DEIS
Dipartimento di Elettronica Informatica e Sistemistica
Universita' di Bologna

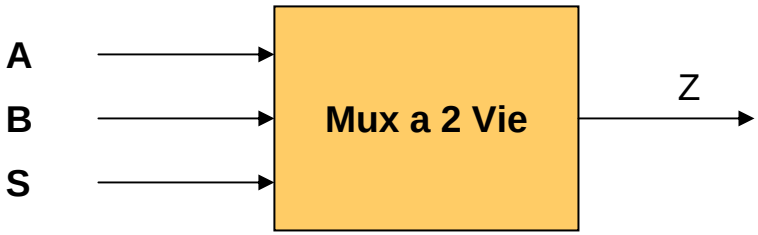
Introduzione al linguaggio VHDL

PARTE II

Stefano Mattoccia
e-mail: smattoccia@deis.unibo.it
Telefono: +39 051 2093860

Bologna, 4 Marzo 2004

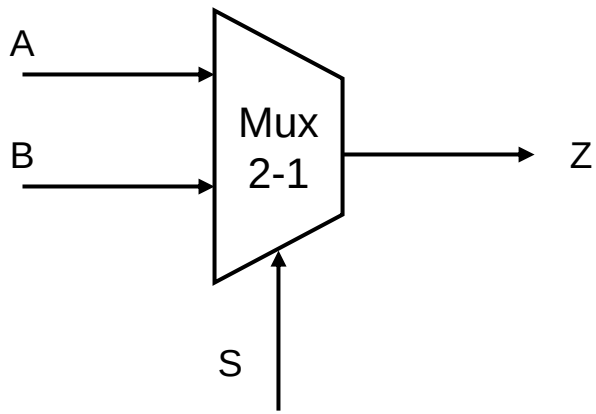
Consideriamo una semplice rete combinatoria: un Multiplexer a 2 vie



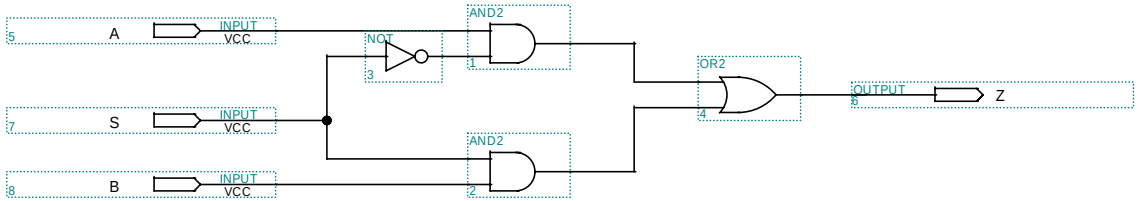
$$Z = \begin{cases} A \text{ se } S = \text{'0'} \\ B \text{ se } S = \text{'1'} \end{cases}$$

Logicamente:

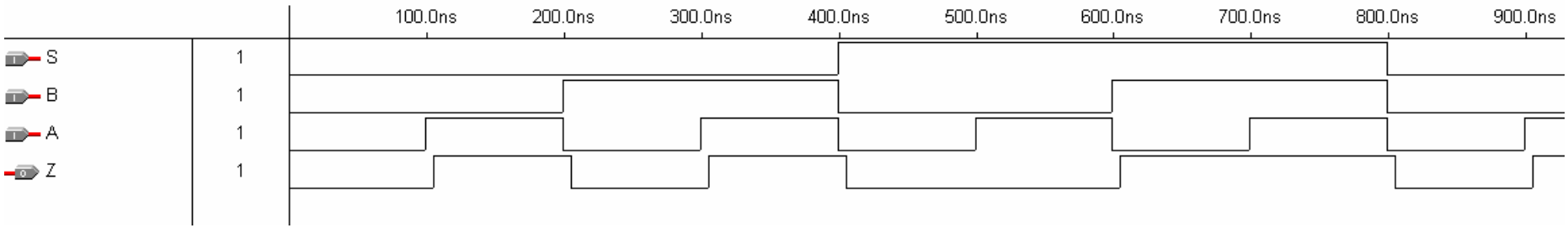
Z = A•S' + B•S



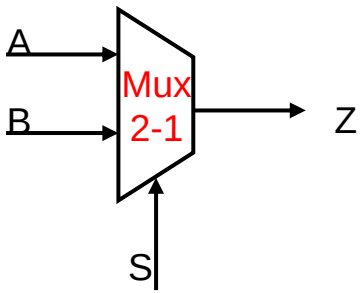
Design entry mediante schema logico (.gdf)



Risultato della simulazione

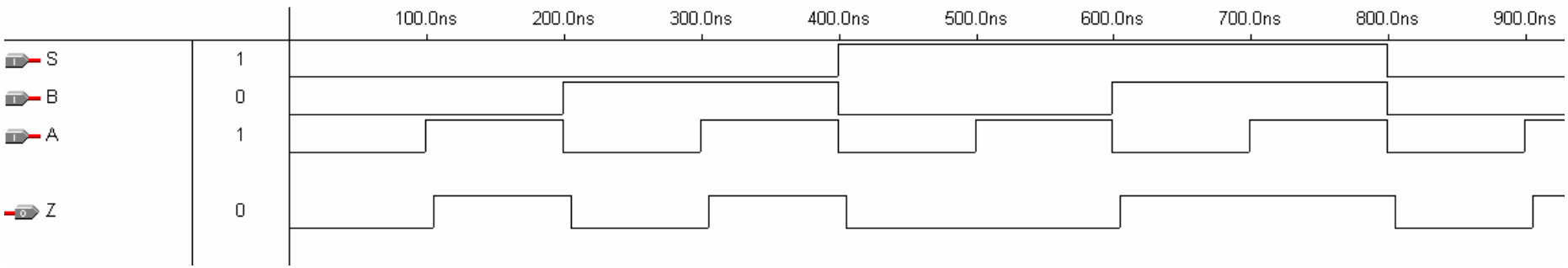


Codice VHDL che sintetizza il multiplexer a 2 vie



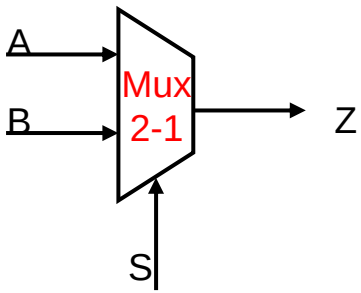
esempio1.vhd

```
-- esempio 1 (Multiplexer a 2 vie)

ENTITY esempio1 IS

        PORT (A,B,S : IN  BIT;
                    Z : OUT BIT);

END esempio1;

ARCHITECTURE arch_esempio1 OF esempio1 IS

BEGIN

        Z<=(A AND (NOT S)) OR (B AND S);

END  arch_esempio1;
```



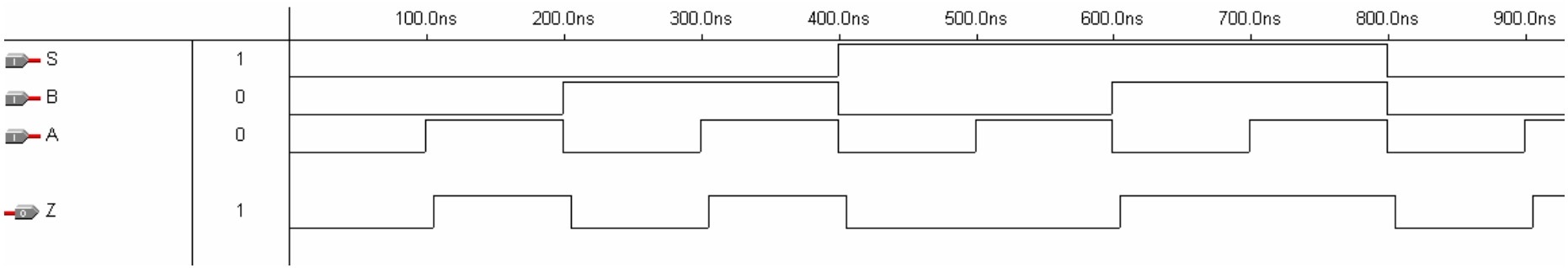
Oppure...

Le tre assegnazioni vengono eseguite in modo paralello (concorrente)

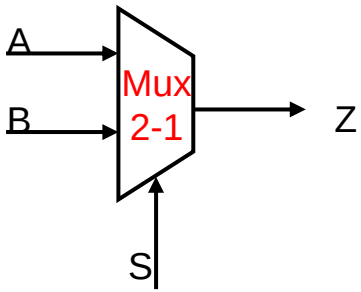


esempio2.vhd

```
-- esempio 2 (Multiplexer a 2 vie)

ENTITY esempio2 IS
        PORT (A,B,S : IN  BIT;
                    Z : OUT BIT);
END esempio2;

ARCHITECTURE arch_esempio2 OF esempio2 IS
SIGNAL TA,TB: BIT;

BEGIN
        TA<=A AND (NOT S);
        TB<=B AND S;

        Z<=TA OR TB;

END  arch_esempio2;
```

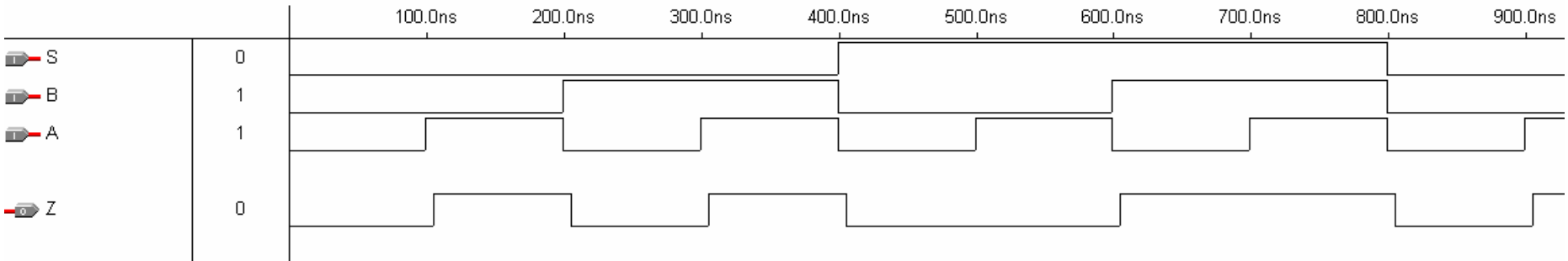


Il risultato e' indipendente dall'ordine con il quale vengono scritte nel codice le 3 assegnazioni (esecuzione parallela). Infatti...

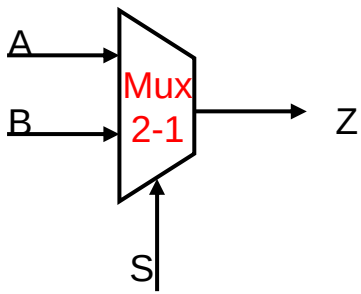


esempio3.vhd

```
-- esempio 3 (Multiplexer a 2 vie)

ENTITY esempio3 IS
        PORT (A,B,S : IN  BIT;
                    Z : OUT BIT);
END esempio3;

ARCHITECTURE arch_esempio3 OF esempio3 IS
SIGNAL TA,TB: BIT;

BEGIN
        Z<=TA OR TB;

        TA<=A AND (NOT S);
        TB<=B AND S;

END  arch_esempio3;
```

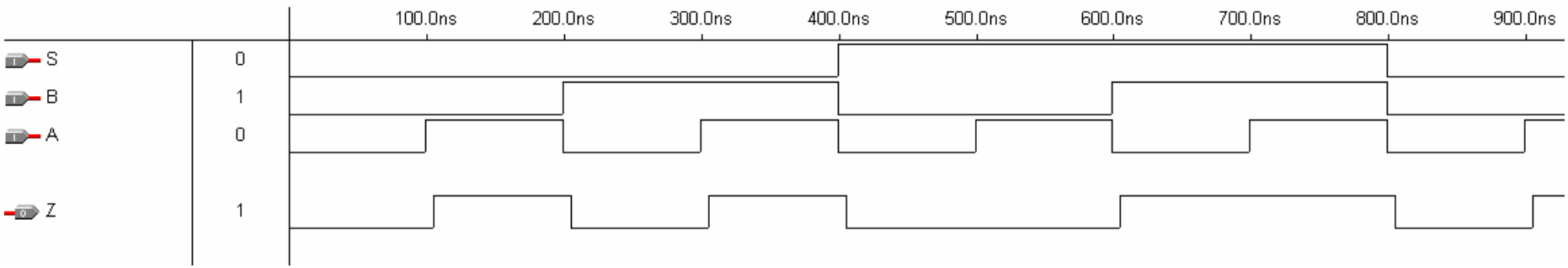


Il codice del mux a 2 con VHDL utilizzando WHEN-ELSE



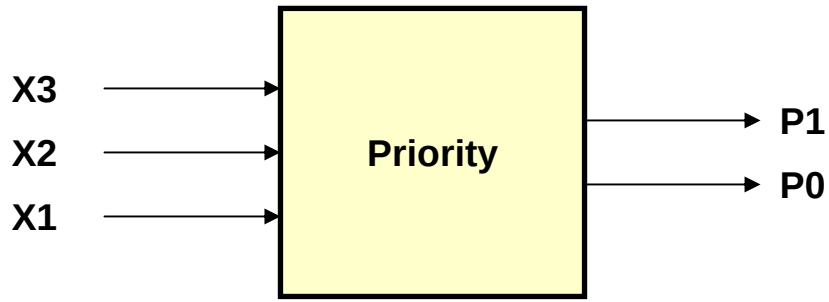

```
-- esempio 4 (Multiplexer a 2 vie) con WHEN-ELSE


ENTITY esempio3 IS
        PORT (A,B,S : IN  BIT;
                    Z : OUT BIT);
END esempio3;

ARCHITECTURE arch_esempio3 OF esempio3 IS

BEGIN

        Z<=A WHEN S='0' ELSE B;

END  arch_esempio3;
```

esempio4.vhd



Rete combinatoria

Rilevatore di priorità (Priority Encoder)

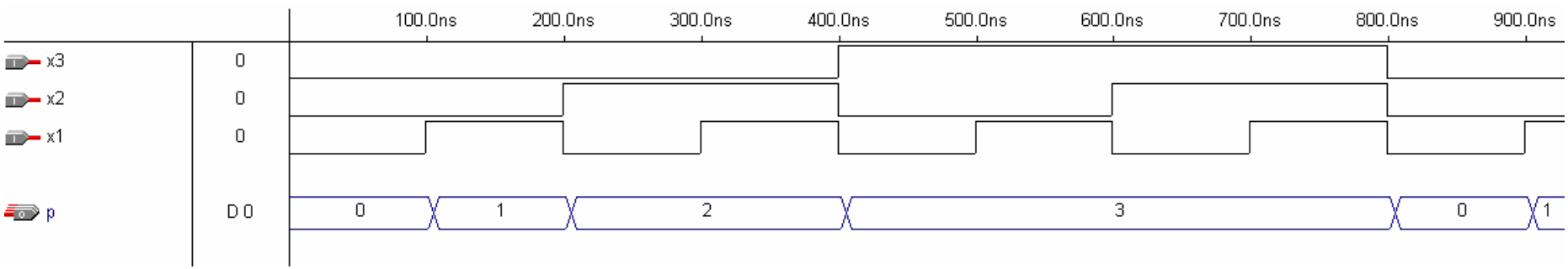


$$P1\ P0 = \begin{cases} 11 & \text{se } x3 = \text{‘1’ e } x2 = \text{‘-’ e } x1 = \text{‘-’} \\ 10 & \text{se } x3 = \text{‘0’ e } x2 = \text{‘1’ e } x1 = \text{‘-’} \\ 01 & \text{se } x3 = \text{‘0’ e } x2 = \text{‘0’ e } x1 = \text{‘1’} \\ 00 & \text{se } x3 = \text{‘0’ e } x2 = \text{‘0’ e } x1 = \text{‘0’} \end{cases}$$

Priority 1

priority1.vhd

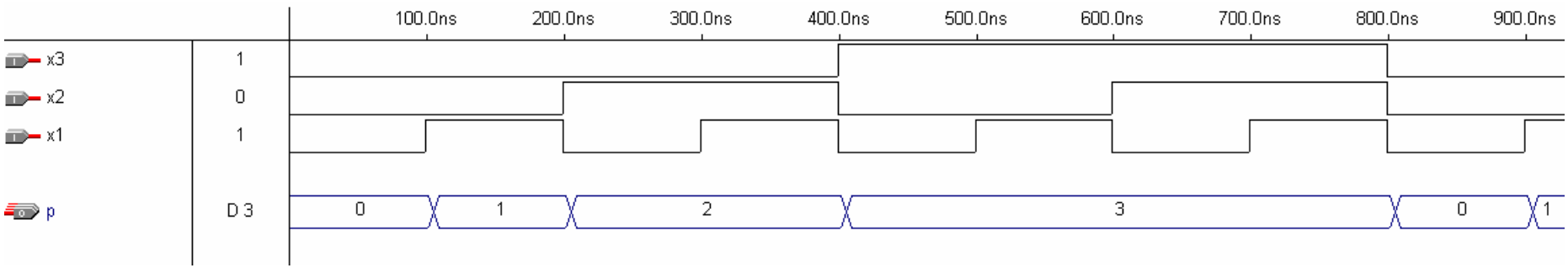
```
ENTITY priority1 IS
   PORT (x3,x2,x1 : IN  BIT;
                 p : OUT BIT_vector(1 DOWNTO 0));
END priority1;

ARCHITECTURE arch_priority1 OF priority1 IS

BEGIN

p<="00" WHEN ((x3='0') AND (x2='0') AND (x1='0')) ELSE
   "01" WHEN ((x3='0') AND (x2='0') AND (x1='1')) ELSE
   "10" WHEN ((x3='0') AND (x2='1') AND ((x1='1') OR (X1='0'))) ELSE
   "11" WHEN ((x3='1') AND ((x2='1') OR (x2='0')) AND ((x1='1') OR (X1='0')));


END arch_priority1;
```



Priority 2

priority2.vhd

```
ENTITY priority2 IS
   PORT (x3,x2,x1 : IN  BIT;
                 p : OUT BIT_vector(1 DOWNTO 0));
END priority2;

ARCHITECTURE arch_priority2 OF priority2 IS

BEGIN

p<=”11" WHEN x3=’1' ELSE
   ”10" WHEN x2=’1' ELSE
   “01” WHEN x1=’1' ELSE
   ”00”;

END arch_priority2;
```

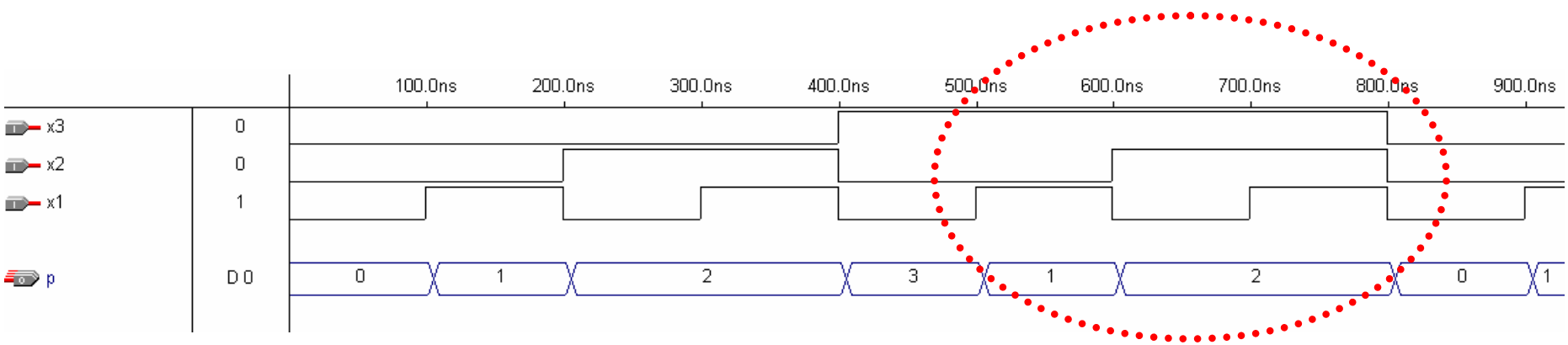


Priority2Wrong (**ERRATO**)

E’ importante l’ordine con il quale vengono eseguiti i test WHEN-ELSE !

priority2wrong.vhd

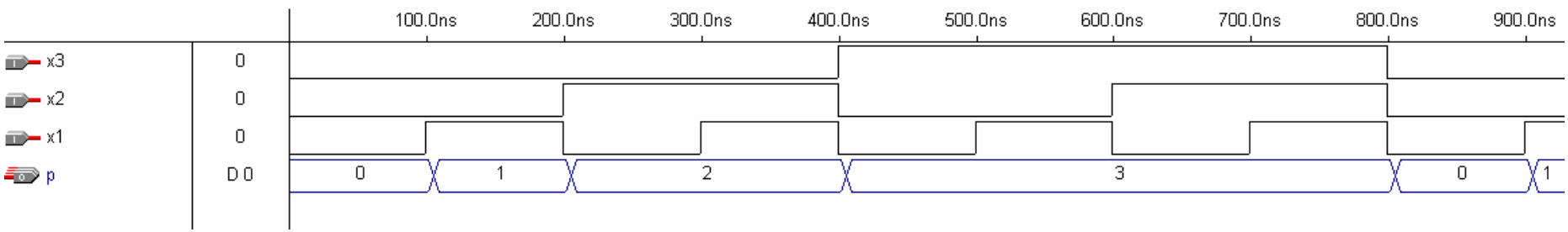
```
ENTITY priority2wrong IS
   PORT (x3,x2,x1 : IN  BIT;
                p : OUT BIT_vector(1 DOWNTO 0));
END priority2wrong;

ARCHITECTURE arch_priority2wrong OF priority2wrong IS

BEGIN

p<=”10" WHEN x2=’1' ELSE
   ”01" WHEN x1=’1' ELSE
   “11” WHEN x3=’1' ELSE
   ”00”;

END arch_priority2wrong;
```



Priority 4

priority4.vhd

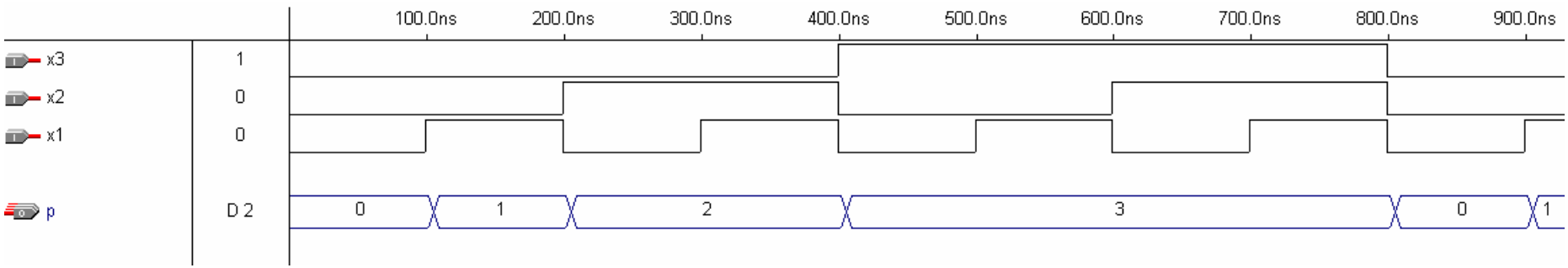
```
ENTITY priority2 IS
   PORT (       X : IN  BIT_vector(3 DOWNTO 1);
                p : OUT BIT_vector(1 DOWNTO 0));
END priority2;

ARCHITECTURE arch_priority2 OF priority2 IS

BEGIN

WITH X SELECT

P <= "00" WHEN "000",
          "01" WHEN "001",
          "10" WHEN "010",
          "10" WHEN "011",
          "11" WHEN OTHERS;


END arch_priority2;
```



PriorityProcess1

priorityprocess1.vhd

```
ENTITY priorityprocess1 IS
   PORT (x3,x2,x1 : IN  BIT;
                 p : OUT BIT_vector(1 DOWNTO 0));
END priorityprocess1;

ARCHITECTURE arch_priorityprocess1 OF priorityprocess1 IS

BEGIN
        PROCESS(x3,x2,x1) -- sensitivity list
                BEGIN

                IF    (x3='1') THEN p<="11";
                ELSIF (x2='1') THEN p<="10";
                ELSIF (x1='1') THEN p<="01";
                ELSE   p<="00";
                END IF;

        END PROCESS;

END arch_priorityprocess1;
```



SEQUENTIAL STATEMENTS

Sequential statements possono essere utilizzati solo all'interno di processi (PROCESS), funzioni (FUNCTION) e procedure (PROCEDURE).
L'assegnamento dei segnali all'interno di un processo e' sequenziale e l'ordine con il quale vengono effettuati gli assegnamenti influenza la logica di funzionamento.
L'insieme delle istruzioni (statements) che costituiscono un processo costituiscono un *concurrent statement*.

- **IF-THEN-ELSE**
- **CASE-WHEN**
- **LOOP**
- **WAIT**

IF-THEN-ELSE

```
IF <condition> THEN
                <istruzione1>;
                ELSE
                <istruzione2>;
END IF;
```

Il costrutto **IF-THEN-ELSE** e' utilizzato per eseguire un set di istruzioni selezionate sulla base del valore di una espressione booleana (***<condition>*** **).** Ad esempio:

```
processo1: PROCESS (a,b,c)
BEGIN
          IF a='1' THEN
                    z='1';
                    ELSE
                    z='0';
          END IF;
END PROCESS;
```

IF-THEN-ELSE-ELSIF

```
IF <condition1> THEN
                <itruzione1>;
                ELSIF <condition2> THEN
                <istruzione2>;
                ELSE
                <istruzione3>;
END IF;
```

Il costrutto **IF-THEN-ELSE** può essere espanso ulteriormente per consentire la valutazione di più condizioni attraverso l'utilizzo di **ELSIF**.

Si noti come l'esecuzione di ***<instruzione3>*** sia subordinata alla mancata verifica delle condizioni ***<condition1>*** e ***<condition2>***.

CASE-WHEN

```
CASE <selection_signal> IS
                WHEN <value_1_selection_signal> => <istruzione_1>;
                WHEN <value_2_selection_signal> => <istruzione_2>;
                WHEN <value_3_selection_signal> => <istruzione_3>;
                ..      ..      ..
                ..      ..      ..
                WHEN <value_n_selection_signal> => <istruzione_n>;
                WHEN OTHERS => <istruzione_others>;
END CASE;
```

Il costrutto **CASE-WHEN** e' utilizzato per eseguire un set di istruzioni selezionate sulla base del valore di uno specifico SEGNALE ***<selection_signal>***.

La parola chiave **OTHERS** e' utilizzata per eseguire il set di istruzioni ***<instruzione_others>*** nel caso nessuna condizione sia verificata.

LOOP

```
WHILE (<condition>) LOOP
        <instruction>;
END LOOP;
```

WHILE-LOOP

```
FOR <var> IN <min_value> TO <max_value> LOOP
        <instruction>;
END LOOP;
```

FOR-LOOP (0FORWARD)

```
FOR <var> IN <max_value> DOWNTO <min_value> LOOP
        <instruction>;
END LOOP;
```

FOR-LOOP (BACKWARD)

All’interno di un ciclo è possibile interrompere l’iterazione corrente e passare a quella successiva utilizzando l’istruzione **NEXT**.
Un ciclo può essere completamente interrotto attraverso l’istruzione **EXIT**.

Segnali VS Variabili

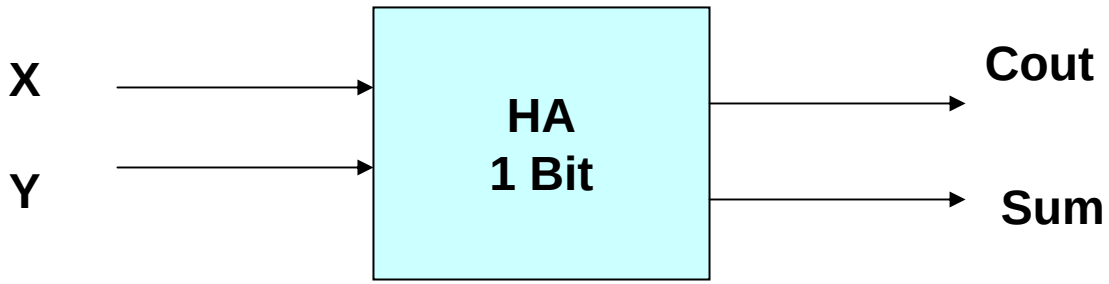
Esiste una sostanziale differenza su come viene effettuato l'assegnamento di un valore ad un segnale e ad una variabile all'interno di un processo (esecuzione sequenziale).

I valori assegnati alle variabili hanno effetto istantaneamente mentre i valori assegnati ai segnali vengono aggiornati solo all'uscita del processo.

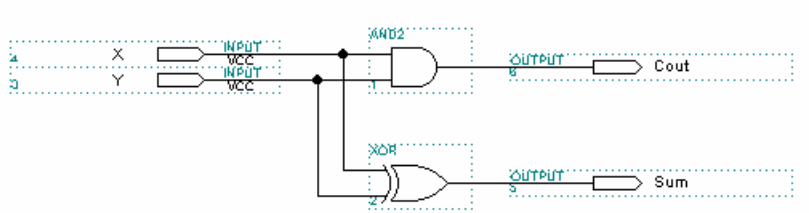
Half-Adder a 1 bit

Progettare una rete che esegua la somma di due numeri di 1 bit:

S = X + Y



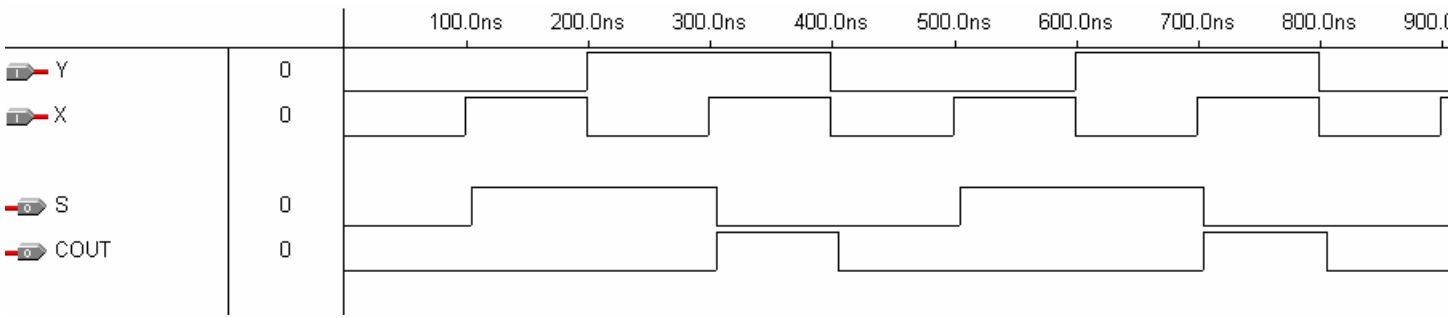
Y	X	Cout	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



ha_base.vhd

```
ENTITY ha IS
   PORT (X,Y     : IN   BIT;
         S,COUT  : OUT  BIT);
END ha;

ARCHITECTURE arch_ha OF ha IS

BEGIN
-- Somma
S<=X XOR Y;
-- Carry
COUT<= A AND Y;

END arch_ha;
```

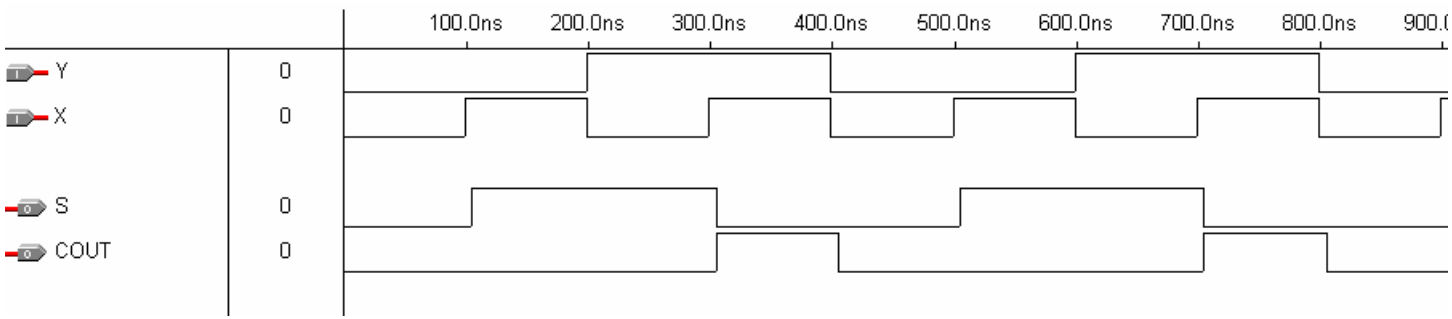
Half Adder progettato utilizzando l'esecuzione concorrente



ha_base1.vhd

```
ENTITY ha_base1 IS
   PORT (X,Y    : IN   BIT;
         S,COUT : OUT  BIT);
END ha_base1;

ARCHITECTURE arch_ha_base1 OF ha_base1 IS

BEGIN
          PROCESS(X,Y)
          VARIABLE ST,CT :BIT;
          BEGIN
                    -- Somma
                    ST:= X XOR Y;
                    -- Carry
                    CT:= X AND Y;

                    -- assegna valori ai segnali S e COUT
                    S<=ST;
                    COUT<= CT;
          END PROCESS;
END arch_ha_base1;
```

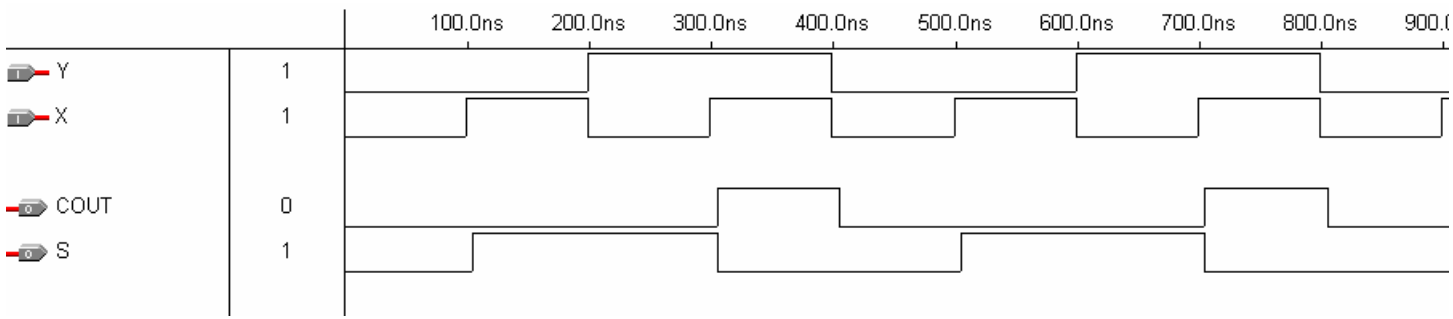
Half Adder progettato utilizzando l'esecuzione sequenziale (PROCESS) e le variabili



ha.vhd

```
ENTITY ha IS
   PORT (X,Y     : IN   BIT;
         S,COUT  : OUT  BIT);
END ha;

ARCHITECTURE arch_ha OF ha IS

BEGIN
          PROCESS(X,Y)
          BEGIN

                    IF X /= Y
                              THEN S<='1';
                              ELSE S<='0';
                    END IF;

                    IF ( X='1' AND Y='1' )
                              THEN COUT<='1';
                              ELSE COUT<='0';
                    END IF;
          END PROCESS;
END arch_ha;
```

Half Adder progettato utilizzando l'esecuzione sequenziale (PROCESS) e il costrutto IF-THEN-ELSE.



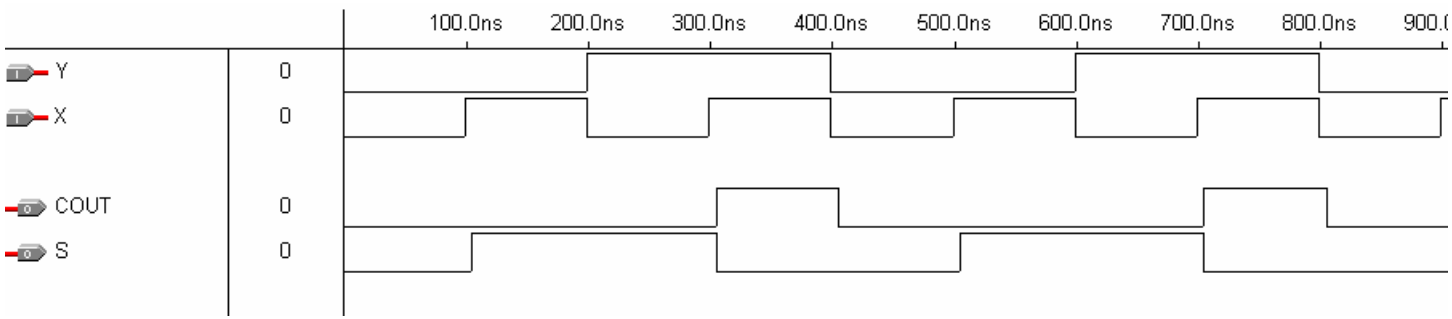
ha1.vhd

```
ENTITY ha1 IS
   PORT(X,Y     : IN   BIT;
        S,COUT  : OUT  BIT);
END ha1;

ARCHITECTURE arch_ha1 OF ha1 IS
BEGIN
    PROCESS(X,Y)
    BEGIN
          -- Somma
          IF X /= Y
               THEN S<='1';
               ELSE S<='0';
          END IF;

          -- Carry-Out
          COUT<='0';
          IF ( X='1' AND Y='1' )
                THEN COUT<='1';
          END IF;

   END PROCESS;

END arch_ha1;
```

Half Adder progettato utilizzando l'esecuzione sequenziale (PROCESS) e il costrutto IF-THEN-ELSE. Sono presenti assegnamenti multipli al segnale COUT.

Attenzione, il compilatore con questo codice genera un *warning* segnalando che esistono assegnamenti multipli al segnale COUT. Solo l'ultimo assegnamento avrà effetto. E' bene evitare queste situazioni utilizzando i costrutti del VHDL (e.g. ELSE) o segnali temporanei (VARIABLE).

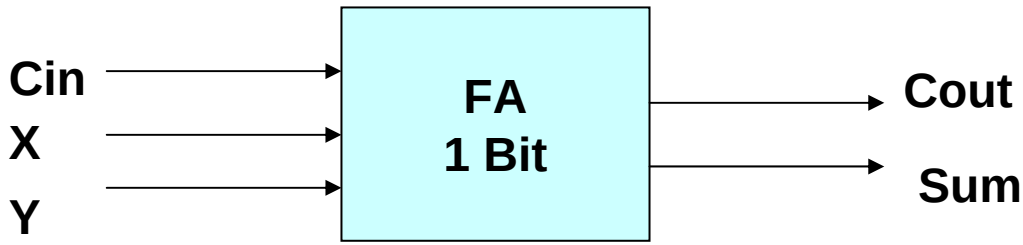
Il risultato della simulazione sul codice “*ha1.vhd*” della pagina precedente è comunque corretto, come mostrato nella figura seguente.

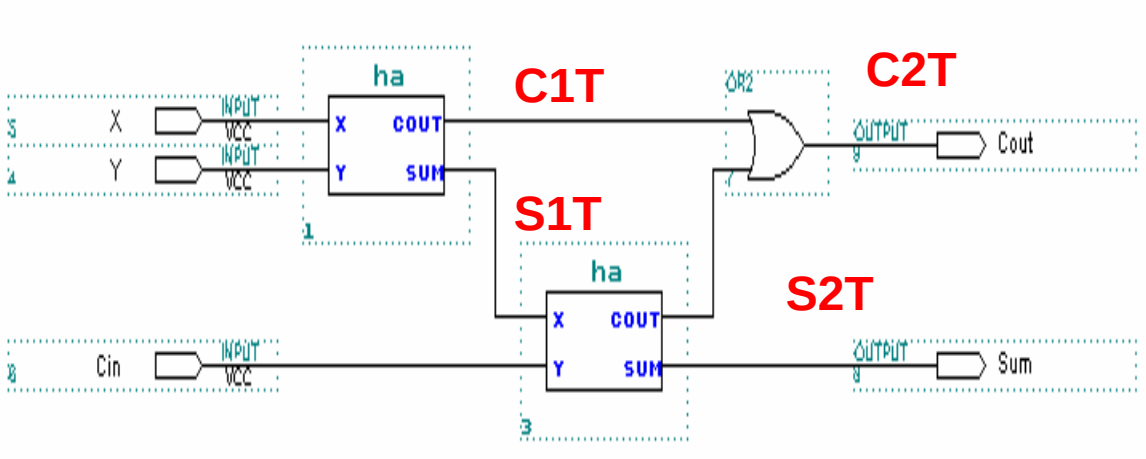


Full-Adder a 1 bit

Progettare Full Adder a 1 bit:

Sum = X + Y + Cin

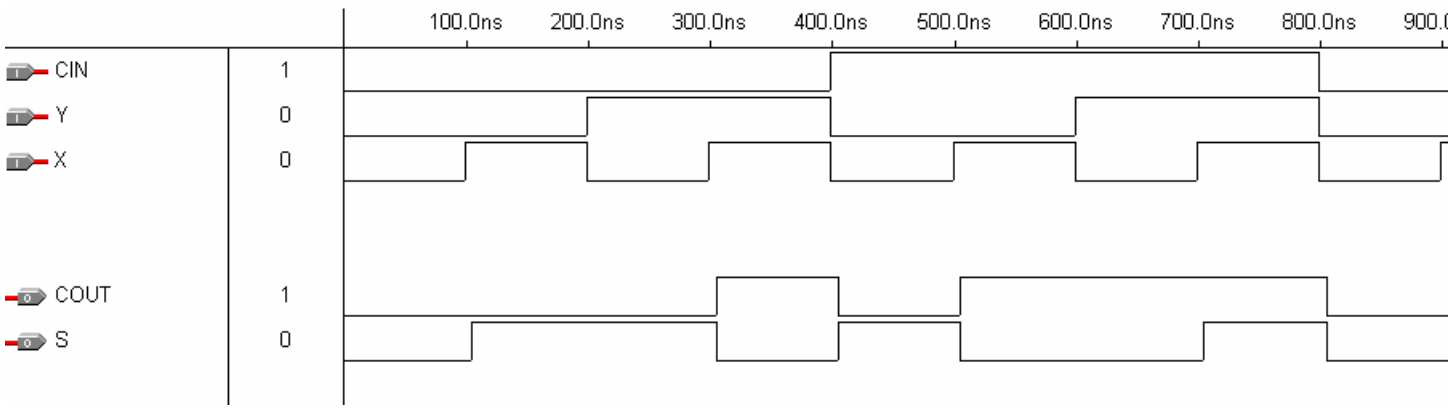




fa.vhd

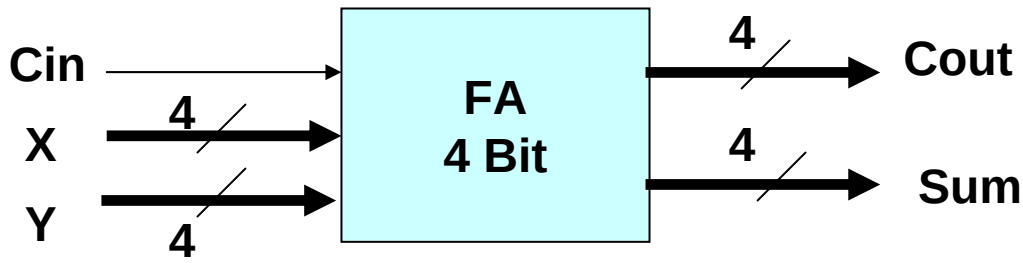
```
ENTITY fa IS
   PORT (X,Y,CIN : IN   BIT;
         S,COUT  : OUT  BIT);
END fa;

ARCHITECTURE arch_fa OF fa IS
BEGIN
          PROCESS(X,Y,CIN)
          VARIABLE S1T,S2T,C1T,C2T :BIT;
          BEGIN
                    -- Primo Ha
                    C1T:= X AND Y;
                    S1T:= X XOR Y;
                    -- Secondo Ha
                    C2T:=S1T AND CIN;
                    S2T:=S1T XOR CIN;
                    -- assegna valori ai segnali S e COUT
                    S<=S2T;
                    COUT<= C1T OR C2T;

          END PROCESS;
END arch_fa;
```

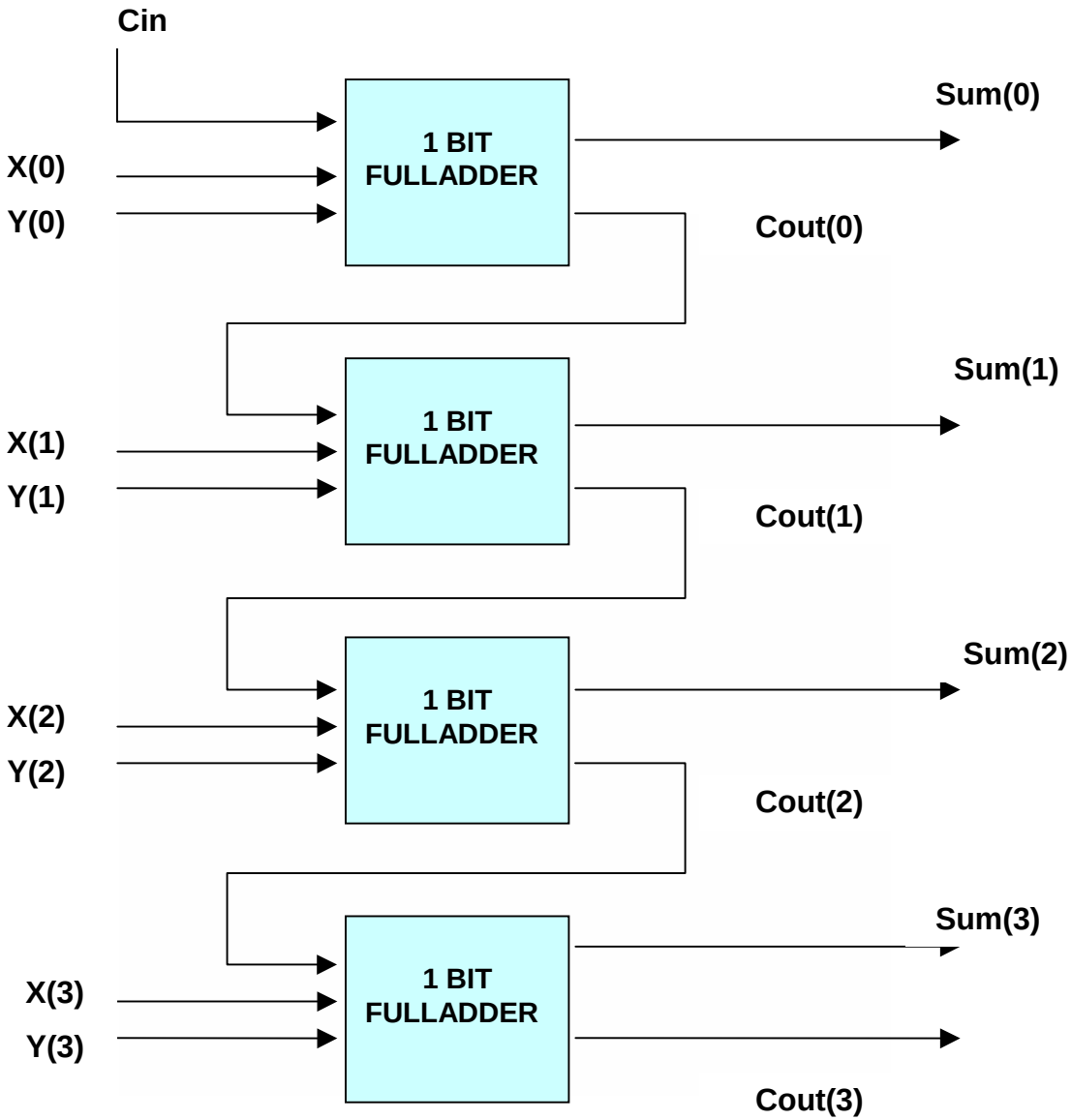
Half Adder progettato utilizzando l'esecuzione sequenziale (PROCESS) e segnali temporanei (VARIABLE).



Full-Adder a 4 bit

Progettare un Full Adder a 4 bit utilizzando 4 Full Adder a 1 bit





Una possibile sintesi (non ottimale) VHDL del sommatore a 4 bit è mostrato nel lucido seguente.

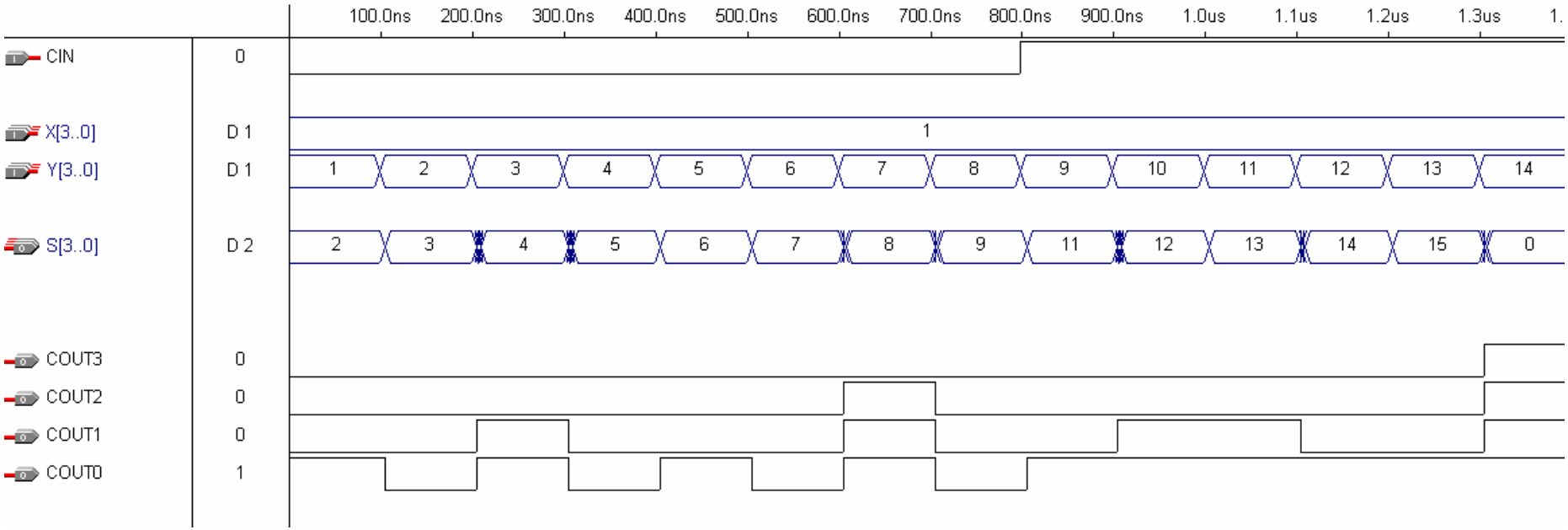
Fa_4bit_noloop.vhd

```
ENTITY fa_4bit_noloop IS
  PORT (X,Y      : IN  BIT_VECTOR(3 DOWNTO 0);
        CIN      : IN  BIT;
        S,COUT : OUT  BIT_VECTOR(3 DOWNTO 0));
END fa_4bit_noloop;

ARCHITECTURE arch_fa_4bit_noloop OF fa_4bit_noloop IS

BEGIN
        PROCESS(X,Y,CIN)

        VARIABLE TX,TY,TCOUT            :BIT_VECTOR(3 DOWNTO 0);
        VARIABLE CARRY_IN,CARRY_OUT     :BIT;
        VARIABLE S1T,S2T,C1T,C2T        :BIT;

        BEGIN

        -- inizializza variabili
        TX:=X;
        TY:=Y;

                -- ******************
                -- *** Primo FA ****
                -- ******************

                CARRY_IN:=CIN;

                -- Primo Ha
                C1T:= TX(0) AND TY(0);
                S1T:= TX(0)  XOR TY(0);

                -- Secondo Ha
                C2T:=S1T AND CARRY_IN;
                S2T:=S1T  XOR CARRY_IN;

                -- assegna valore al segnale S(0)
                S(0)<=S2T;

                -- assegna valore al segnale COUT(0)
                CARRY_OUT:=C1T OR C2T;
                COUT(0)<= CARRY_OUT;
```


Full Adder a 4 bit progettato utilizzando l'esecuzione sequenziale (PROCESS) e segnali temporanei (VARIABLE).

```
-- ******************
-- *** Secondo FA ***
-- ******************
-- riporto da precedente operazione
CARRY_IN:=CARRY_OUT;

-- Primo Ha
C1T:= TX(1) AND TY(1);
S1T:= TX(1)  XOR TY(1);

-- Secondo Ha
C2T:=S1T AND CARRY_IN;
S2T:=S1T  XOR CARRY_IN;

-- assegna valore al segnale S(1)
S(1)<=S2T;

-- assegna valore al segnale COUT(1)
CARRY_OUT:=C1T OR C2T;
COUT(1)<= CARRY_OUT;

-- ******************
-- *** Terzo FA ****
-- ******************
-- riporto da precedente operazione
CARRY_IN:=CARRY_OUT;

-- Primo Ha
C1T:= TX(2) AND TY(2);
S1T:= TX(2) XOR TY(2);

-- Secondo Ha
C2T:=S1T AND CARRY_IN;
S2T:=S1T  XOR CARRY_IN;

-- assegna valore al segnale S(2)
S(2)<=S2T;

-- assegna valore al segnale COUT(2)
CARRY_OUT:=C1T OR C2T;
COUT(2)<= CARRY_OUT;
```

```
                    -- ******************
                    -- *** Quarto FA ***
                    -- ******************

                    -- riporto da precedente operazione
                    CARRY_IN:=CARRY_OUT;

                    -- Primo Ha
                    C1T:= TX(3) AND TY(3);
                    S1T:= TX(3)  XOR TY(3);

                    -- Secondo Ha
                    C2T:=S1T AND CARRY_IN;
                    S2T:=S1T  XOR CARRY_IN;

                    -- assegna valore al segnale S(3)
                    S(3)<=S2T;

                    -- assegna valore al segnale COUT(3)
                    CARRY_OUT:=C1T OR C2T;
                    COUT(3)<= CARRY_OUT;
        END PROCESS;
END arch_fa_4bit_noloop;
```



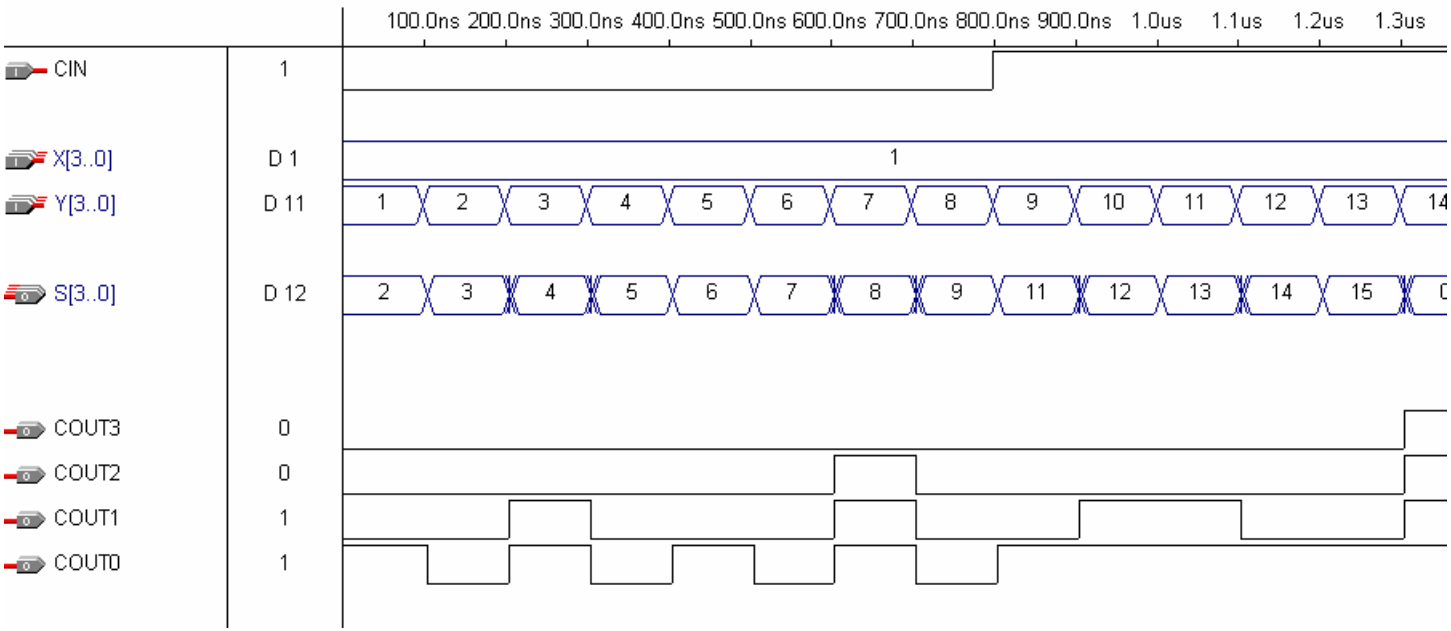
Sommatore a 4 bit: risultato della simulazione

Sfruttando la modularità del sommatore a 4 bit e utilizzando l'istruzione LOOP e' possibile scrivere una versione più compatta del codice VHDL, come mostrato nel lucido seguente.

Fa_4bit.vhd

```
ENTITY fa_4bit IS
   PORT (X,Y    : IN   BIT_VECTOR(3 DOWNTO 0);
         CIN    : IN   BIT;
         S,COUT : OUT  BIT_VECTOR(3 DOWNTO 0));
END fa_4bit;

ARCHITECTURE arch_fa_4bit OF fa_4bit IS
BEGIN
          PROCESS(X,Y,CIN)
              VARIABLE TX,TY,TCOUT         :BIT_VECTOR(3 DOWNTO 0);
              VARIABLE CARRY_IN,CARRY_OUT  :BIT;
              VARIABLE S1T,S2T,C1T,C2T     :BIT;
          BEGIN
             -- inizializza variabili prima di iniziare il loop
             TX:=X;
             TY:=Y;
          FOR i IN 0 TO 3 LOOP
                    -- seleziona il corretto carry in ingresso
                    IF (i=0) THEN
                        CARRY_IN:=CIN;
                    ELSE
                        CARRY_IN:=CARRY_OUT;
                    END IF;

                    -- Primo Ha
                    C1T:= TX(i) AND TY(i);
                    S1T:= TX(i) XOR TY(i);
                    -- Secondo Ha
                    C2T:=S1T AND CARRY_IN;
                    S2T:=S1T XOR CARRY_IN;
                    -- assegna valore al segnale S
                    S(i)<=S2T;
                    -- assegna valori al segnale COUT
                    CARRY_OUT:=C1T OR C2T;
                    COUT(i)<= CARRY_OUT;
          END LOOP;
          END PROCESS;
END arch_fa_4
```


Full Adder a 4 bit progettato utilizzando l'esecuzione sequenziale (PROCESS), segnali temporanei (VARIABLE) e l'istruzione costrutto LOOP.



Sommatore a 4 bit: risultato della simulazione

Il codice della diapositiva precedente, scritto per il Full Adder a 4 bit, si presta ad essere facilmente esteso al caso di un sommatore a N bit...