RELAZIONE DEL PROGETTO DI UN CONTATORE BINARIO UP/DOWN MODULO 4 PER IL CORSO DI APPARATI ELETTRONICI

1. INTRODUZIONE

In generale un contatore è un dispositivo che memorizza (e a volte visualizza) il numero di volte che un particolare evento o processo si verifica, oppure la quantità di una sostanza solida, liquida o gassosa che transita sotto il suo controllo. Il dispositivo può essere meccanico, elettromeccanico, o elettronico; in quest'ultimo caso viene impegnata componentistica digitale la quale utilizza una base di tempo (clock) come segnale primario. I contatori possono lavorare in due modi:

- in progressione, cioè il valore è incrementato
- in regressione, cioè il valore è decrementato (decontatori)

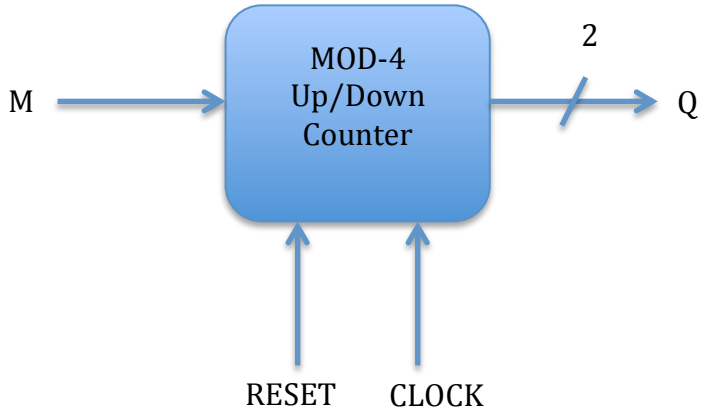
Un contatore integrato è un componente elettronico costituito da un circuito integrato in cui sono implementate funzioni di contatore digitale. E' possibile realizzare questa funzione anche impiegando semplici dispositivi di "registro" come i flip-flop. Sono disponibili, già riuniti in un unico chip, contatori in codice binario, decimale, esadecimale a 4 oppure a 8 bit.
I contatori più versatili dispongono di alcuni pin di condizionamento, per mezzo dei quali il progettista può scegliere la tipologia di conteggio ottimale per il circuito in progetto:

- Up/Down, consente di impostare la modalità di conteggio in progressione o regressione.
- Bin/Dec, consente di effettuare il conteggio in binario o in esadecimale.
- MR (Master Reset), serve ad azzerare il contatore; la transizione del livello logico su questo pin forza il contatore ad azzerarsi, indipendentemente dallo stato in cui si trova. Può essere sincrono o asincrono.
- Enable, serve per abilitare o disabilitare il contatore al conteggio.
- PL (Parallel Load), la transizione del livello logico di questo pin, permette di trasferire al contatore un valore numerico in formato binario, preventivamente già presente su appropriati pin, sostituendolo al valore presente in quel momento.
- TC (Terminal Count) o RC (Ripple Clock) o RCO (Ripple Carry Out), su questo pin viene generato un segnale sotto forma di impulso, ogniqualvolta il contatore arriva a fine conteggio.

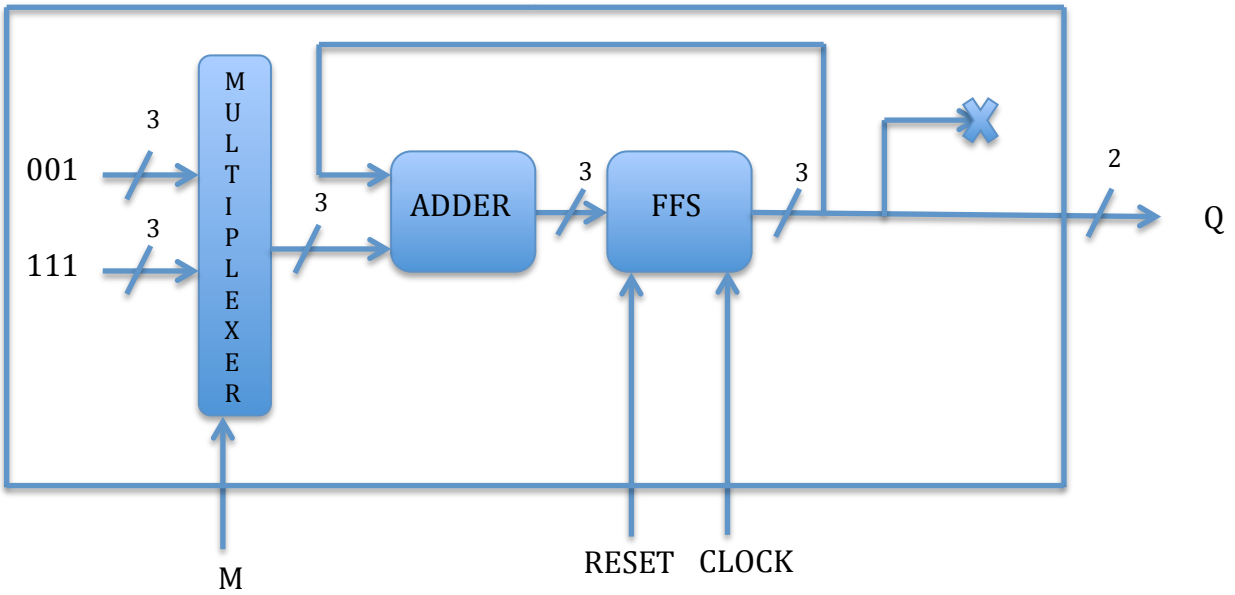
Il segnale necessario a incrementare il conteggio è costituito da un impulso (clock) applicato al pin appropriato, e a seconda del tipo di circuito, lo "scatto" può avvenire durante la transizione dell'impulso da zero a uno, o viceversa. Alcuni contatori sono provvisti di due ingressi di clock, il primo viene utilizzato per contare in avanti, il secondo per contare all'indietro.
I contatori binari costituiti da flip-flop T sono utilizzati anche come divisori di frequenza, funzione largamente utilizzata in molteplici applicazioni.

2. DESCRIZIONE DELL'ARCHITETTURA SELEZIONATA PER LA REALIZZAZIONE

Le specifiche fornite, indicano come tipologia di contatore quella up/down modulo 4 in base 2. Analizziamo il diagramma a blocchi assieme agli ingressi e alle uscite:



L'input del circuito è dato dai piedini "M", "RESET" e "CLOCK". L'output è dato da "Q" costituito da due piedini. Se "M" vale 0 il contatore lavorerà incrementando ad ogni ciclo di clock il valore attuale del conteggio. Viceversa se "M" vale 1, il contatore decrementerà il valore attuale del conteggio ad ogni ciclo di clock. Il piedino "RESET" azzera il valore attuale del conteggio. Il piedino "CLOCK" viene utilizzato come segnale di sincronizzazione del circuito. L'uscita "Q" sta su 2 bit, in quanto il contatore è in modulo 4, e rappresenta il valore attuale di conteggio.
Per la progettazione di questo circuito si è scelto di combinare componenti più semplici tra loro:



Come si evince dallo schema a blocchi, il contatore è costituito da 3 sottoreti: un multiplexer, un sommatore e un flip flop di tipo d-edge triggered.

Il multiplexer prende in ingresso 2 sequenze di 3 bit “001” e “111”, rispettivamente 1 e -1 in rappresentazione complemento a 2. Se M=0 l’uscita del multiplexer sarà pari a “001” altrimenti sarà uguale a “111”.
Il sommatore prende in ingresso l’uscita del multiplexer e il risultato della somma effettuata nel precedente ciclo di clock. Gli ingressi vengono interpretati come numeri interi rappresentati in complemento a due. Questa soluzione consente di utilizzare un sommatore anche come sottrattore.
Il flip flop d-edge triggered consente di rendere sincrona la rete: quando si verifica un fronte in salita del clock (transizione del piedino “CLOCK” da “0” a “1”) il valore in ingresso (corrispondente all’uscita del sommatore) viene trasmetto in uscita. Nel caso di attivazione del piedino “RESET” l’uscita del flip flop varrà “0”.
L’uscita del flip flop è su 3 bit, e viene rimandata in ingresso al sommatore. L’uscita Q, su 2 bit, è determinata dai 2 bit meno significativi dell’uscita del flip flop grazie alla rappresentazione utilizzata.

3. CODICE VHDL

Il codice VHDL è distribuito su più file: uno per ogni componente utilizzato, in aggiunta al file di configurazione, di testbench e del contatore vero e proprio.

MULTIPLEXER

1 · multiplexer.vhd · 2012-03-13 23:20 · Stefano

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY multiplexer is

   generic (N : INTEGER:=3);
   port(i          : in  std_logic;
        o          : out std_logic_VECTOR (N-1 downto 0)
   );
END multiplexer;

architecture bhv of multiplexer is

BEGIN
    MUL:process(i)
         begin
         CASE i is
              when '0' => o<= "001"; --  1
              when '1' => o<= "111"; -- -1
              when others =>     NULL;
         end case;
     end  process MUL;

END bhv;
```

SOMMATORE

1 · adder.vhd · 2012-03-13 23:13 · Stefano

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY adder is

   generic (N : INTEGER:=3);
   port( a          : in  std_logic_VECTOR (N-1 downto 0);
         b          : in  std_logic_VECTOR (N-1 downto 0);
         carry_in   : in  std_logic;
         s          : out std_logic_VECTOR (N-1 downto 0);
         carry_out  : out std_logic);
END adder;

architecture BEHAVIOURAL of adder is

BEGIN
    SUM:process(a,b,carry_in)
      variable C:std_logic;
     begin
      C:=carry_in;
        FOR i IN 0 TO N-1 LOOP
         -- Calculate bit sum using carry from previous step, then carry out
         s(i)<= a(i) XOR b(i) XOR C;
         C:= (a(i) AND b(i)) OR (a(i) AND C) OR (b(i) AND C);
      END LOOP;
       carry_out <= C;
      END  process SUM;

END BEHAVIOURAL;
```

FLIP FLOP D-EDGE TRIGGERED

1 · dff.vhd · 2012-03-13 23:25 · Stefano

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY dff is
     generic (N : INTEGER:=3);
   port( d          : in  std_logic_VECTOR (N-1 downto 0);
         clk        : in  std_logic;
         reset      : in  std_logic;
         q          : out std_logic_VECTOR (N-1 downto 0));
END dff;

architecture BEHAVIOURAL of dff is

BEGIN
    DFF:process(clk)
     begin
      IF (clk'EVENT AND clk='1') THEN

              FOR  i IN 0 TO N-1 LOOP
                q(i) <= d(i) and reset;
              END LOOP;

      END IF;
     END process DFF;

END BEHAVIOURAL;
```

CONTATORE

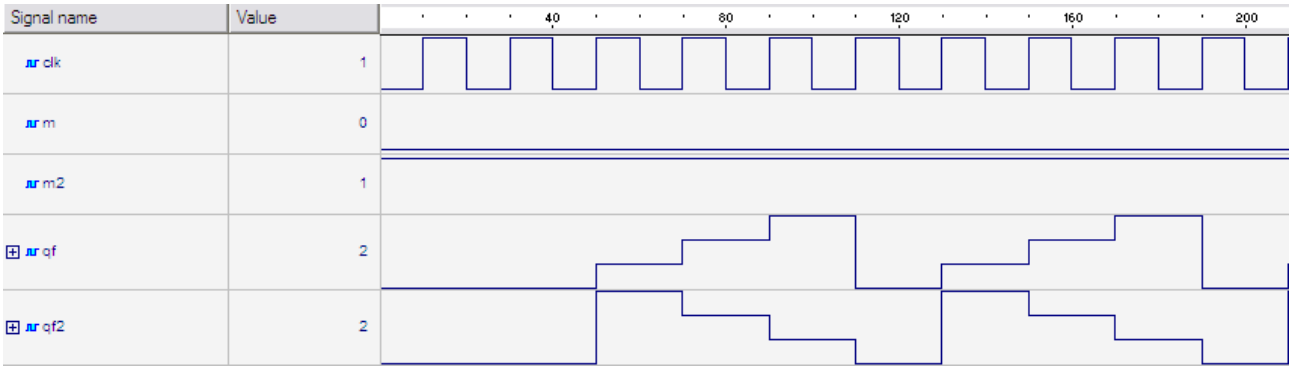
1 · contatore.vhd · 2012-03-14 00:41 · Stefano

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY contatore is
   port( m          : in  std_logic;
          clk        : in  std_logic;
         reset      : in  std_logic;
         qf         : out std_logic_VECTOR (1 downto 0));
END contatore;

architecture structural of contatore is

     component multiplexer is    --multiplexer
     generic (N : INTEGER:=3);
     port(
          i          : in  std_logic;
          o          : out std_logic_VECTOR (N-1 downto 0)
          );
     end component multiplexer;

     component adder is          --sommatore
          port(
               a             : in  std_logic_VECTOR (2 downto 0);
               b             : in  std_logic_VECTOR (2 downto 0);
            carry_in   : in  std_logic;
            s          : out std_logic_VECTOR (2 downto 0);
            carry_out  : out std_logic
               );
    end component adder ;

    component dff is            --flip flop
        port(
            d          : in  std_logic_VECTOR (2 downto 0);
            clk        : in  std_logic;
            reset      : in  std_logic;
            q          : out std_logic_VECTOR (2 downto 0)
             );
    end component dff;

    signal mul_to_adder : std_logic_VECTOR (2 downto 0);    --uscita del multiplexer
    signal adder_to_dff : std_logic_VECTOR (2 downto 0);    --uscita del sommatore
    signal dff_to_qf : std_logic_VECTOR (2 downto 0);       --uscita del flip flop
BEGIN
    MUL: multiplexer port map (m,mul_to_adder);
    AD1: adder port map (dff_to_qf,mul_to_adder,'0',adder_to_dff,open);
    DFF1: dff port map (adder_to_dff,clk,reset,dff_to_qf);
    qf<=dff_to_qf(1 downto 0);
END architecture structural;
```

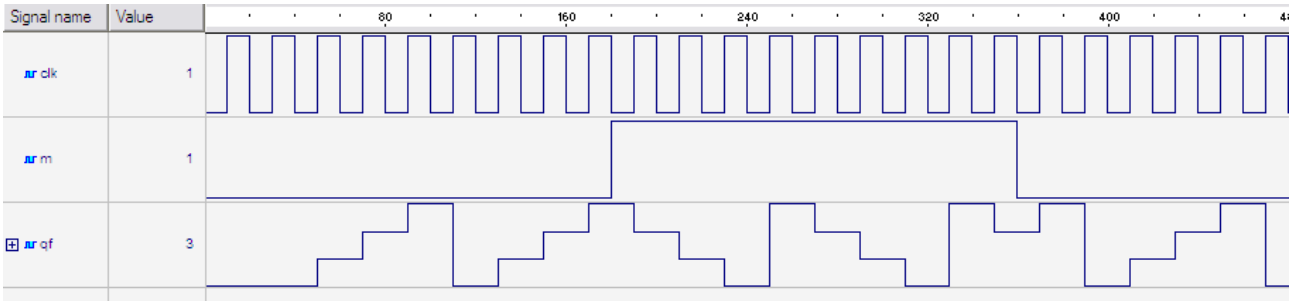
Il file di configurazione non viene riportato in quanto la configurazione utilizzata è quella di default.

4. TESTBENCH PER LA VERIFICA

Per quanto riguarda il testbench, inizialmente sono state create due istanze del contatore in questione con ingressi “M” differenti e uscite “Q” differenti. In questo modo è possibile determinare la correttezza del circuito dall’andamento dei valori in uscita.



Dal grafico sopra riportato è possibile notare come nel caso della prima istanza, con ingresso “m” pari al valore “0” e uscita “qf”, quest’ultima abbia un andamento ciclicamente crescente da “0” a “3”. Viceversa, nella seconda istanza con ingresso “m2” di valore “1” e uscita “qf2”, quest’utima ha un andamento ciclicamente decrescente da “3” a “0”. Da ciò si può evincere che le modalità “up” e “down” funzionano correttamente.
Questa verifica però non è parsa soddisfacente, in quanto il circuito viene testato da un punto di vista statico. Per tutta la durata del test infatti, la variabile di ingresso “M” rimane costante. Perciò in seguito si è deciso di implementare il testbench in modo che periodicamente la variabile d’ingresso “M” passasse da “0” a “1” e viceversa. Più precisamente “M” cambierà valore ogni 180 ns, tempo appositamente scelto in modo da non essere multiplo di 80 ns, corrispondente a 4 cicli di clock.



Si può facilmente notare come al variare di dell’ingresso “m” corrisponda una variazione dell’andamento dell’uscita “qf”. Possiamo concludere quindi che il circuito si comporta in maniera corretta anche variando nel tempo l’ingresso.

TESTBENCH

1 · contatore_test.vhd · 2012-03-14 00:40 · Stefano

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY contatore_tb IS

END contatore_tb;

ARCHITECTURE contatore_test OF contatore_tb IS

   COMPONENT contatore
         port(     m          : in  std_logic;
              clk       : in  std_logic;
              reset     : in  std_logic;   -- Active low --
              qf        : out std_logic_VECTOR (1 downto 0)
               );
   END COMPONENT;

   ----------------------------------------------------------------------------


   CONSTANT MckPer  :  TIME     := 20 ns;     -- Master Clk period
   CONSTANT TestLen :  INTEGER  := 1024;      -- No. of Count (MckPer/2) for test

-- I N P U T     S I G N A L S

   SIGNAL   clk   : std_logic := '0';
   SIGNAL   reset : std_logic := '0';
   SIGNAL     m     : std_logic := '0';

-- O U T P U T     S I G N A L S

SIGNAL qf: std_logic_VECTOR (1 downto 0);

   SIGNAL clk_cycle : INTEGER;
   SIGNAL Testing: Boolean := True;

BEGIN

   I: contatore
   PORT MAP(m, clk, reset, qf);
   -- Generates clk and change m every 180 ns

         clk     <= NOT clk AFTER MckPer/2 WHEN Testing ELSE '0';
         m         <= NOT m AFTER MckPer*9 WHEN Testing;

   -- Runs simulation for TestLen cycles;

   Test_Proc: PROCESS(clk)
      VARIABLE count: INTEGER:= 0;
   BEGIN
     clk_cycle <= (count+1)/2;

     CASE count IS
          WHEN   0  => reset <= '0';
          WHEN   4  => reset <= '1';

          WHEN (TestLen - 1) =>   Testing <= False;
          WHEN OTHERS => NULL;
     END CASE;

     count:= count + 1;
   END PROCESS Test_Proc;

END contatore_test;
```

5. CONCLUSIONI

Il lavoro svolto ha previsto la progettazione di un contatore binario modulo 4 di tipo up/down dotato di reset. Tale circuito è stato ottenuto tramite il corretto “assemblaggio” di circuiti più semplici come multiplexer, sommatore e flip flop d-edge triggered. In una prima fase è stato prodotto uno schema a blocchi dell’architettura utilizzata per la realizzazione; dopodiché si è passati alla scrittura del relativo codice VHDL tramite lo strumento software “Active-HDL”. Infine, è stato scritto il codice VHDL relativo al testbench, coerentemente a quanto già detto nel capitolo precedente. Data la bassa complessità del circuito in oggetto, e le esercitazioni effettuate durante il corso, che sono state un ottimo spunto per il compimento di questo progetto, non sono stati riscontrati particolari problemi durante lo sviluppo.