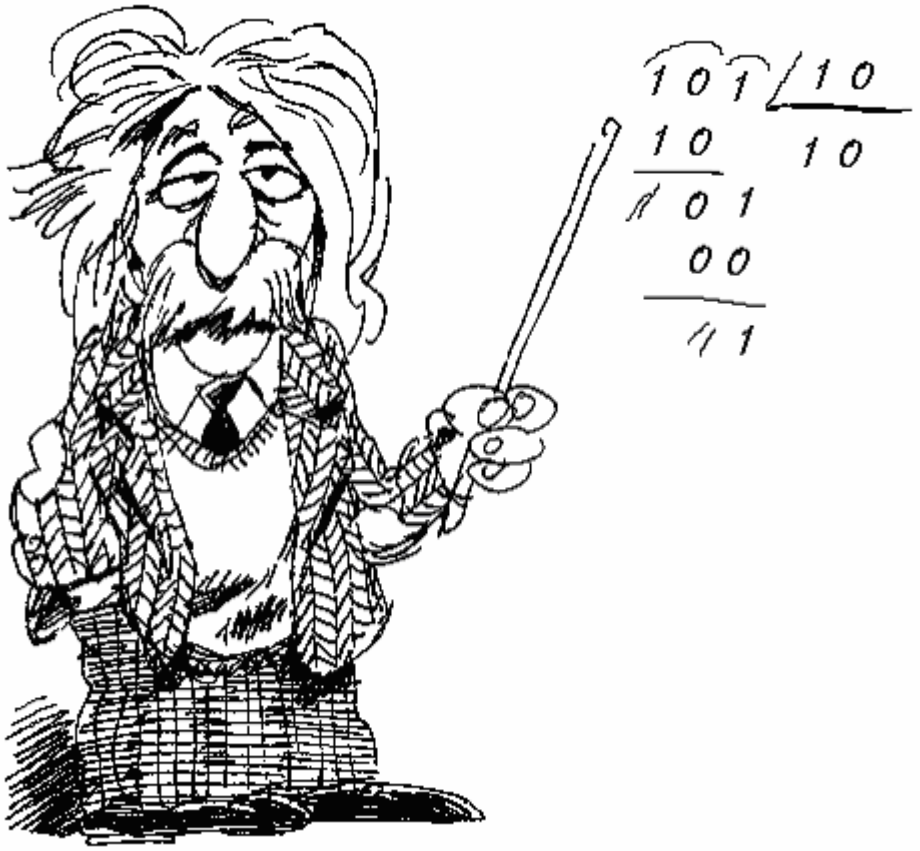
Implementazione di un divisore binario in VHDL



Gioacchino Ilario Gargiulo
Salvatore Nugnes
Vincenzo Capuano

REVISIONE 0

OTTOBRE 2007

Se il lettore nota errori, imprecisioni o altro, gli autori sarebbero felici di ricevere queste osservazioni per integrarle in una nuova revisione del documento.

E' possibile contattare gli autori ai seguenti indirizzi di posta elettronica:

- oigio2003@yahoo.it
- salvatorenugnes@libero.it
- enzo@sitoserio.it

SOMMARIO

PREFAZIONE

Oggi giorno l'elettronica digitale fa da padrone in ogni campo. Troviamo la sua applicazione in moltissime cose che utilizziamo, basti pensare all'automobile per spostarci, al climatizzatore o ai videogames... potremmo stare qui ad elencare migliaia di oggetti, tutti oramai entrati nelle nostre abitudini quotidiane.

La progettazione dei sistemi elettronici digitali avviene sempre più spesso attraverso l'utilizzo di veri e propri linguaggi di programmazione. Chi ne fa da padrone è il VHDL, il quale rappresenta lo strumento fondamentale per la progettazione dei moderni circuiti integrati digitali. Le sue applicazioni spaziano dai microprocessori (DSP, acceleratori grafici), alle comunicazioni (Cellulari, TV satellitare), all'automobile (Navigatori, controllo stabilità) e molte altre.

Il VHDL (*VHSIC Hardware Description Language*) consente di modellare facilmente l'interazione tra i vari blocchi funzionali che compongono un sistema. Queste interazioni sono essenzialmente lo scambio di segnali di controllo e di dati tra i vari oggetti che costituiscono il sistema. In un sistema hardware infatti, ogni oggetto da modellizzare, che sia esso una semplice porta logica o un complesso microprocessore, reagisce istantaneamente agli stimoli in ingresso producendo dei cambiamenti sulle sue uscite. Ogni blocco funzionale, a sua volta, è descritto nella relazione ingressi-uscite, usando i classici costrutti del linguaggi di programmazione (*if*, *for*, *while*).

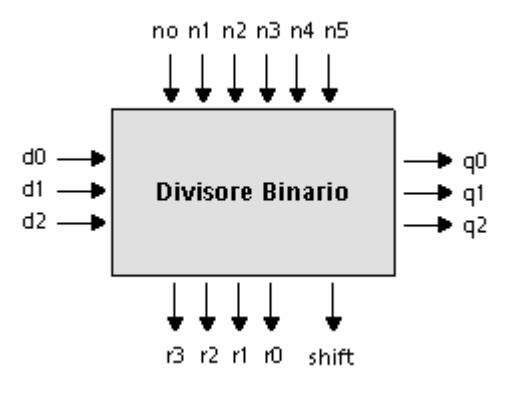
Nelle pagine che seguono presentiamo un semplice esempio, partendo dalla progettazione teorica per poi arrivare all'implementazione del codice.
L'esempio mostrato si presenta utile dal punto di vista didattico e non si propone affatto di fornire la soluzione completa a tutte le problematiche della progettazione col VHDL, si faccia quindi riferimento alla documentazione più specifica (libri, pagine di manuali, ecc.) per ottenere una spiegazione esauriente e dettagliata degli argomenti trattati.

NOTE SULLA DISTRIBUZIONE

Progetto

1.1. Scopo

Lo scopo del nostro progetto è quello di realizzare un divisore binario che abbia il numeratore a 6 bit, il divisore e il quoziente a 3 bit ed il resto a 4 bit, come mostrato nella figura seguente:



Nel caso in cui il dividendo è minore del divisore ($N<D$) si deve garantire quanto segue:

- Dividendo (numeratore) a 6 bit del tipo: $N = 0.n_5\, n_4\, n_3\, n_2\, n_1\, n_0$
- Divisore (denominatore) a 3 bit del tipo: $D = 0.\, d_2\, d_1\, d_0$
- Quoziente a 3 bit del tipo: $Q = 0.\, q_2\, q_1\, q_0$
- Resto a 4 bit del tipo: $R = 0\ .00\, r_3\, r_2\, r_1\, r_0$

Se $N<D$ allora indicando con $N^* = n_5\, n_4\, n_3\, n_2\, n_1\, n_0$ e $D^* = d_2\, d_1\, d_0$ dovrà risultare:

$$\boldsymbol{N^* < 8\, D^*} \qquad (1)$$

Più in generale se la dimensione di D è k ($dim(D)=k$) e quella di N è $2k$ ($dim(N)=2k$), dovendosi verificare $N<D$ allora risulta:

$$\boldsymbol{N^* / 2^{2k} < D^* / 2^k \ \rightarrow\ N^* < 2^k\, D} \qquad (2)$$

Nella tabella riportata a pagina 11 sono elencate tutte le possibili combinazioni che soddisfano la relazione (1).

Nel caso in cui $N \geq D$ per poter ancora utilizzare la struttura si deve realizzare uno **shift** a destra di due posti del dividendo in maniera tale da rientrare ancora nel caso $N<D$. Ovviamente il quoziente di questa operazione ($N \geq D$) sarà spostato a sinistra di due posti rispetto al caso $N<D$. Un'operazione del genere comporterà, chiaramente, una perdita di precisione sul risultato in quanto nello spostare a destra il dividendo stiamo perdendo proprio le cifre meno significative. Il quoziente (caso $N \geq D$) dovrà essere, quindi, del tipo:

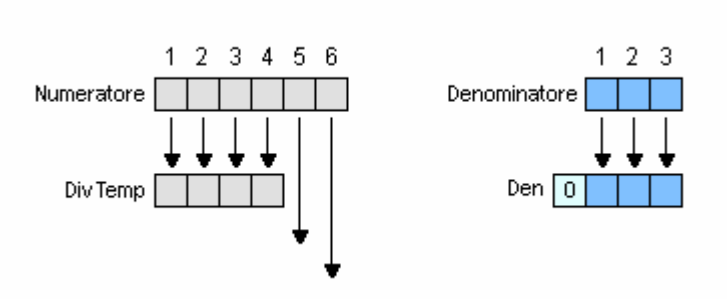
$$Q = q_2\, q_1.q_0 \qquad (3)$$

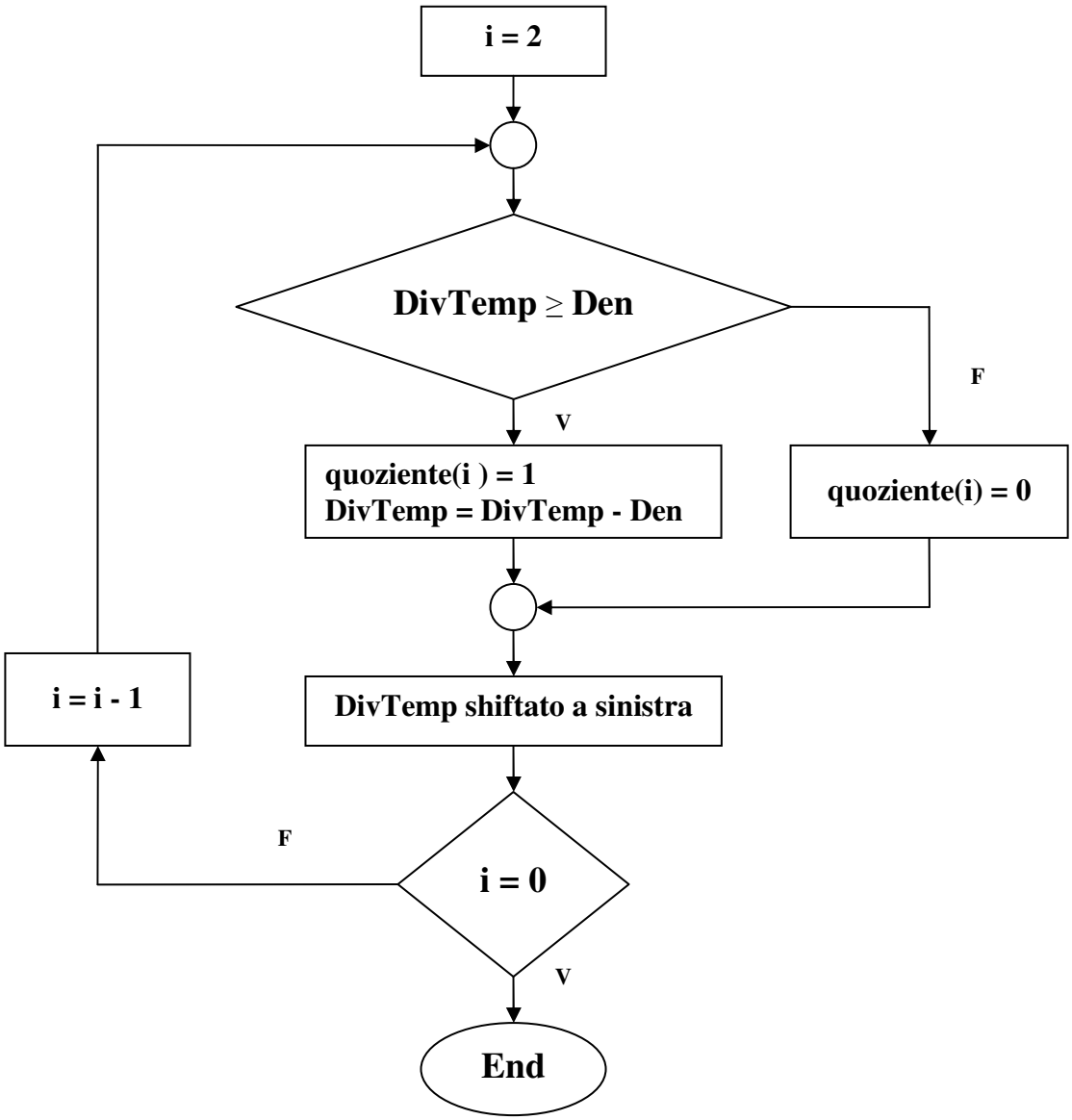
1.2. Architettura

Cerchiamo ora di capire come implementare un dispositivo del genere.

Nel linguaggio VHDL questo caso particolare sarà implementato con una procedura del tipo *comportamentale* (le combinazioni relative a N≥D sono tutte quelle non riportate nella tabella finale, per es. $N^*=28$ $D^*=2$).

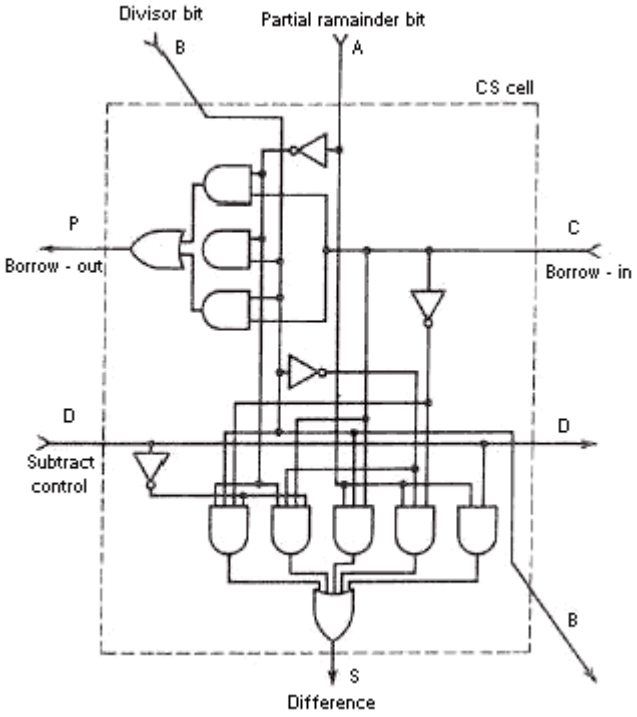
Dal punto di vista logico per calcolare il quoziente utilizzeremo la tecnica delle *sottrazioni successive*. Essa effettua sottrazioni del divisore (denominatore) dal dividendo (numeratore), ed ad ogni passo verifica se il risultato è maggiore o minore di zero. Nel primo caso continua a sottrarre ponendo a 1 il bit di quoziente, nel caso contrario pone lo stesso a 0. Ad ogni passo di sottrazione viene effettuato uno spostamento a sinistra del dividendo, e si va avanti col procedimento fintantoché si sono esauriti tutti gli spostamenti del dividendo verso sinistra. Il procedimento può risultare più chiaro se si utilizza il diagramma di flusso dell'algoritmo, riportato alla pagina seguente.





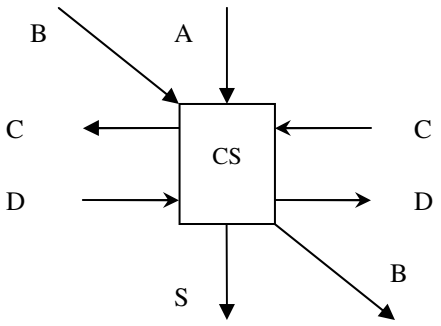
Per poter implementare un algoritmo del genere con una macchina combinatoria abbiamo bisogno di una struttura elementare che ci permetta di effettuare una sottrazione su singoli bit.

La struttura in questione è la seguente:

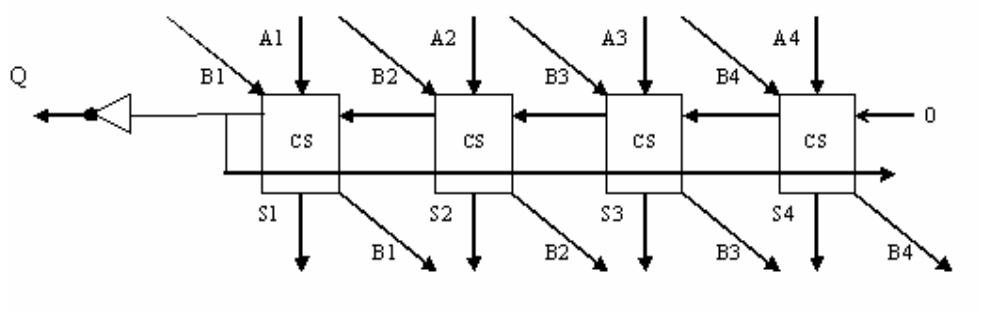


dove:

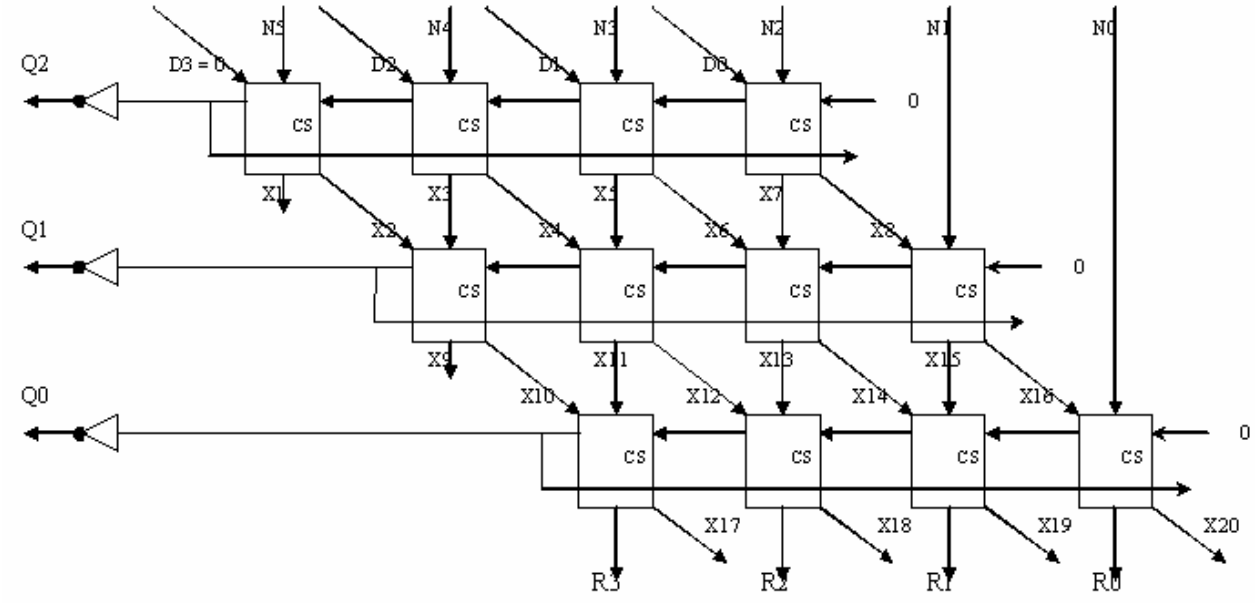
$$P = \overline{A}B + \overline{A}C + BC$$
$$S = AD + A\overline{B}\overline{C} + ABC + \overline{A}B\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D}$$



Poiché stiamo effettuando sottrazioni tra cifre con più bit, affiancheremo le strutture elementari in maniera da ottenere una riga del tipo:



Ovviamente sistemando più strutture riga di questo genere in modo da formare una matrice, otterremo uno schema circuitale che implementa l'algoritmo delle sottrazioni successive:



Si evidenzia dalla figura sovrastante come il resto (**borrow**) in uscita della cella elementare più significativa di ogni riga esplicita se la sottrazione dia risultato positivo o negativo (ossia risultato positivo significa *Divtemp≥den,* risultato negativo significa *Divtemp<den*). Inoltre viene inserita una porta *not* in quanto l'algoritmo prevede che il bit di quoziente sia alto quando la sottrazione può essere effettuata (ossia borrow in uscita uguale a zero).

Nel linguaggio VHDL il circuito combinatorio è stato sviluppato in modo *strutturale*. Per verificare la condizione N≥D o N<D è stato utilizzato una descrizione *comportamentale*. In particolare l'evento N≥D viene segnalato da un segnale chiamato *SHIFT* (in questo caso è a valore logico alto).

$N^* = n_5\, n_4\, n_3\, n_2\, n_1\, n_0$ $D^* = d_2\, d_1\, d_0$ $Q^* = q_2\, q_1\, q_0$ $R^* = r_3\, r_2\, r_1 r_0$

N^*	D^*	Q^*	R^*	N^*	D^*	Q^*	R^*	N^*	D^*	Q^*	R^*	N^*	D^*	Q^*	R^*	N^*	D^*	Q^*	R^*
0	1	0	0	0	4	0	0	16	5	3	1	24	6	4	0	24	7	3	3
1	1	1	0	1	4	0	1	17	5	3	2	25	6	4	1	25	7	3	4
2	1	2	0	2	4	0	2	18	5	3	3	26	6	4	2	26	7	3	5
3	1	3	0	3	4	0	3	19	5	3	4	27	6	4	3	27	7	3	6
4	1	4	0	4	4	0	4	20	5	4	0	28	6	4	4	28	7	4	0
5	1	5	0	5	4	1	1	21	5	4	1	29	6	4	5	29	7	4	1
6	1	6	0	6	4	1	2	22	5	4	2	30	6	5	0	30	7	4	2
7	1	7	0	7	4	1	3	23	5	4	3	31	6	5	1	31	7	4	3
0	2	0	0	8	4	2	0	24	5	4	4	32	6	5	2	32	7	4	4
1	2	0	1	9	4	2	1	25	5	5	0	33	6	5	3	33	7	4	5
2	2	1	0	10	4	2	2	26	5	5	1	34	6	5	4	34	7	4	6
3	2	1	1	11	4	2	3	27	5	5	2	35	6	5	5	35	7	5	0
4	2	2	0	12	4	3	0	28	5	5	3	36	6	6	0	36	7	5	1
5	2	2	1	13	4	3	1	29	5	5	4	37	6	6	1	37	7	5	2
6	2	3	0	14	4	3	2	30	5	6	0	38	6	6	2	38	7	5	3
7	2	3	1	15	4	3	3	31	5	6	1	39	6	6	3	39	7	5	4
8	2	4	0	16	4	4	0	32	5	6	2	40	6	6	4	40	7	5	5
9	2	4	1	17	4	4	1	33	5	6	3	41	6	6	5	41	7	5	6
10	2	5	0	18	4	4	2	34	5	6	4	42	6	7	0	42	7	6	0
11	2	5	1	19	4	4	3	35	5	7	0	43	6	7	1	43	7	6	1
12	2	6	0	20	4	5	0	36	5	7	1	44	6	7	2	44	7	6	2
13	2	6	1	21	4	5	1	37	5	7	2	45	6	7	3	45	7	6	3
14	2	7	0	22	4	5	2	38	5	7	3	46	6	7	4	46	7	6	4
15	2	7	1	23	4	5	3	39	5	7	4	47	6	7	5	47	7	6	5
0	3	0	0	24	4	6	0	0	6	0	0	0	7	0	0	48	7	6	6
1	3	0	1	25	4	6	1	1	6	0	1	1	7	0	1	49	7	7	0
2	3	0	2	26	4	6	2	2	6	0	2	2	7	0	2	50	7	7	1
3	3	1	0	27	4	6	3	3	6	0	3	3	7	0	3	51	7	7	2
4	3	1	1	28	4	7	0	4	6	0	4	4	7	0	4	52	7	7	3
5	3	1	2	29	4	7	1	5	6	0	5	5	7	0	5	53	7	7	4
6	3	2	0	30	4	7	2	6	6	1	0	6	7	0	6	54	7	7	5
7	3	2	1	31	4	7	3	7	6	1	1	7	7	1	0	55	7	7	6
8	3	2	2	0	5	0	0	8	6	1	2	8	7	1	1				
9	3	3	0	1	5	0	1	9	6	1	3	9	7	1	2				
10	3	3	1	2	5	0	2	10	6	1	4	10	7	1	3				
11	3	3	2	3	5	0	3	11	6	1	5	11	7	1	4				
12	3	4	0	4	5	0	4	12	6	2	0	12	7	1	5				
13	3	4	1	5	5	1	0	13	6	2	1	13	7	1	6				
14	3	4	2	6	5	1	1	14	6	2	2	14	7	2	0				
15	3	5	0	7	5	1	2	15	6	2	3	15	7	2	1				
16	3	5	1	8	5	1	3	16	6	2	4	16	7	2	2				
17	3	5	2	9	5	1	4	17	6	2	5	17	7	2	3				
18	3	6	0	10	5	2	0	18	6	3	0	18	7	2	4				
19	3	6	1	11	5	2	1	19	6	3	1	19	7	2	5				
20	3	6	2	12	5	2	2	20	6	3	2	20	7	2	6				
21	3	7	0	13	5	2	3	21	6	3	3	21	7	3	0				
22	3	7	1	14	5	2	4	22	6	3	4	22	7	3	1				
23	3	7	2	15	5	3	0	23	6	3	5	23	7	3	2				

Implementazione

2.1. Codice

Passiamo ora al codice VHDL che implementa il nostro divisore binario.

```
----------------------
-- cella elementare

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- def. entità CS

entity CS is
port (A,B_in,C,D_in:IN STD_LOGIC;B_out,D_out,P,S:OUT STD_LOGIC);
end CS;

-- def. architettura di CS (da qui in poi seguono
-- tutte le istruzioni del programma)

architecture arch_CS of CS is

signal Aneg,Bneg,Cneg,Dneg,P1,P2,P3,S1,S2,S3,S4,S5: std_logic;

-- NOTA: i ritardi utilizzati sono a scopo didattico
--       e si possono discostare dalla realtà

begin
        Aneg <= not(A) after 2 ns;
        Bneg <= not(B_in) after 2 ns;
        Cneg <= not(C) after 2 ns;
        Dneg <= not(D_in) after 2 ns;

        P1 <= (B_in and Aneg) after 2 ns;
        P2 <= (C and Aneg) after 2 ns;
        P3 <= (B_in and C) after 2 ns;
        P <= ((P1 or P2) or P3) after 2 ns;

        S1 <= (A and D_in) after 2 ns;
        S2 <= ((A and Bneg) and Cneg) after 2 ns;
        S3 <= ((A and B_in) and C) after 2 ns;
```

```
        S4 <= ((Aneg and B_in) and (Cneg and Dneg)) after 2 ns;
        S5 <= ((Bneg and Aneg) and (Dneg and C)) after 2 ns;
        S <= (((S1 or S2) or (S3 or S4)) or S5) after 2 ns;

        B_out <= B_in after 50 ps;
        D_out <= D_in after 50 ps;

end arch_cs;



----------
-- Porta Not

LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity porta_not is port(Input : IN STD_LOGIC; Output : OUT STD_LOGIC);
end porta_not;

architecture arch_porta_not of porta_not is
begin
        Output <= not(Input);
end arch_porta_not;



--------------------------------------------------------------------------
-- descrizione strutturale di una riga del restoring cellular array divider

LIBRARY ieee;
USE ieee.std_logic_1164.all;


entity riga is
port(A1,A2,A3,A4,B1in,B2in,B3in,B4in:IN STD_LOGIC;
B1out,B2out,B3out,B4out,S1,S2,S3,S4,Q:OUT STD_LOGIC);
end riga;

architecture arch_riga of riga is

component CS is
port (A,B_in,C,D_in:IN STD_LOGIC; B_out,D_out,P,S:OUT STD_LOGIC);
end component;

component porta_not is
port(Input:IN STD_LOGIC; Output:OUT STD_LOGIC);
end component;

signal W1,W2,W3,W4,P1,P2,P3,P4: STD_LOGIC;
signal C4: STD_LOGIC :='0';

-- NOTA: C1=P2 , C2=P3, C3=P4
--          D1in=P1, W1=D1out=D2in, W2=D2out=D3in, W3=D3out=D4in, W4=D4out
```

```
begin
        cmp1 : porta_not
        port map(P1,Q);
        cmp2 : CS
        port map(A1,B1in,P2,P1,B1out,W1,P1,S1);
        cmp3 : CS
        port map(A2,B2in,P3,W1,B2out,W2,P2,S2);
        cmp4 : CS
        port map(A3,B3in,P4,W2,B3out,W3,P3,S3);
        cmp5 : CS
        port map(A4,B4in,C4,W3,B4out,W4,P4,S4);

end arch_riga;



--------------------------------------------------------------------------
-- descrizione strutturale dell'intera macchina come composizione dei tre righe

LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity schema is
port(N0,N1,N2,N3,N4,N5,D0,D1,D2:IN STD_LOGIC;
R0,R1,R2,R3,Q0,Q1,Q2:OUT STD_LOGIC);
end schema;

architecture arch_schema of schema is

component riga is
port(A1,A2,A3,A4,B1in,B2in,B3in,B4in:IN STD_LOGIC;
B1out,B2out,B3out,B4out,S1,S2,S3,S4,Q:OUT STD_LOGIC);
end component;

signal X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,
X11,X12,X13,X14,X15,X16,X17,X18,X19,X20: STD_LOGIC;

signal D3: STD_LOGIC :='0';

-- NOTA: C1=P2 , C2=P3, C3=P4
--          D1in=P1, W1=D1out=D2in, W2=D2out=D3in, W3=D3out=D4in, W4=D4out

begin
        riga01: riga
        port map(N5,N4,N3,N2,D3,D2,D1,D0,X2,X4,X6,X8,X1,X3,X5,X7,Q2);
        riga02: riga
        port map(X3,X5,X7,N1,X2,X4,X6,X8,X10,X12,X14,X16,X9,X11,X13,
        X15,Q1);
        riga03: riga
        port map(X11,X13,X15,N0,X10,X12,X14,X16,X17,X18,X19,X20,R3,R2,
        R1,R0,Q0);

end arch_schema;
```

```
----------------------------------------------------------------
-- descrizione comportamentale dell'intera macchina con shifter

LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity schema_con_shifter is
port(num: IN STD_LOGIC_VECTOR(5 downto 0);
den: IN STD_LOGIC_VECTOR(2 downto 0);
resto: OUT STD_LOGIC_VECTOR(3 downto 0);
quoto: OUT STD_LOGIC_VECTOR(2 downto 0);
shift: OUT STD_LOGIC);
end schema_con_shifter;

architecture arch_shifter of schema_con_shifter is
component schema is
port(N0,N1,N2,N3,N4,N5,D0,D1,D2:IN STD_LOGIC;
R0,R1,R2,R3,Q0,Q1,Q2:OUT STD_LOGIC);
end component;

signal den2,num2: STD_LOGIC_VECTOR(5 downto 0);

begin
den2 <= (den(2),den(1),den(0),'0','0','0');

process(num,den2)
begin
if(num < den2) then
        num2(0) <= num(0);
        num2(1) <= num(1);
        num2(2) <= num(2);
        num2(3) <= num(3);
        num2(4) <= num(4);
        num2(5) <= num(5);
        shift <= '0';
else
        num2(0) <= num(2);
        num2(1) <= num(3);
        num2(2) <= num(4);
        num2(3) <= num(5);
        num2(4) <= '0';
        num2(5) <= '0';
        shift <= '1';
end if;
end process;

divisore: schema port map(num2(0),num2(1),num2(2),num2(3),num2(4),
num2(5),den(0),den(1),den(2),resto(0),resto(1),resto(2),resto(3), quoto(0),quoto(1),quoto(2));

end arch_shifter;
```

2.2. Risultati dei test bench

Per verificare che un componente si comporti secondo le specifiche di progetto, lo si sottopone ad un test bench ("banco di prova"):

- si forzano in ingresso stimoli controllati;
- si verifica che le uscite corrispondano.

La descrizione degli stimoli e delle corrispondenti uscite arricchisce la documentazione del progetto.

Questo procedimento in VHDL è realizzabile mediante un simulatore VHDL che consenta di eseguire i dovuti test bench.

Nel nostro caso ci limiteremo ad effettuare solo 3 test bench, in particolare osserveremo il funzionamento della cella elementare e del divisore con e senza shift.

Detto ciò passiamo alla scrittura dei nostri test bench.

```
---------------------------------------
-- Test Bench della cella elementare

LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity TestBench_CS is
end TestBench_CS;

architecture TestProva of TestBench_CS is

component CS is
port (A,B_in,C,D_in:IN STD_LOGIC;B_out,D_out,P,S:OUT STD_LOGIC);
end component;

signal signal1,signal2,signal3,signal4,signal5,signal6,signal7,
signal8: STD_LOGIC;

begin

        TB_CS: CS port map( A => signal1, B_in => signal2, C => signal3,
        D_in => signal4,B_out => signal5, D_out => signal6, P => signal7,
        S => signal8);

        signal1 <= '1','1' after 100 ns;
        signal2 <= '1','1' after 100 ns;
        signal3 <= '0','1' after 100 ns;
        signal4 <= '0','0' after 100 ns;

end TestProva;
```

```
-----------------------------------
-- Test Bench del divisore

LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity TestBench_Schema is
end TestBench_Schema;

architecture TestSchema of TestBench_Schema is

component schema is
port(N0,N1,N2,N3,N4,N5,D0,D1,D2:IN STD_LOGIC;
R0,R1,R2,R3,Q0,Q1,Q2:OUT STD_LOGIC);
end component;

signal signal1,signal2,signal3,signal4,signal5,signal6,signal7,
signal8,signal9,signal10,signal11,signal12,signal13,signal14,
signal15,signal16: STD_LOGIC;

begin

TB_Schema: schema port map( N0=>signal1, N1=>signal2, N2=>signal3,
N3=>signal4, N4=>signal5, N5=>signal6, D0=>signal7, D1=>signal8,
D2=>signal9, R0=>signal10, R1=>signal11, R2=>signal12, R3=>signal13,
Q0=>signal14, Q1=>signal15, Q2=>signal16 );

signal1 <= '1';
signal2 <= '0';
signal3 <= '0';
signal4 <= '0';
signal5 <= '1';
signal6 <= '0';
signal7 <= '0';
signal8 <= '1';
signal9 <= '1';

end TestSchema;
```

```
-----------------------------------------
-- Test Bench del divisore con shifter

LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity TestBench_Schema_Shifter is
end TestBench_Schema_Shifter;

architecture TestSchemaShifter of TestBench_Schema_Shifter is

component schema_con_shifter is
port(num: IN STD_LOGIC_VECTOR(5 downto 0);
```

```
den: IN STD_LOGIC_VECTOR(2 downto 0);
resto: OUT STD_LOGIC_VECTOR(3 downto 0);
quoto: OUT STD_LOGIC_VECTOR(2 downto 0);
shift: OUT STD_LOGIC);
end component;

signal signal1,signal2,signal3,signal4,signal5,signal6,
signal7,signal8,signal9,signal10,signal11,signal12,signal13,
signal14,signal15,signal16,signal17: STD_LOGIC;

begin

TB_SchemaShifter: schema_con_shifter port map( num(0)=>signal1, num(1)=>signal2, num(2)=>signal3,
num(3)=>signal4, num(4)=>signal5, num(5)=>signal6, den(0)=>signal7, den(1)=>signal8, den(2)=>signal9,
resto(0)=>signal10, resto(1)=>signal11, resto(2)=>signal12, resto(3)=>signal13, quoto(0)=>signal14,
quoto(1)=>signal15, quoto(2)=>signal16, shift=>signal17 );

signal1 <= '1','1' after 100 ns;
signal2 <= '1','0' after 100 ns;
signal3 <= '1','1' after 100 ns;
signal4 <= '1','0' after 100 ns;
signal5 <= '1','0' after 100 ns;
signal6 <= '1','1' after 100 ns;
signal7 <= '1','1' after 100 ns;
signal8 <= '1','1' after 100 ns;
signal9 <= '0','1' after 100 ns;

end TestSchemaShifter;
```

A questo punto non ci resta che osservare il risultato delle simulazioni.

In tal caso i test bench sono stati condotti utilizzando la versione FREE edition di **VHDL Simili** [1].

Nel caso della cella elementare si è fissato:

A = 1, B = 1, C = 0, D = 0

e dopo 100 ns

A = 1, B = 1, C = 1, D = 0.

Se teniamo conto delle espressioni di P e di S si deve ottenere:

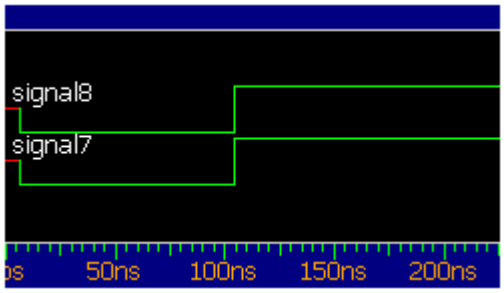
P = 0, S = 0

[1] VHDL Simili è un ambiente di sviluppo per VHDL e contiene un simulatore VHDL comportamentale. Symphony EDA (vedi http://www.symphonyeda.com) ne distribuisce gratuitamente una versione free (con limite di utilizzo basato sulla data). Qualora la licenza sia scaduta, è possibile rinnovarla seguendo le istruzioni suggerite dal programma stesso.

e dopo 100 ns

P = 1, S = 1.

Il risultato della simulazione da credito a quanto detto fin ora:



dove signal7 = P e signal8 = S.

Nel caso del divisore senza shift si è fissato:

$n_0 = 1$, $n_1 = 0$, $n_2 = 0$, $n_3 = 0$, $n_4 = 1$, $n_5 = 0$, $d_0 = 0$, $d_1 = 1$, $d_2 = 1$,

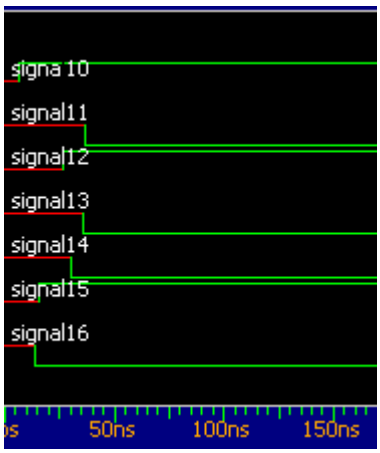
ovvero abbiamo diviso 17/64 con 6/8.

Se teniamo conto della tabella riportata nelle pagine precedenti il risultato deve essere:

quoto = 2/8 e resto = 5/64,

ovvero $q_0 = 0$, $q_1 = 1$, $q_2 = 0$, $r_0 = 1$, $r_1 = 0$, $r_2 = 1$, $r_3 = 0$.

Il risultato della simulazione da credito a quanto detto fin ora:



dove signal10 = r_0, signal11 = r_1, signal12 = r_2, signal13 = r_3, signal14 = $q_0 = 0$, signal15 = q_1, sihnal16 = q_2.

Nel caso del divisore con shift si è fissato:

$n_0 = 1, n_1 = 1, n_2 = 1, n_3 = 1, n_4 = 1, n_5 = 1, d_0 = 1, d_1 = 1, d_2 = 0,$

ovvero abbiamo diviso 63/64 con 3/8 e dopo 100 ns

$n_0 = 1, n_1 = 0, n_2 = 1, n_3 = 0, n_4 = 0, n_5 = 1, d_0 = 1, d_1 = 1, d_2 = 1,$

ovvero abbiamo diviso 37/64 con 7/8.

Se teniamo conto di quanto detto precedentemente il risultato deve essere

quoto = 21/8 e resto = 0 (divisione con shift),

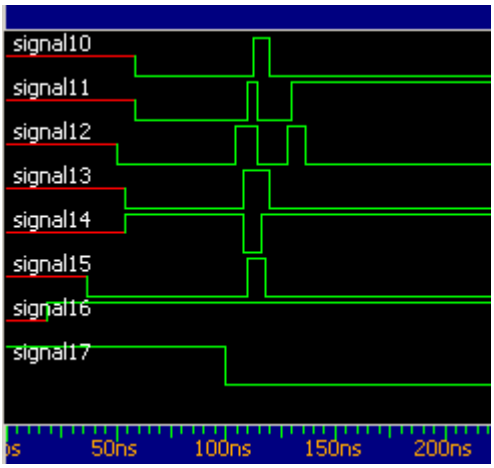
ovvero $q_0 = 1, q_1 = 0, q_2 = 1, r_0 = 0, r_1 = 0, r_2 = 0, r_3 = 0$, shift = 1

e dopo 100 ns

quoto = 5/8 e resto = 2/64 (divisione senza shift – vedi tabella),

ovvero $q_0 = 1, q_1 = 0, q_2 = 1, r_0 = 0, r_1 = 1, r_2 = 0, r_3 = 0$, shift = 0.

Il risultato della simulazione da credito a quanto fatto fin ora:



dove signal10 = r_0, signal11 = r_1, signal12 = r_2, signal13 = r_3, signal14 = q_0 = 0,
signal15 = q_1, sihnal16 = q_2, signal17 = shift.

Si notino gli impulsi presenti nella figura precedente, i quali sono dovuti alla inevitabile presenza di alee.