DEIS
Dipartimento di Elettronica Informatica e Sistemistica
Universita' di Bologna

Introduzione al linguaggio VHDL

PARTE III

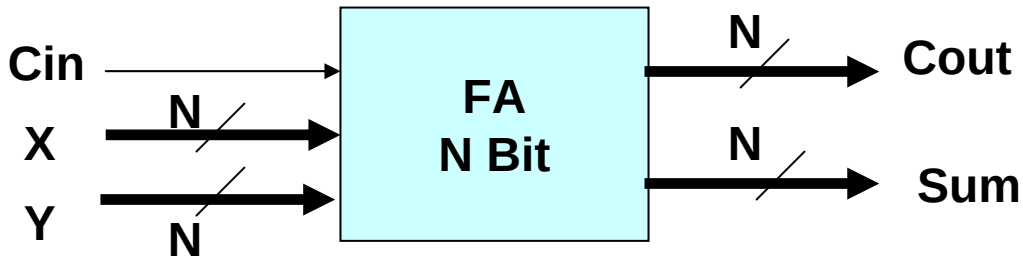
Stefano Mattoccia
e-mail: smattoccia@deis.unibo.it
Telefono: +39 051 2093860

Bologna, 4 Marzo 2004

Full-Adder a N bit

Il VHDL consente un elevato grado di astrazione.

Si supponga di dover progettare un sommatore ad N bit partendo dal codice del sommatore a 4 bit mostrato nelle pagine precedenti...



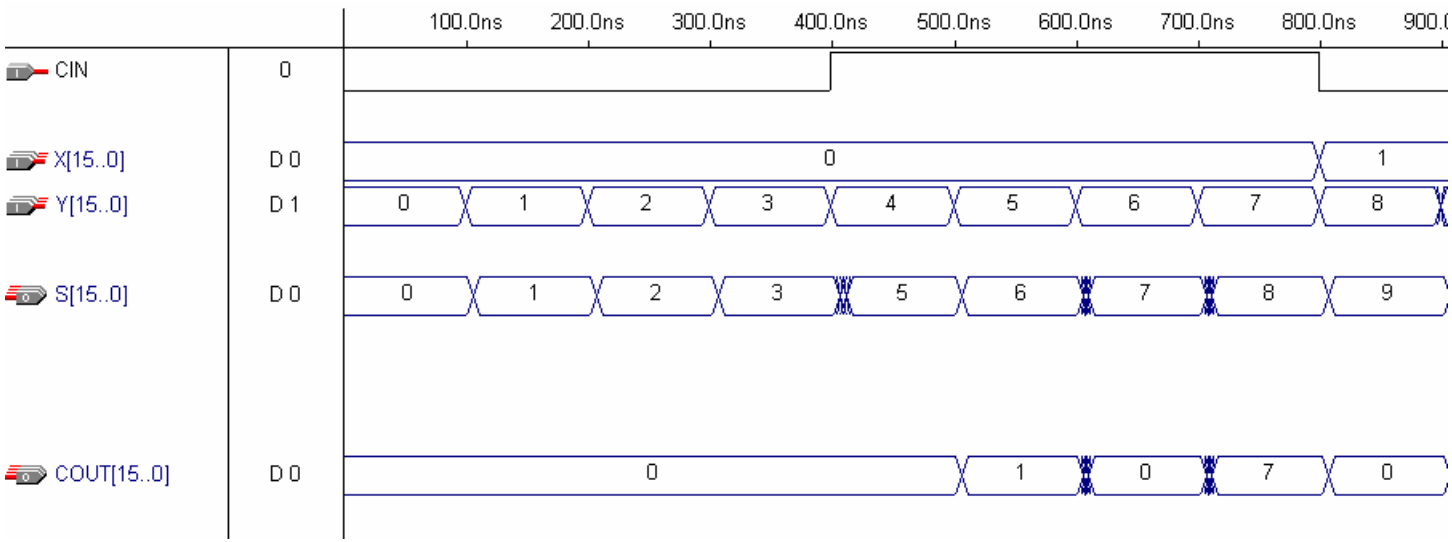
Consideriamo per esempio il caso di N=16.

Fa_Nbit.vhd

```
ENTITY fa_Nbit IS
   GENERIC(N: NATURAL :=16);
   PORT (X,Y    : IN   BIT_VECTOR(N-1 DOWNTO 0);
         CIN    : IN   BIT;
         S,COUT : OUT  BIT_VECTOR(N-1 DOWNTO 0));
END fa_Nbit;

ARCHITECTURE arch_fa_Nbit OF fa_Nbit IS
BEGIN
          PROCESS(X,Y,CIN)
                    VARIABLE TX,TY,TCOUT            :BIT_VECTOR(N-1 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 N-1 LOOP
                    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_Nbit;
```

Codice VHDL di un generico sommatore a N bit.

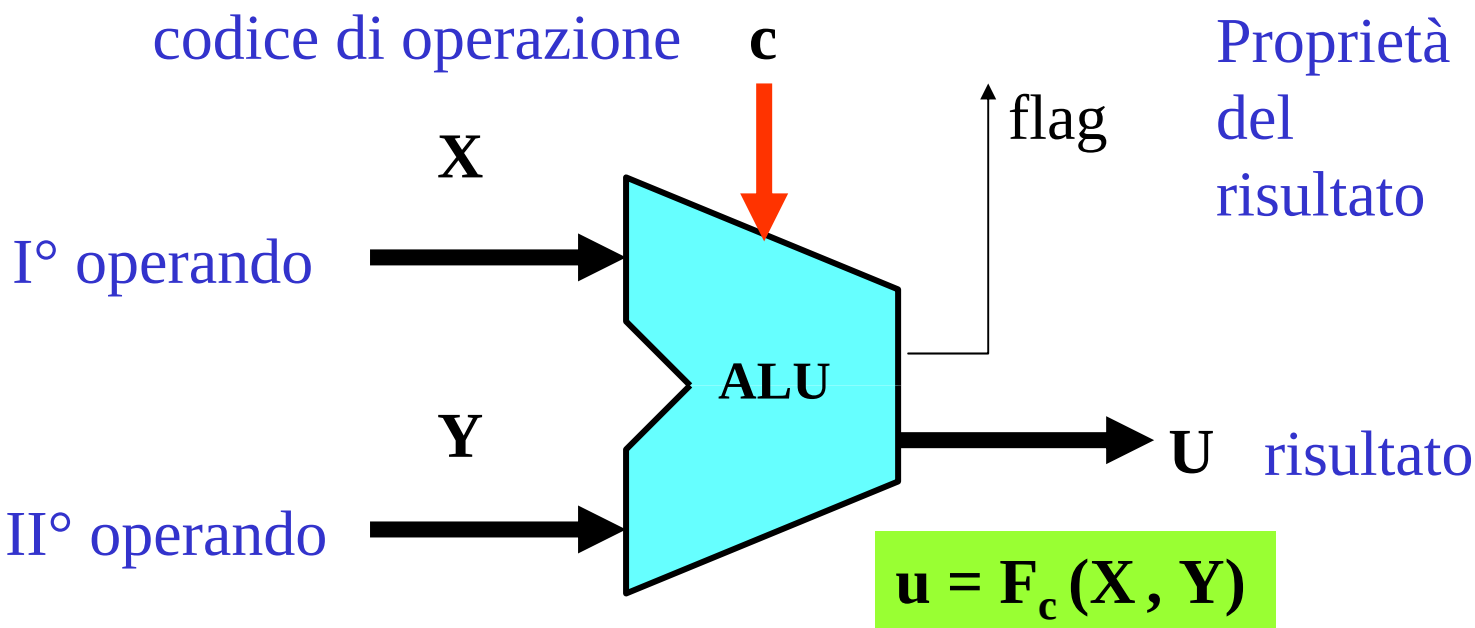
Caso specifico di N=16.



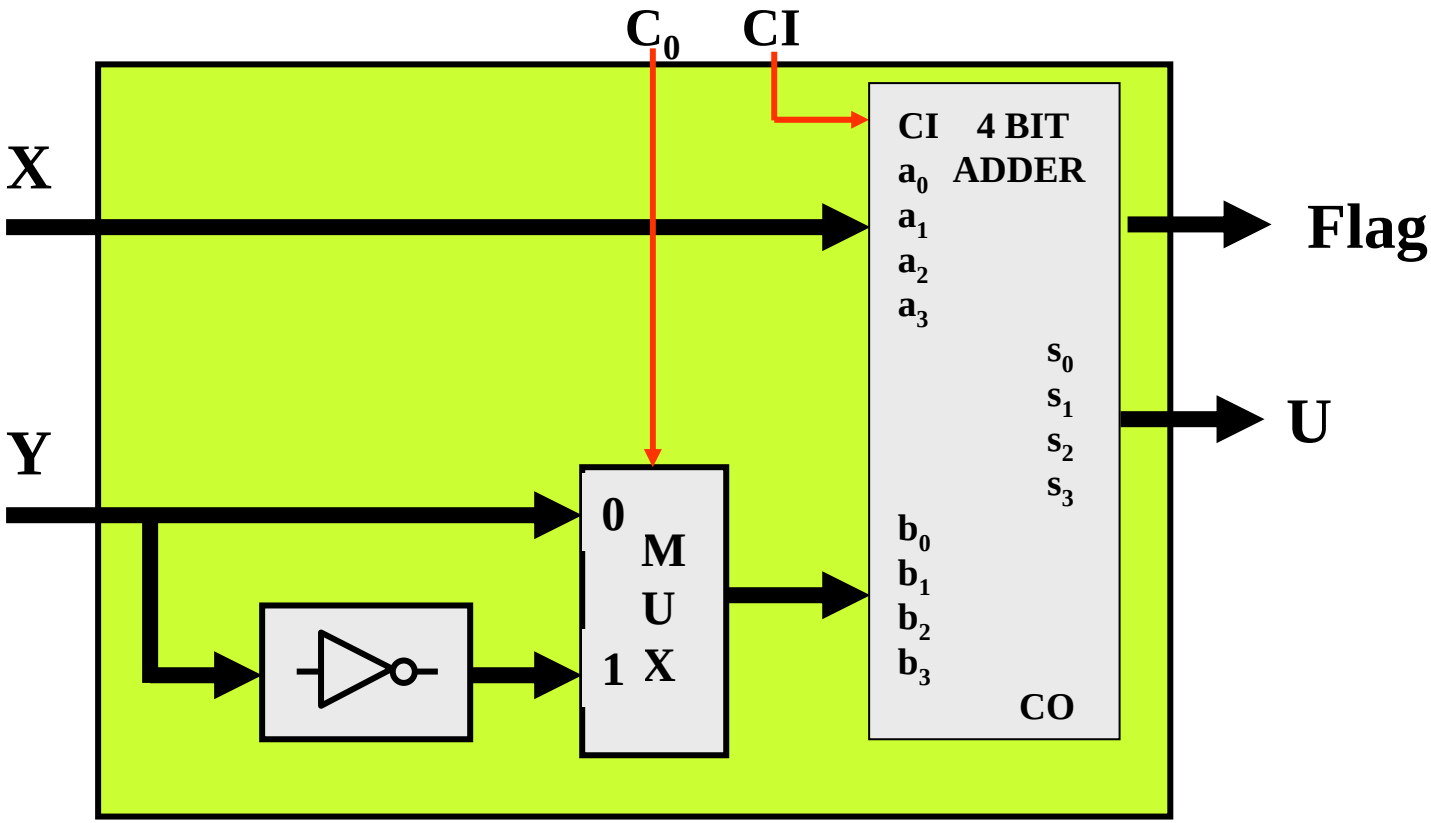
Sommatore a N bit: risultato della simulazione (N=16)

Avendo a disposizione un generico sommatore ad N bit e' immediato progettare una semplice ALU a N bit...

ALU a N bit

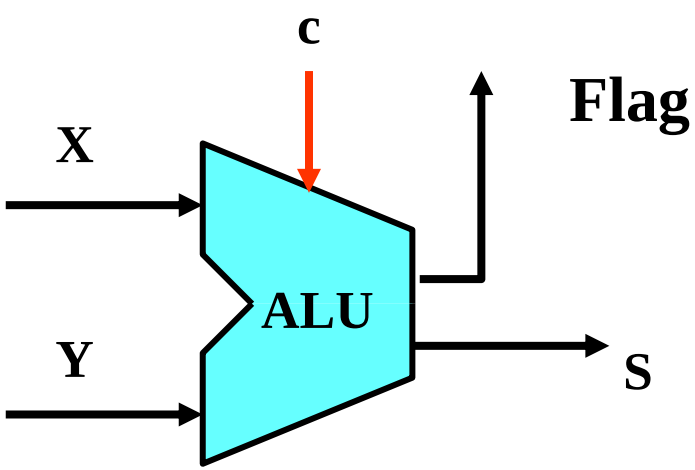

codice di operazione
c
Proprietà del risultato
flag
X
I° operando
ALU
Y
II° operando
U
risultato
u = F_c (X , Y)

ALU base a N bit



CI / C_0	0	1
0	X+Y	X+Y+1
1	X-Y-1	X-Y

Segnali di “flag”



ZF → ZERO FLAG: vale “1” se il risultato vale “0”.

SF → SIGN FLAG: vale “1” se il risultato è negativo.

CF → CARRY FLAG: vale “1” se CO=1 .

OVF → OVERFLOW FLAG: vale “1” in caso di traboccamento del risultato di un’operazione fra numeri relativi rappresentati in complemento a 2.

PF → PARITY FLAG: vale “1” se il bit di parità del risultato vale 1.

Generazione dei flag per una ALU a N bit

$$ZF = (S_0 + S_1 + S_2 + \bullet\ \bullet\bullet + S_{N-1})'$$

$$SF = S_{N-1}$$

$$CF = Cout_{N-1}$$

$$OVF = Cout_{N-2} \oplus Cout_{N-1}$$

$$PF = S_0 \oplus S_1 \oplus S_2 \oplus \bullet\ \bullet\bullet \oplus S_{N-1}$$

alu_Nbit.vhd

```
ENTITY alu_Nbit IS
   GENERIC(N: NATURAL :=4);
   PORT (X,Y            : IN   BIT_VECTOR(N-1 DOWNTO 0);  -- input
         CIN,C0         : IN   BIT;                       -- control bits
         S,COUT         : OUT  BIT_VECTOR(N-1 DOWNTO 0);  -- output bits
        SF,CF,OVF       : OUT  BIT);                      -- flags
END alu_Nbit;
ARCHITECTURE arch_alu_Nbit OF alu_Nbit IS
BEGIN
   PROCESS(X,Y,CIN,C0)
            VARIABLE TX,TY,TCOUT           :BIT_VECTOR(N-1 DOWNTO 0);
            VARIABLE CARRY_IN,CARRY_OUT    :BIT;
            VARIABLE S1T,S2T,C1T,C2T       :BIT;

            BEGIN
            -- ****** rete di pre-laborazione ********
            TX:=X;
            CASE (C0) IS
                        WHEN '0' => TY:=Y;
                        WHEN '1' => TY:=NOT(Y);
            END CASE;
            FOR i IN 0 TO N-1 LOOP
                        -- seleziona il corretto carry in ingresso
                        IF (i=0) THEN
                                    -- riporto in ingresso al sommatore CIN
                                    CARRY_IN:=CIN;
                        ELSE
                                    -- riporto da precedente operazione
                                    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;
```

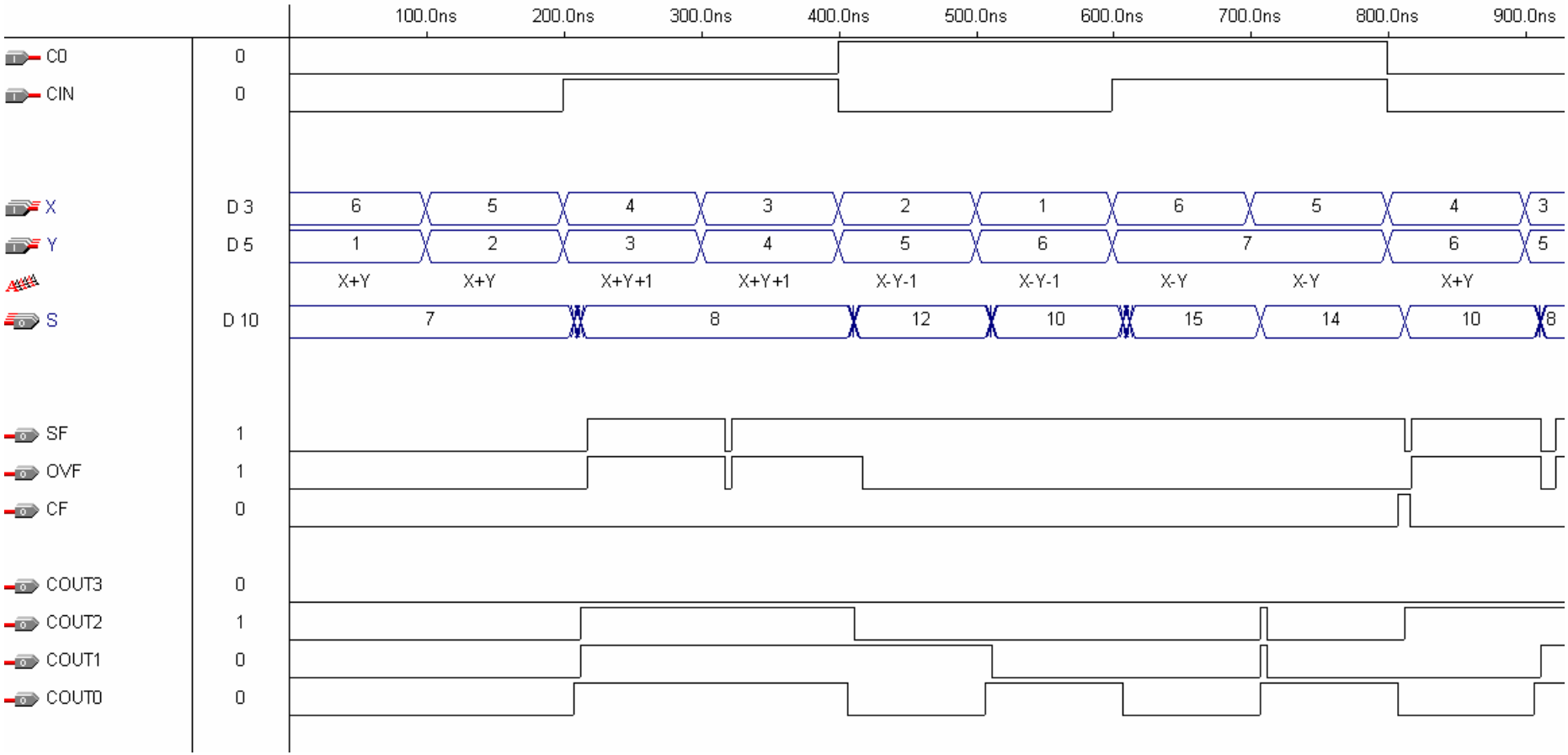

Codice VHDL di una semplice ALU a N bit.

Caso specifico di N=4.

```
            -- ******** Flags ***********

            -- Se il MSB e' '1' il segno e' negativo
            IF (S2T='1')
                        THEN SF<='1';
                        ELSE SF<='0';
            END IF;

            -- Se il Cout dell'ultimo sommatore e' '1' allora CF='1'
            IF (CARRY_OUT='1')
                        THEN CF<='1';
                        ELSE CF<='0';
            END IF;

            -- Verifica Overflow (complemento a 2)
            IF ((CARRY_OUT XOR CARRY_IN)='1')
                        THEN OVF<='1';
                        ELSE OVF<='0';
            END IF;

   END PROCESS;

END arch_alu_Nbit;
```


Questo esempio mette in evidenza come il VHDL consenta di progettare reti logiche modulari facilitando il lavoro del progettista.

Nel caso specifico della ALU, e' possibile progettare ALU a N bit con un N qualsiasi cambiando un solo parametro del codice VHDL!



ALU a N bit: risultato della simulazione (N=4)

***Esercizio**: scrivere il codice VHDL che sintetizza il "flag di zero" (ZF) e il "flag di parità" (PF).*

WAIT

L'istruzione WAIT consente di sospendere l'esecuzione di un processo fino al verificarsi di un determinato evento.
Gli eventi che possono essere utilizzati in VHDL per riprendere l'esecuzione di un processo sono riassumibili nelle seguenti 3 categorie:

- **il cambiamento di uno o più segnali** (**WAIT ON**)
- **il verificarsi di una espressione booleana** (**WAIT UNTIL**)
- **la fine di un determinato intervallo di tempo** (**WAIT FOR**)

L'istruzione WAIT e' tipicamente utilizzata nel progetto di reti sequenziali sincrone perché consente di bloccare l'evoluzione di un processo fino a che non si verifica un evento (es. fronte del segnale di clock).

In un processo se si utilizza l'istruzione WAIT non si deve definire la *sensitivity list*.

Formalmente, la sintassi dell’istruzione WAIT e’ la seguente:

```
WAIT ON <signal_changes>;

es: WAIT ON clock,a,b
```

```
WAIT UNTIL <expression_is_TRUE>;

es: WAIT UNTIL ((4*x)<50);
```

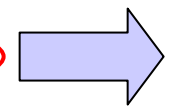
```
WAIT FOR <specific_amount_of_time>

es: WAIT FOR 50 ns;
```

Flip-Flop D

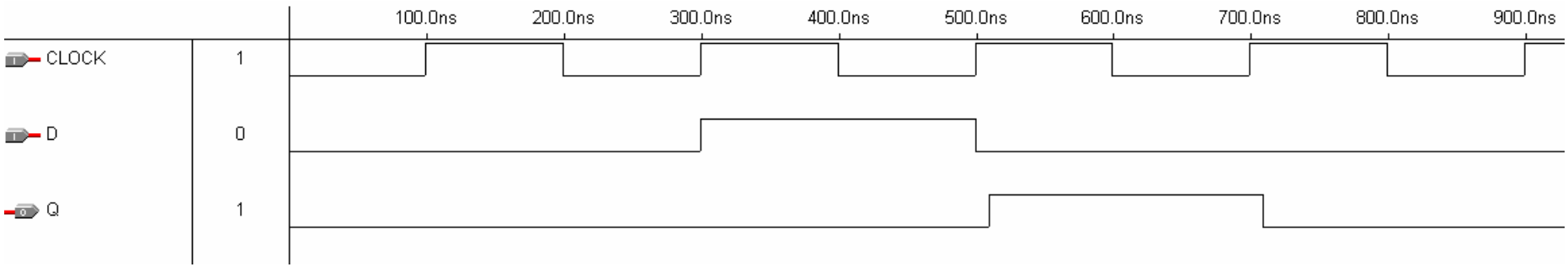
Il codice seguente mostra la realizzazione in VHDL di un Flip-Flop D utilizzando l'istruzione WAIT.
Il segnale di ingresso deve propagarsi in uscita quando si verifica un fronte positivo del clock.

fliflopd.vhd

```
ENTITY flipflopD IS

        PORT(D,CLOCK  : IN   BIT;
             Q        : OUT  BIT);

END flipflopD;

ARCHITECTURE arch_flipflopD OF flipflopD IS

BEGIN
        PROCESS
        BEGIN
            WAIT UNTIL CLOCK = '1' AND CLOCK'EVENT;

            Q<=D;

        END PROCESS;
END arch_flipflopD;
```



Codice VHDL che sintetizza un Flip-Flop D



Flip Flop D: risultato della simulazione

Segnali ed attributi

In VHDL ai segnali sono associati degli attributi.
Tali attributi consentono di avere informazioni sull'evoluzione temporale dei segnali.

'EVENT

'ACTIVE

'LAST_VALUE

'LAST_EVENT

'LAST_ACTIVE

Esempio: L'istruzione **WAIT UNTIL CLOCK = '1' AND CLOCK'EVENT;**
blocca l'esecuzione del processo fino a che:

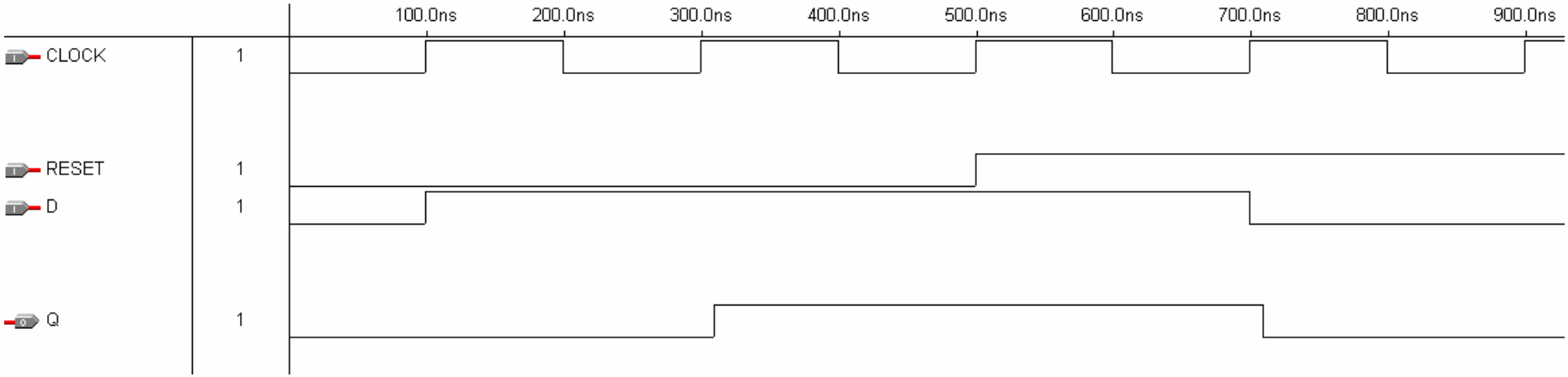
a) il segnale di clock e' a livello logico '1'

b) si verifica un evento (fronte) sul segnale di clock

Flip-Flop D con RESET sincrono

```
ENTITY flipflopD_reset_sincrono IS

        PORT(D,CLOCK,RESET   : IN   BIT;
             Q               : OUT  BIT);

END flipflopD_reset_sincrono;

ARCHITECTURE arch_flipflopD OF flipflopD _reset_sincrono IS

BEGIN
        PROCESS
        BEGIN
            WAIT UNTIL CLOCK = '1' AND CLOCK'EVENT;

            IF (RESET='1')
                THEN
                  Q<='0';
                ELSE
                  Q<=D;
            END IF;

        END PROCESS;
END arch_flipflopD;
```

Codice VHDL che sintetizza un Flip-Flop D con segnale di RESET sincrono



Flip Flop D con RESET sincrono: risultato della simulazione

Contatore modulo N

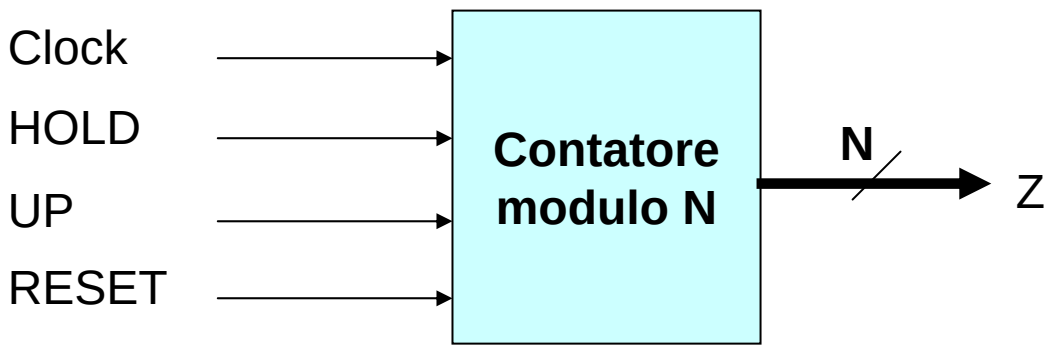
Si desidera progettare un contatore modulo N in VHDL.

Il contatore presenta un ingresso di clock e tre ingressi sincroni HOLD, UP e RESET:

HOLD - se '1' blocca il conteggio

UP - se '1' conta in avanti, se '0' conta indietro

RESET - se '1' azzera l'uscita



Perché e' importante limitare il range del tipo di dato utilizzato per memorizzare lo stato ?

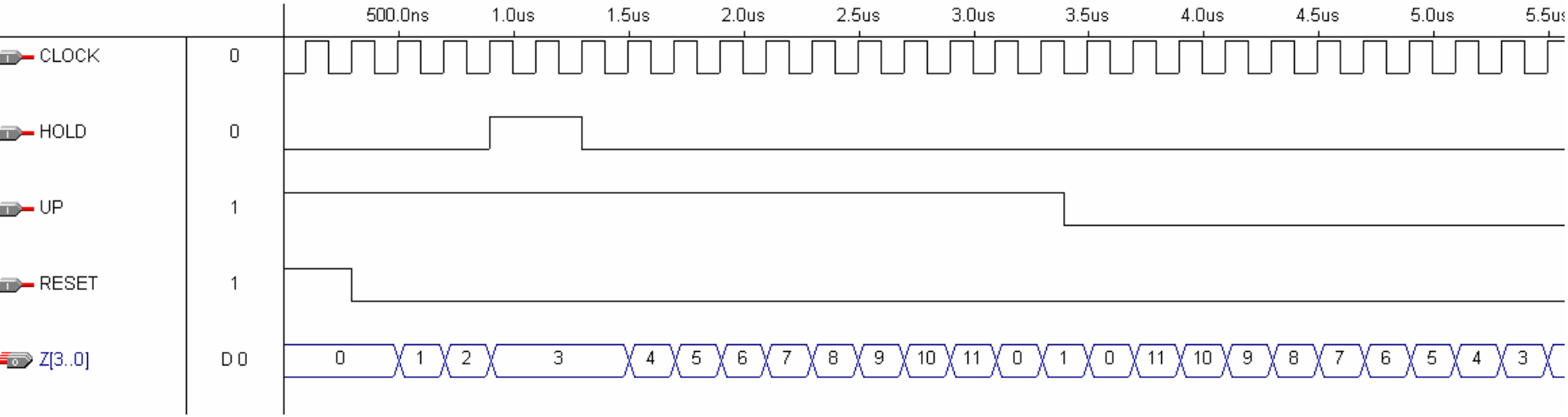
(mediante 'TYPE SMALL_INTEGER IS RANGE 0 TO 12;' per un contatore modulo 12*)?*

Codice VHDL che sintetizza un generico contatore modulo N (N=12)

counterbase.vhd

```
PACKAGE MYTYPES IS
         TYPE SMALL_INTEGER IS RANGE 0 TO 12;    --  dimensione della base di conteggio
END PACKAGE;

USE WORK.MYTYPES.ALL;

ENTITY CounterBase IS
   GENERIC(BASE          : SMALL_INTEGER :=12;    --  base di conteggio
           MAX           : SMALL_INTEGER :=11);   --  max valore di conteggio (BASE-1)

   PORT (CLOCK           : IN   BIT;              -- clock
         HOLD,UP,RESET   : IN   BIT;              -- control inputs (sincroni)
         Z               : OUT  SMALL_INTEGER);   -- output bits
END CounterBase;
```

```
ARCHITECTURE arch_CounterBase OF CounterBase IS

BEGIN
          PROCESS
          VARIABLE STATO :SMALL_INTEGER;

          BEGIN
          WAIT UNTIL (CLOCK='1' AND CLOCK'EVENT);
          -- fronte positivo del clock
                              IF (UP='1' AND HOLD='0')
                                        THEN
                                                  IF (STATO=MAX)
                                                            THEN STATO:=0;
                                                            ELSE STATO:=(STATO+1);
                                                  END IF;
                              END IF;

                              IF (UP='0' AND HOLD='0')
                                        THEN
                                                  IF (STATO=0)
                                                            THEN STATO:=MAX;
                                                            ELSE STATO:=STATO-1;
                                                  END IF;
                              END IF;

                    -- Reset prioritario rispetto ad HOLD e UP
          IF (RESET='1')
                    THEN STATO:=0;
                    END IF;

          Z<=STATO;

          END PROCESS;

END arch_CounterBase;
```



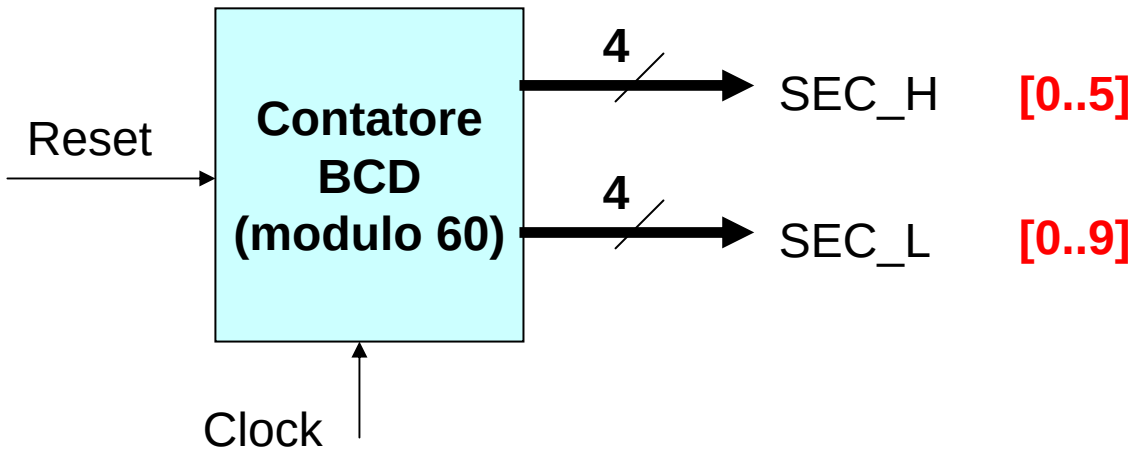
Contatore modulo N: risultato della simulazione (N=12)

***Esercizio**: data modularità del codice presentato e' facile ottenere un contatore utilizzabile per il conteggio dei secondi di un orologio digitale (modulo 60).*

***Soluzione**: inclusa nel file "counter60.vhd" reperibile presso il sito web di Reti Logiche L-A e/o Laboratorio di Informatica L-A.*

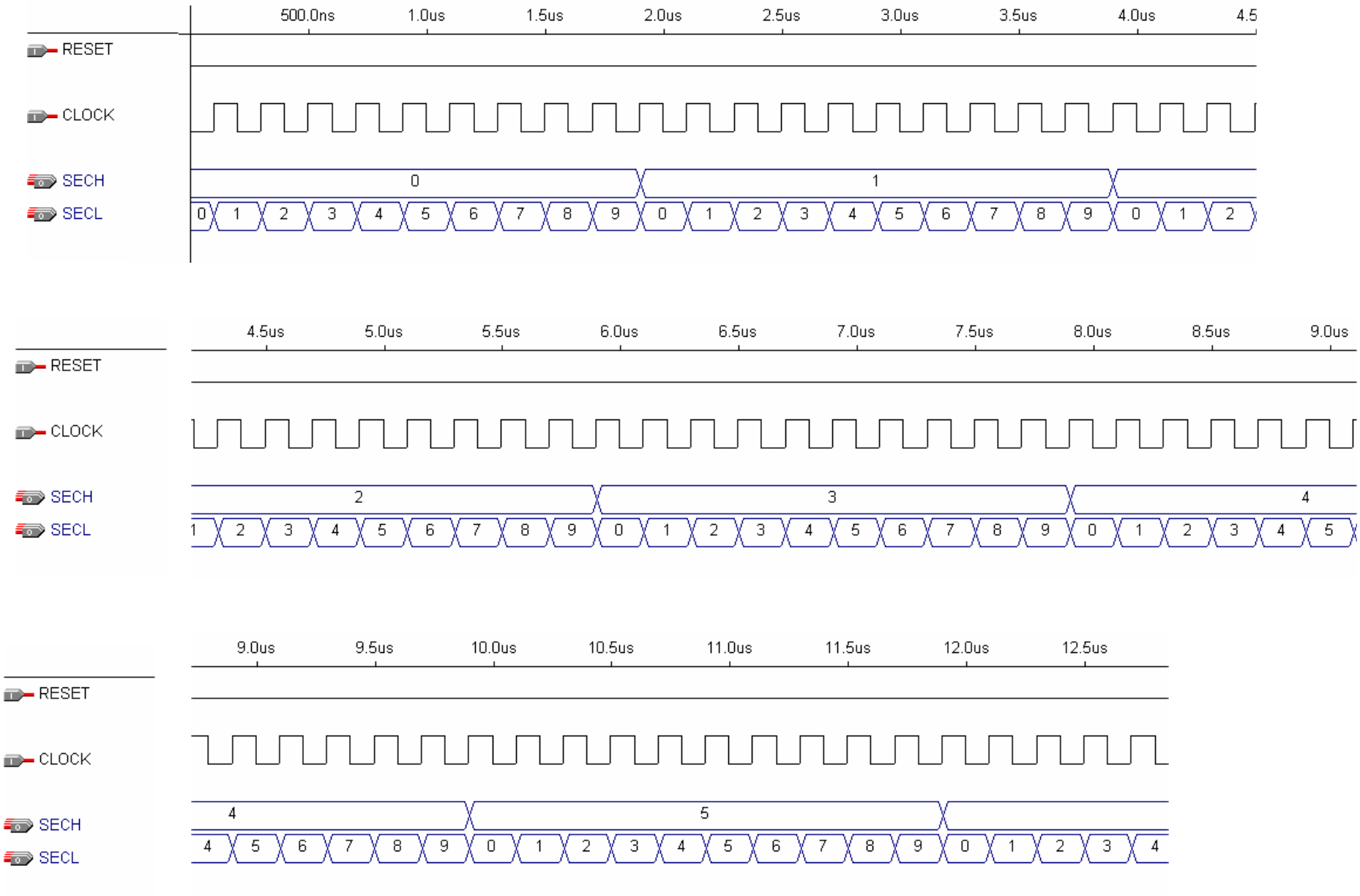
Contatore BCD

Si desidera progettare un contatore BCD modulo 60.



```
PACKAGE MYTYPES IS
          TYPE SMALL_INTEGER IS RANGE 0 TO 10;  -- dimensione base massima conteggio
END PACKAGE;

USE WORK.MYTYPES.ALL;

ENTITY CounterBCD IS

   PORT (CLOCK   : IN   BIT;                -- clock
         RESET   : IN   BIT;                -- control input (sincrono)
         SECH    : OUT  SMALL_INTEGER;      -- output, cifra piu' significativa dei secondi
         SECL    : OUT  SMALL_INTEGER);     -- output, cifra meno significativa dei sec
END CounterBCD;

ARCHITECTURE arch_CounterBCD OF CounterBCD IS

SIGNAL SECONDI_L :SMALL_INTEGER;
SIGNAL SECONDI_H :SMALL_INTEGER;

BEGIN
-- @@@@@@@@@@@@@@@ SECONDI 'LOW' @@@@@@@@@@@@@@@@
LOW:PROCESS

          VARIABLE STATOL :SMALL_INTEGER;
          BEGIN
          WAIT UNTIL (CLOCK='1' AND CLOCK'EVENT);

          IF (STATOL=9)
                    THEN STATOL:=0;
                    ELSE STATOL:=(STATOL+1);
          END IF;
          -- Reset (prioritario)
          IF (RESET='1')
                    THEN
                    STATOL:=0;
          END IF;
          SECONDI_L<=STATOL;
END PROCESS;
```

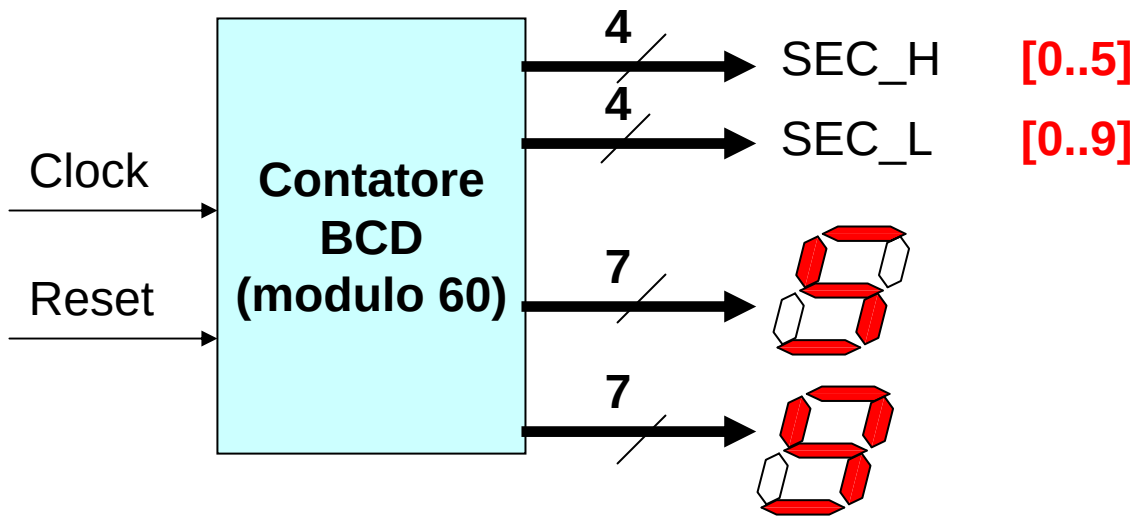
```
-- @@@@@@@@@@@@@@@ SECONDI 'HIGH' @@@@@@@@@@@@@@@@

HIGH:PROCESS

           VARIABLE STATOH :SMALL_INTEGER;
           BEGIN
           WAIT UNTIL (CLOCK='1' AND CLOCK'EVENT);

           IF (SECONDI_L=9) THEN
                     IF (STATOH=5)
                               THEN STATOH:=0;
                               ELSE STATOH:=(STATOH+1);
                     END IF;
           END IF;


           -- Reset (prioritario)
    IF (RESET='1')
           THEN STATOH:=0;
    END IF;

           SECONDI_H<=STATOH;

END PROCESS;

-- assegnamenti (concorrenti)

SECL<=SECONDI_L;
SECH<=SECONDI_H;


END arch_CounterBCD;
```



Contatore BCD modulo 60: risultato della simulazione

Contatore BCD modulo 60 con uscita codificata con codice a 7 segmenti

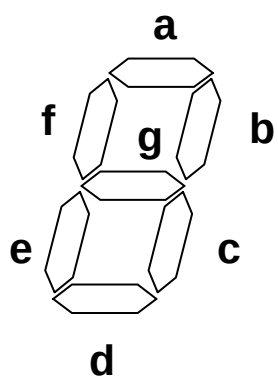
Si desidera progettare un contatore BCD modulo 60 con uscite in grado di pilotare due display a 7 segmenti.



```vhdl
PACKAGE MYTYPES IS
            TYPE SMALL_INTEGER IS RANGE 0 TO 10; -- dimensione base massima di conteggio
END PACKAGE;

USE WORK.MYTYPES.ALL;

ENTITY CounterBCD7segmenti IS
   PORT (CLOCK          : IN   BIT;                          -- clock
         RESET          : IN   BIT;                          -- control input (sincrono)
         SECH           : OUT  SMALL_INTEGER;                -- output, cifra piu' significativa dei secondi
         SECL           : OUT  SMALL_INTEGER;                -- output, cifra meno significativa dei secondi
         SEGMENTI_H     : OUT  BIT_VECTOR(6 DOWNTO 0);
         SEGMENTI_L     : OUT  BIT_VECTOR(6 DOWNTO 0));

END CounterBCD7segmenti;

ARCHITECTURE arch_CounterBCD7segmenti OF CounterBCD7segmenti IS
            SIGNAL SECONDI_L :SMALL_INTEGER;
            SIGNAL SECONDI_H :SMALL_INTEGER;
BEGIN

-- @@@@@@@@@@@@@@@ SECONDI 'LOW' @@@@@@@@@@@@@@@@

LOW:PROCESS
   VARIABLE STATOL :SMALL_INTEGER;
            BEGIN

            WAIT UNTIL (CLOCK='1' AND CLOCK'EVENT);

            IF (STATOL=9)
                        THEN STATOL:=0;
                        ELSE STATOL:=(STATOL+1);
            END IF;

            -- Reset (prioritario)
            IF (RESET='1')
                        THEN
                        STATOL:=0;
            END IF;
SECONDI_L<=STATOL;

END PROCESS;
```

```
-- @@@@@@@@@@@@@@@ SECONDI 'HIGH' @@@@@@@@@@@@@@@@

HIGH:PROCESS
   VARIABLE STATOH :SMALL_INTEGER;

            BEGIN
            WAIT UNTIL (CLOCK='1' AND CLOCK'EVENT);
            IF (SECONDI_L=9) THEN
                        IF (STATOH=5)
                                    THEN STATOH:=0;
                                    ELSE STATOH:=(STATOH+1);
                        END IF;
            END IF;
            -- Reset (prioritario)
    IF (RESET='1')
            THEN STATOH:=0;
    END IF;

SECONDI_H<=STATOH;

END PROCESS;

-- @@@@@@@@@ BCD -> 7 Segmenti (LOW) @@@@@@@@@@

            --           gfedcba
            WITH SECONDI_L SELECT
            SEGMENTI_L<="1000000" WHEN 0,
                        "1111001" WHEN 1,
                        "0100100" WHEN 2,
                        "0110000" WHEN 3,
                        "0011001" WHEN 4,
                        "0010010" WHEN 5,
                        "0000010" WHEN 6,
                        "1111000" WHEN 7,
                        "0000000" WHEN 8,
                        "0010000" WHEN 9,
                        "0110000" WHEN OTHERS;   -- Errore (E)
```

```
-- @@@@@@@@@@ BCD -> 7 Segmenti (HIGH) @@@@@@@@@@

            --                          gfedcba

            WITH SECONDI_H SELECT
                        SEGMENTI_H<="1000000" WHEN 0,
                                    "1111001" WHEN 1,
                                    "0100100" WHEN 2,
                                    "0110000" WHEN 3,
                                    "0011001" WHEN 4,
                                    "0010010" WHEN 5,
                                    "0110000" WHEN OTHERS;   -- Errore (E)


-- assegnamenti (concorrenti)

SECL<=SECONDI_L;
SECH<=SECONDI_H;


END arch_CounterBCD7segmenti;
```



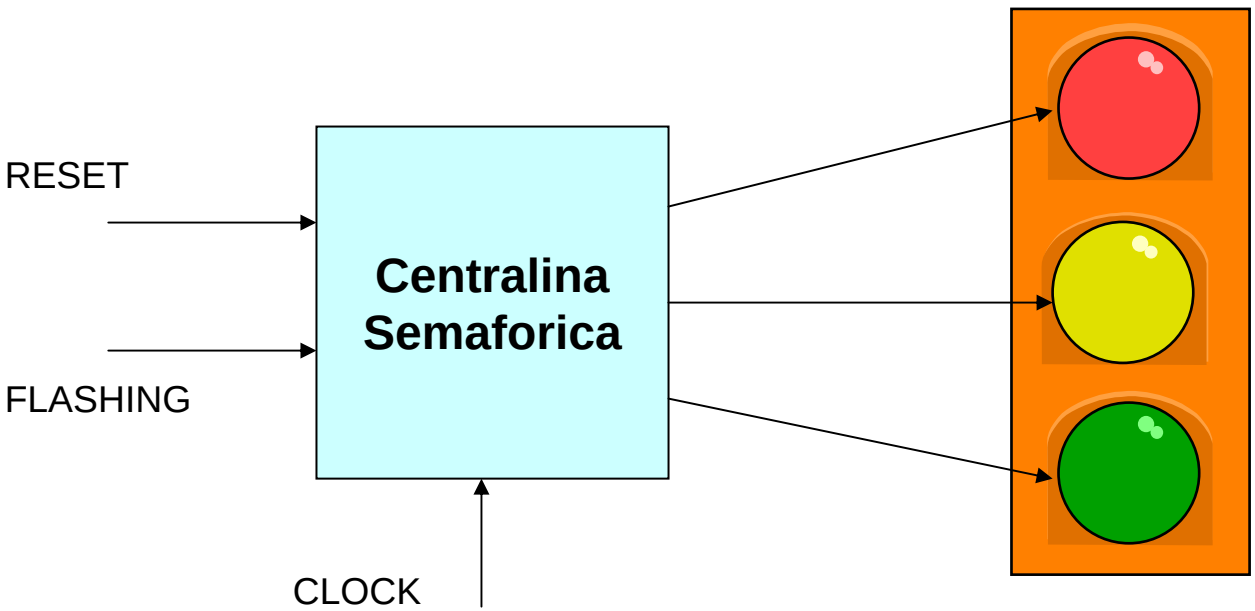
Rete di controllo di un semaforo

Si desidera progettare una rete logica che controlli il flusso di un semaforo secondo le seguenti specifiche:

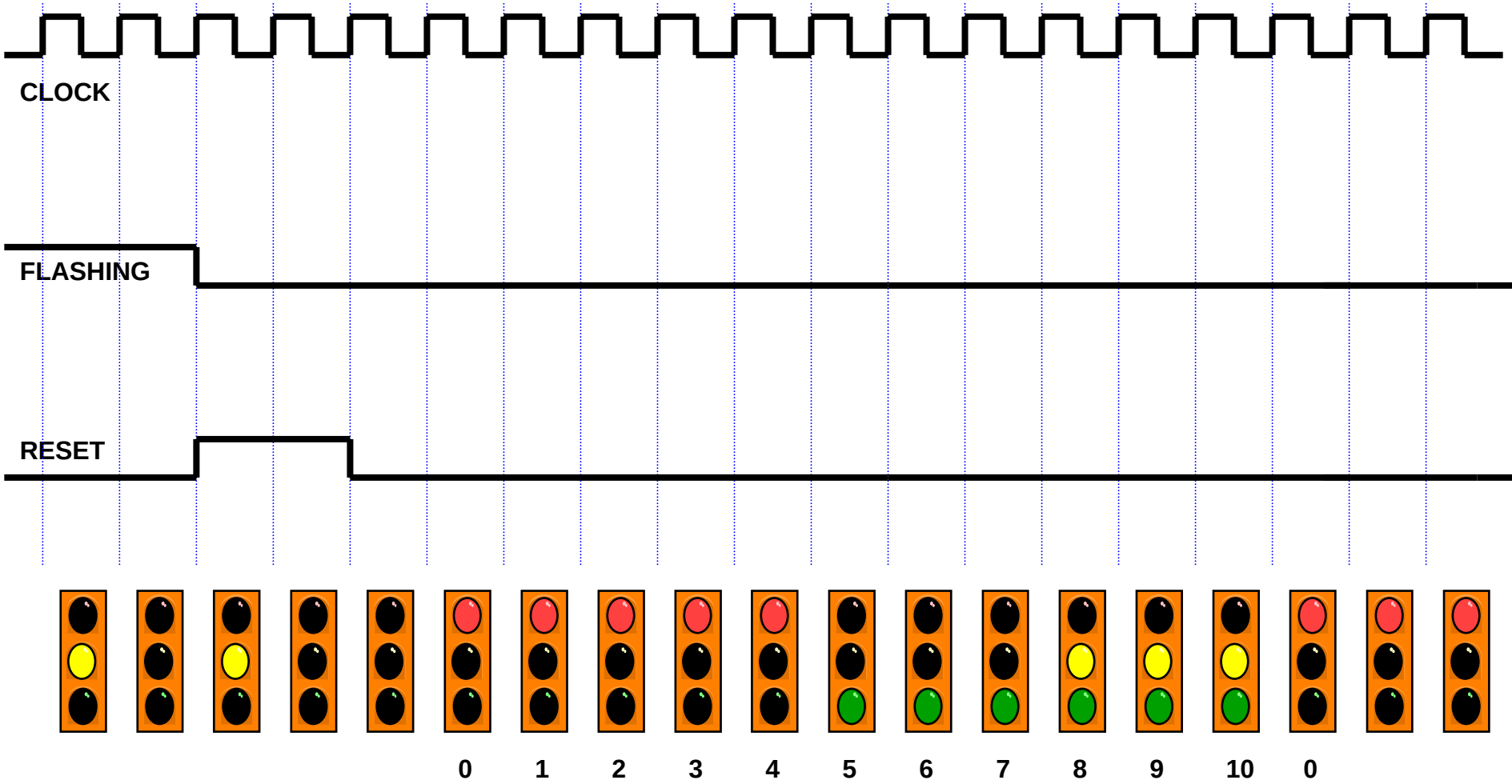
Se RESET attivo tutte le luci del semaforo devono essere spente. Quando il RESET si disattiverà il semaforo deve partire con la prima fase di ROSSO.

Se FLASHING attivo la luce gialla deve lampeggiare e le altre due luci devono essere spente.

Altrimenti il semaforo deve funzionare in modo standard rimando ROSSO per 5 periodi di clock, VERDE per 3 periodi di clock e con le luci GIALLE e VERDI attive per altri 3 periodi di clock.



Il principio di funzionamento del semaforo che si intende progettare e' illustrato dalle forme d'onda seguenti:



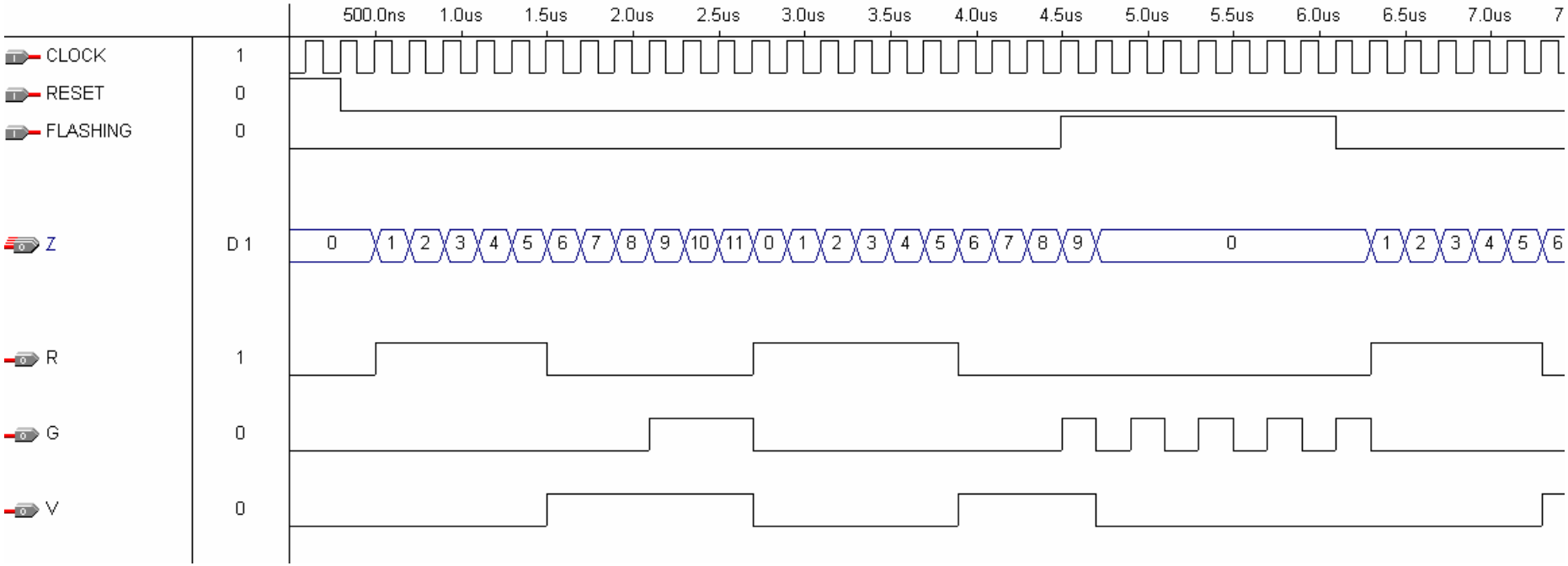
```
PACKAGE MYTYPES IS
          TYPE SMALL_INTEGER IS RANGE 0 TO 11;    --  dimensione della base di conteggio
END PACKAGE;

USE WORK.MYTYPES.ALL;

ENTITY semaforo IS
   GENERIC(BASE              : SMALL_INTEGER :=11;   --  base di conteggio
           MAX               : SMALL_INTEGER :=10;   --  max valore di conteggio (BASE-1)
           PERIODO_ROSSO     : SMALL_INTEGER :=5;    --  durata segnale rosso
           PERIODO_VERDE     : SMALL_INTEGER :=3;    --  durata segnale verde
           PERIODO_GIALLO    : SMALL_INTEGER :=3);   --  durata segnale verde-giallo

   PORT (CLOCK      : IN   BIT;           -- Clock
         FLASHING   : IN   BIT;           -- Se attivo (1) il semaforo lampeggia
         RESET      : IN   BIT;           -- Reset
         Z          : OUT SMALL_INTEGER;  -- Output contatore (debug)
         R,G,V      : OUT  BIT);          -- output bits (R,G,V)

END semaforo;

ARCHITECTURE arch_semaforo OF semaforo IS

BEGIN
          PROCESS
          VARIABLE STATO                :SMALL_INTEGER;
          VARIABLE ROSSO,VERDE,GIALLO   : BIT;
```

```
BEGIN
WAIT UNTIL (CLOCK='1' AND CLOCK'EVENT);
-- fronte positivo del clock

                    -- funzionamento standard
                    IF (FLASHING='0')
                              THEN
                                        -- aggiorna stato
                                        IF (STATO<BASE)
                                        THEN
                                                  STATO:=STATO+1;
                                        ELSE
                                                  STATO:=0;
                                        END IF;

                                        -- elabora uscite in fnzione dello stato
                                        IF (STATO<=PERIODO_ROSSO)
                                        THEN
                                                  -- Rosso
                                                  ROSSO:='1';
                                                  VERDE:='0';
                                                  GIALLO:='0';
                                        ELSIF (STATO<=(PERIODO_ROSSO+PERIODO_VERDE))
                                                  THEN
                                                  -- Verde
                                                  ROSSO:='0';
                                                  VERDE:='1';
                                                  GIALLO:='0';
                                        ELSE
                                                  -- Verde-Giallo
                                                  ROSSO:='0';
                                                  VERDE:='1';
                                                  GIALLO:='1';
                                        END IF;
                    END IF;
```

```
                    -- semaforo lampeggiante ?
                    IF (FLASHING='1')
                            THEN
                                    STATO:=0;
                                    ROSSO:='0';
                                    VERDE:='0';
                                    GIALLO:=NOT(GIALLO);
                            END IF;

                    -- Reset ? (prioritario)
                    IF (RESET='1')
                            THEN
                                    STATO:=0;
                                    ROSSO:='0';
                                    VERDE:='0';
                                    GIALLO:='0';
                            END IF;

            R<=ROSSO;
            V<=VERDE;
            G<=GIALLO;

            Z<=STATO;

            END PROCESS;
```



```
END arch_semaforo;
```

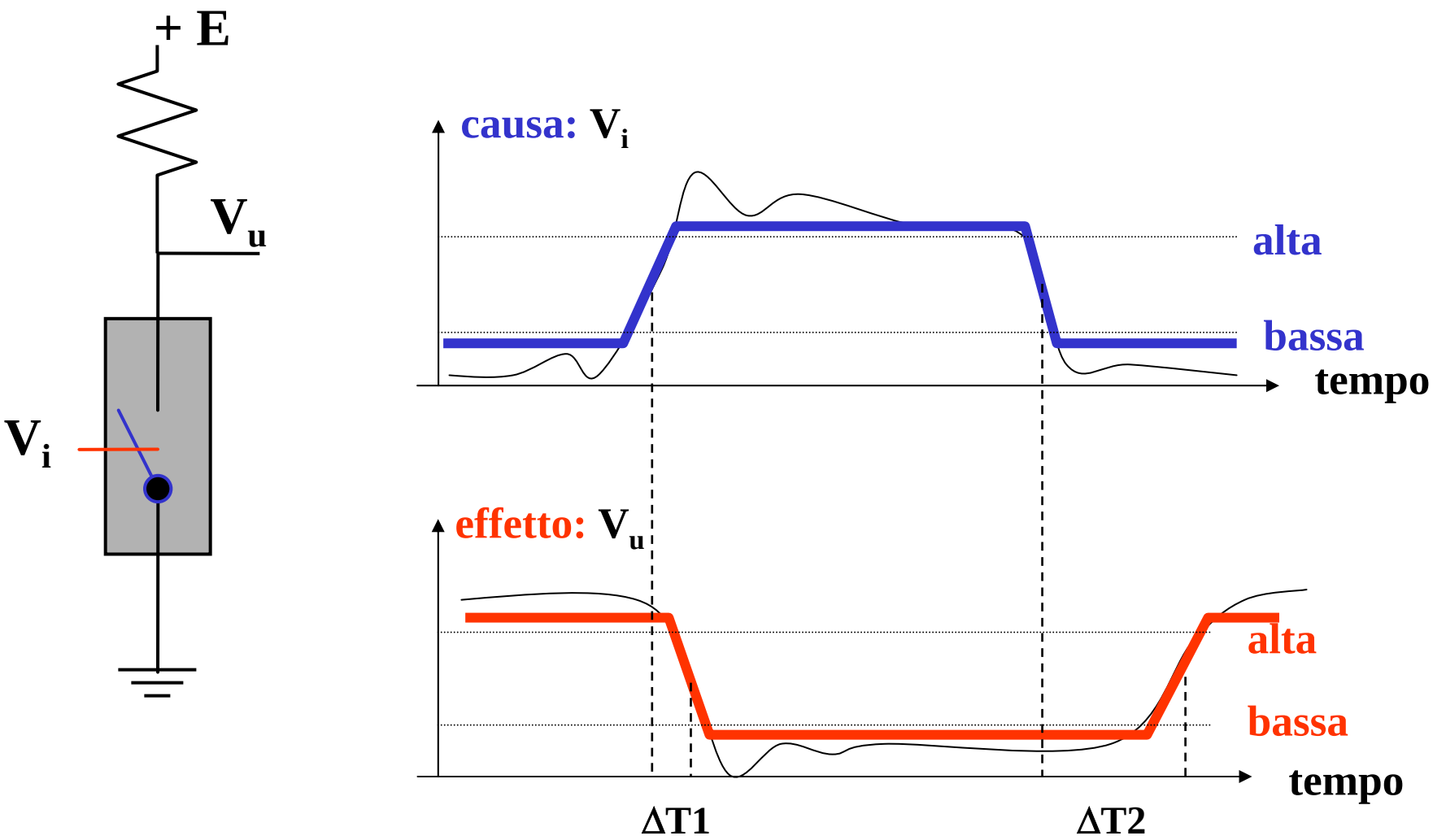


Controllo di un semaforo : risultato della simulazione

L'approccio seguito per il progetto della rete di controllo di un semaforo risulta sostanzialmente diverso dalla sintesi mediante metodi canonici studiati nel Corso di Reti Logiche. Anche in questo caso si evidenzia la modularità della sintesi attraverso il VHDL: cambiando pochi parametri del codice si ottiene una rete di controllo adattabile alle diverse fasi di controllo di un semaforo generico.

Modelli di ritardo disponibili in VHDL

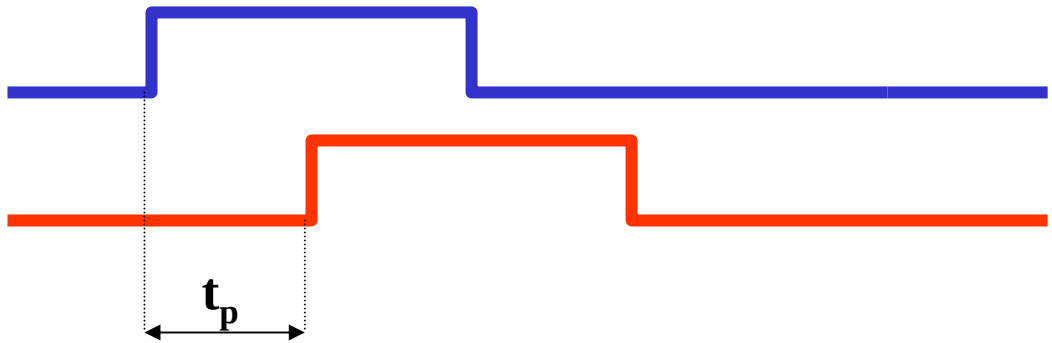
Il ritardo del Not elettronico



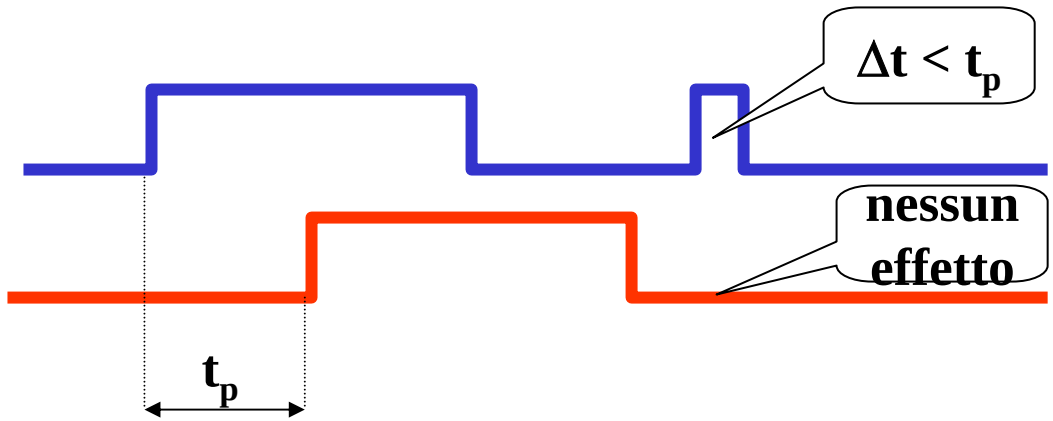
Il ritardo di propagazione

ritardo di propagazione: t_p

- **Ritardo puro**



- **Ritardo inerziale**



Il modello del ritardo inerziale è il più vicino alla realtà.
Il ritardo puro (o matematico) è però più facile da simulare.

In VHDL(*) sono simulabili entrambi i tipi di ritardo:

Ritardo Inerziale (default)

```
y <= a AFTER 5 ns;
```

Ritardo Puro

```
y <= TRANSPORT a AFTER 5 ns;
```

(*) Altera Max + II effettua la sintesi su dispositivi con ritardi specifici.