleana

Multiplexer 2x1

```
ENTITY mux2x1 IS
PORT ( a, b, sel : IN std_logic;
   q :OUT std_logic);
END mux2x1;
ARCHITECTURE sequenziale OF mux2x1 IS
BEGIN
  PROCESS(a,b,sel)
   BEGIN
    IF sel = '0' THEN q <= a;
      ELSIF sel='1' THEN q<= b;
      ELSE q <= 'X';
    END IF;
  END PROCESS;
END sequenziale;
```

FFD

```
ENTITY ffd IS
PORT ( rst,d, clk: IN std_logic;
    q: OUT std_logic );
END ffd;

ARCHITECTURE behavioral OF ffd IS
BEGIN
 PROCESS(CLK,RST)
  BEGIN
  IF RST='1' THEN q <= '0'    ;
  ELSIF clk='0' AND clk'EVENT AND clk'LAST_VALUE='1'
      THEN q<= d AFTER 2ns;
   END IF;
  END PROCESS;
END behavioral;
```

Istruzione CASE

```
CASE espressione IS
  WHEN valore_1 =>
      sequenza_di_istruzioni
  WHEN valore_2 | valore_3 =>
       sequenza_di_istruzioni
  WHEN valore_4 to valore_N =>
      sequenza_di_istruzioni
  WHEN OTHERS  =>
      sequenza_di_istruzioni
END CASE;
```

Selettore

```
ENTITY selettore IS
 PORT (a0,a1,a2,a3 : IN std_logic;
       sel : IN integer range 0 to 7;
   q: OUT std_logic);
 END selettore;

ARCHITECTURE sequenziale OF selettore IS
   BEGIN
   PROCESS(a0,a1,a2,a3,sel)
    BEGIN
    CASE sel IS
        WHEN 0     => q <= a0;
        WHEN 1 | 3 | 5    => q <= a1;
        WHEN 6 to 7       => q <= a2;
        WHEN OTHERS    => q <= a3;
    END CASE;
   END PROCESS;
END Behavioral;
```

Istruzione WHILE

```
[nome_label:]     WHILE condizione LOOP
                    sequenza_di_istruzioni
                  END LOOP [nome_label] ;
```

Esempio While

```
ENTITY confronta IS
   PORT (a,b : IN std_logic_vector(7 downto 0);
      ris: OUT std_logic);
END confronta;

ARCHITECTURE sequenziale OF confronta IS
BEGIN
 PROCESS(a,b)
   VARIABLE ind: integer;
   VARIABLE uguale: std_logic;
    BEGIN
     ind:=0; uguale:='1';
     WHILE (uguale='1' AND ind<=7) LOOP
       IF a(ind) /= b(ind) THEN uguale := '0';
        END IF;
       ind := ind +1;
     END LOOP;
     ris <= uguale;
 END PROCESS;
END sequenziale;
```

Istruzione FOR

```
[nome_label :]
FOR identificatore IN intervallo_discreto LOOP
   sequenza_di_istruzioni
END LOOP  [nome_label] ;
```

L'identificatore :

- non deve essere dichiarato;
- può essere utilizzato solo in lettura;
- non è visibile fuori dal FOR.

Conta del numero di '1' in un vettore

```
ENTITY conta1 IS
PORT (    a : IN std_logic_vector(7 downto 0);
      b : OUT integer range 0 to 8 );
END conta1;

ARCHITECTURE seq OF conta1 IS
  BEGIN
PROCESS(a)
   VARIABLE conta : integer range 0 to 8;
  BEGIN
   conta := 0;
   FOR i IN 0 to 7 LOOP
     IF a(i) = '1' THEN conta := conta +1;
     END IF;
   END LOOP;
   b <= conta;
END PROCESS;
END seq;
```

Generate

- Utilizzato per la creazione di schematici con struttura regolare
- Un GENERATE può includere altri GENERATE

```
name: FOR i IN 0 TO d-1 GENERATE
       --concurrent statements
       END GENERATE name
```

Esempio

```
AND32: For i in 0 to 31 generate
      AI: and2 port map (A(i), B(i), C(i));
      end generate AND32;
```

Generate: if scheme

Permette la creazione condizionata di componenti

```
name: IF condizione GENERATE
          --concurrent-statements
        END GENERATE name;
```

Non può essere usato ELSE o ELSIF

Esempio

```
GA: for i in 0 to d-1 generate
     GA0: if i=0 generate
            a0: adder1 port map (A(0),B(0),'0',C(0),R(0));
          end generate GA0;

     GAI: if i>0 generate
           ai: adder1 port map (A(i),B(i),C(i-1),C(i), R(i));
         end generate GAI;
end generate GA;
```

Istruzione ASSERT

E' usata per fornire una stringa al progettista durante la simulazione

```
ASSERT condizione
[REPORT stringa]
[SEVERITY livello_di_severity]
```

La stringa viene visualizzata se la condizione è falsa.

Livello di severity:

- note
- warning
- error (default)
- failure

Esempio Assert

```
PROCESS (b,c)
 BEGIN
b<=c;
   ASSERT (c/=a)
   REPORT "Il Valore di C è uguale a quello di a"
   SEVERITY ERROR;
END PROCESS;
```

Istruzione WAIT

- Questa istruzione sospende l'esecuzione di un process o sottoprogramma.
- L'esecuzione può riprendere quando si verifica una certa condizione.
- Abbiamo tre possibili modi di esprimere questa condizione.
  - - WAIT ON elenco_segnali
  - - WAIT UNTIL condizione
  - - WAIT FOR tempo
- Con WAIT ON l'esecuzione riprende se c'è un evento su uno dei segnali elencati.
- Con WAIT UNTIL l'esecuzione riprende quando la condizione diventa vera.
- Con WAIT FOR l'esecuzione riprende dopo un tempo fissato.
- In un process non è possibile avere contemporaneamente sensitivity list e WAIT.

Esempi con WAIT

```
PROCESS
 BEGIN
   WAIT ON a;
   IF a='1' THEN
     b<= "10";
   ELSE b <= "01";
   END IF:
   WAIT ON c,d;
   F <= c AND d;
END PROCESS;
```

```
PROCESS
 BEGIN
WAIT UNTIL a='1';
c <=  d;
WAIT UNTIL a='0';
c <= f;
END PROCESS;
```

```
PROCESS
 BEGIN
z <= a;
WAIT FOR 20 ns;
Z <= b;
WAIT UNTIL c='1' FOR 40    ns;
assert (c='1)
report "time out violation"
severity error
z <= c;
WAIT ON a FOR 60 ns;
END PROCESS;
```

WAIT ON

Un process con sensitivity list può essere descritto in modo equivalente con un process senza sensitivity list e con un WAIT ON alla fine contenente gli stessi segnali della sensitivity list

```
PROCESS
BEGIN
  IF sel = '1' THEN
     Z<= A;
  ELSE
     Z<= B;
  END IF;
  WAIT ON A,B,SEL;
END PROCESS
```

```
PROCESS(A,B,SEL)
BEGIN
  IF sel = '1' THEN
     Z<= A;
  ELSE
     Z<= B;
  END IF;
END PROCESS
```

WAIT: costrutti non sintetizzabili

```
process
 begin
  wait on a;
  d<= di;
 end process;
```

- -- non è sintetizzabile poiché non è presente un edge

```
process
 begin
  wait for 10 ns;
  d<= di;
 end process;
```

- -- non è sintetizzabile poiché non è presente un edge

WAIT: costrutti non sintetizzabili

```
process
 begin
  wait until c'event and c=1 ;
  d<= d1;
  wait until d'event and d=1 ;
  d<=d2;
 end process;
```

- -- non è sintetizzabile poiché tutti gli edge devono dipendere dallo stesso segnale

WAIT: costrutti non sintetizzabili

```
process
 begin
  wait until c'event and c='1' ;
  d<= d1;
  wait until c'event and c='0' ;
  d<=d2;
 end process;
```

-- non è sintetizzabile poiché i wait devono contenere un solo edge

WAIT: costrutti non sintetizzabili

```
process
 begin
  wait until d'event and c='1' ;
  d<= d1;
  wait until c'event and c='0' ;
  d<=d2;
 end process;
```

-- non è sintetizzabile poiché i wait devono contenere un solo edge

WAIT: costrutti non sintetizzabili

```
process
 begin
  wait until c'event;
  d<= d1;
  wait until c'event;
  d<=d2;
 end process;
```

-- non è sintetizzabile poiché non può essere determinata la polarità dell'edge

WAIT: costrutti non sintetizzabili

```
entity es is
  port (a,c : in integer;
          d : out integer);
end es;
architecture beh of es is
 begin
   process
    begin
        wait until c'event and c= 2;
        d<= a;
   end process;
end beh;
```

- Non è sintetizzabile perché il clock deve essere di un solo bit