Alma Mater Studiorum · Università di Bologna

Elettronica dei Sistemi Digitali M

Sviluppo di Divisore tra numeri interi senza segno a 16 bit

Confronto tra implementazione seriale e combinatoria

Autori:
Riccardo Gaspari
0001139139
Edoardo Casadei
0001131567

Anno Accademico 2023/2024

Indice

1 Descrizione RTL

1.1 Divisore combinatorio

La soluzione combinatoria viene scelta in caso di necessità di minimizzare la latenza, a costo di maggiore area occupata rispetto alla soluzione seriale. A tal fine vengono utilizzate le funzioni "/" e "*rem*" rese disponibili dalla libreria *"IEEE.numeric_std.all"* per implementare il calcolo della divisione in un singolo ciclo di clock; per questo motivo la latenza è uguale a 1 Tck, mentre il throughput risulta essere uguale alla frequenza di clock. Questa soluzione, necessitando di una rete combinatoria complessa per l'implementazione del calcolo, porta a una frequenza massima di funzionamento molto inferiore rispetto alla soluzione seriale. Per la realizzazione sono necessari 64 bit di registri con reset asincrono attivo basso, rispettivamente metà per gli ingressi (operandi a 16 bit), e metà per le uscite (OUT_DIV e OUT_REM a 16 bit).

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity divider is
6       port (
7          clk        : in std_logic;
8          resetn   : in std_logic;
9          IN_A     : in  std_logic_vector(15 downto 0);
10         IN_B     : in  std_logic_vector(15 downto 0);
11         OUT_DIV  : out std_logic_vector(15 downto 0);
12         OUT_REM  : out std_logic_vector(15 downto 0)
13      );
14 end divider;
15
16 architecture behavioral of divider is
17 signal IN_A_REG: std_logic_vector (15 downto 0);
18 signal IN_B_REG: std_logic_vector (15 downto 0);
19 signal OUT_DIV_INT : std_logic_vector (15 downto 0);
20 signal OUT_REM_INT : std_logic_vector (15 downto 0);
21
22 begin
23
24 --sample input
25 process(clk, resetn)
26 begin
27   if(resetn='0') then
28     IN_A_REG <= (others =>'0');
29     IN_B_REG <= (others =>'0');
30   elsif (clk'event and clk='1') then
31     IN_A_REG <= IN_A;
32     IN_B_REG <= IN_B;
33   end if;
34 end process;
35 --sample output
36 process(clk, resetn)
37 begin
38   if (resetn='0') then
39     OUT_DIV <= (others =>'0');
40     OUT_REM <= (others =>'0');
41   elsif (clk'event and clk='1') then
42     OUT_DIV <= OUT_DIV_INT;
43     OUT_REM <= OUT_REM_INT;
44   end if;
45 end process;
46
47 OUT_DIV_INT <= std_logic_vector (unsigned(IN_A_REG) / unsigned(IN_B_REG));
48 OUT_REM_INT <= std_logic_vector (unsigned(IN_A_REG) rem unsigned(IN_B_REG));
49
50 end behavioral;
```

1.2 Divisore seriale

La soluzione seriale viene scelta in caso di necessità di minimizzazione di area; inoltre questa implementazione ci permette di inserire uno stadio combinatorio molto più semplice rispetto alla soluzione precedente, garantendo una frequenza massima di funzionamento più alta rispetto al caso combinatorio. L'algoritmo implementato è quello per sottrazioni successive con reintegro:

1. Estensione dividendo a N+(N-1) (31) bit (zero padding);
2. Sottrazione tra MSBs del dividendo esteso e divisore:
   - *segno positivo*: LSB(quoziente)=1, resto parziale = risultato sottrazione;
   - *segno negativo*: LSB(quoziente)=0, resto parziale invariato
3. Shift a sinistra del quoziente (eccetto l'ultima iterazione);
4. *Iterazioni successive*: ripetizione procedimento tra la concatenazione di N-1 bit del resto parziale e MSB successivo del dividendo e il divisore.

Come mostrato nella figura 1, la concatenazione descritta nel punto 4 viene realizzata con uno shift verso sinistra del registro di campionamento di ingresso e il registro di resto parziale, considerato come estensione più significativa del dividendo.

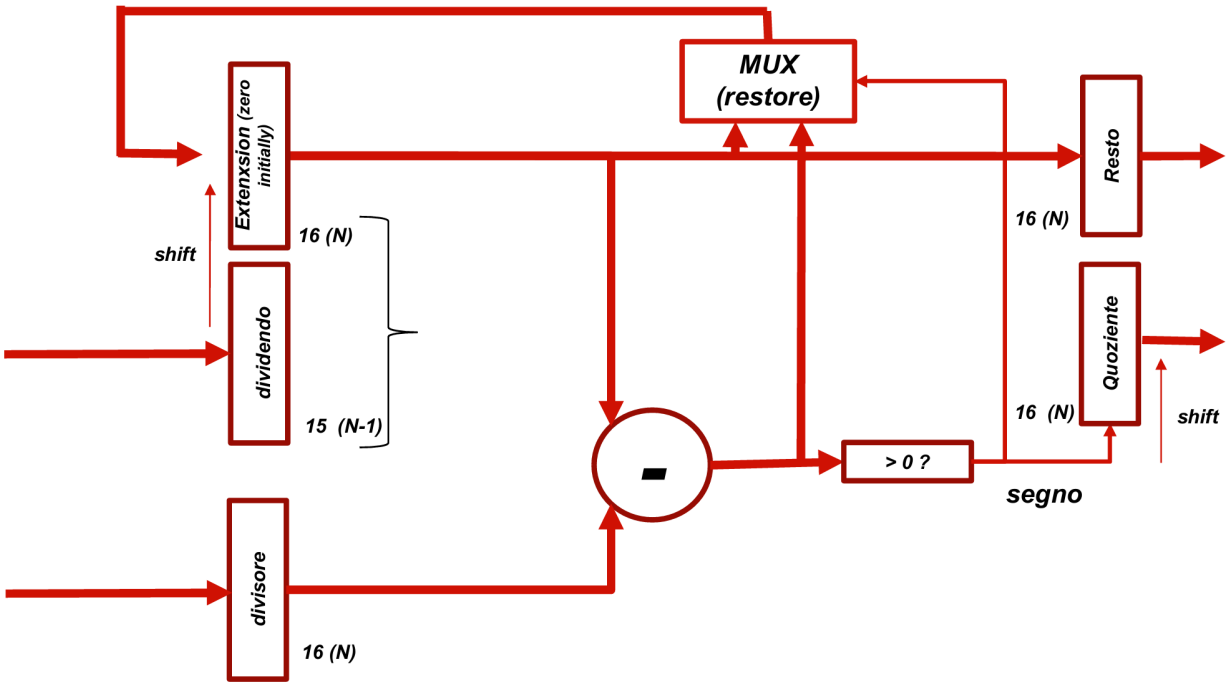


Figura 1: Datapath divisore seriale

Per rimanere coerenti col datapath fornito nella documentazione di progetto, è richiesta una logica di controllo basata su una macchina di Mealy a 4 stati. Tuttavia, si è pensato ad una implementazione a 3 stati con un numero minore i registri, che viene successivamente approfondita nel capitolo 4. La FSM della soluzione canonica è mostrata in figure 2 e 3 (realizzata da Quartus), e può essere riassunta come segue:

- IDLE: attesa del valore alto dell'ingresso start per campionare gli ingressi e avviare la computazione;
- ELABORATION: calcolo risultato attraverso sottrazioni successive con reintegro, di durata complessiva di 16 Tclock;
- WAITING: singolo periodo per attendere la propagazione del segnale di resto al registro di uscita;
- FINISHED: Termine della computazione, riporto dei risultati in uscita. Se start=1, viene saltato lo stato di idle iniziando direttamente una nuova elaborazione.

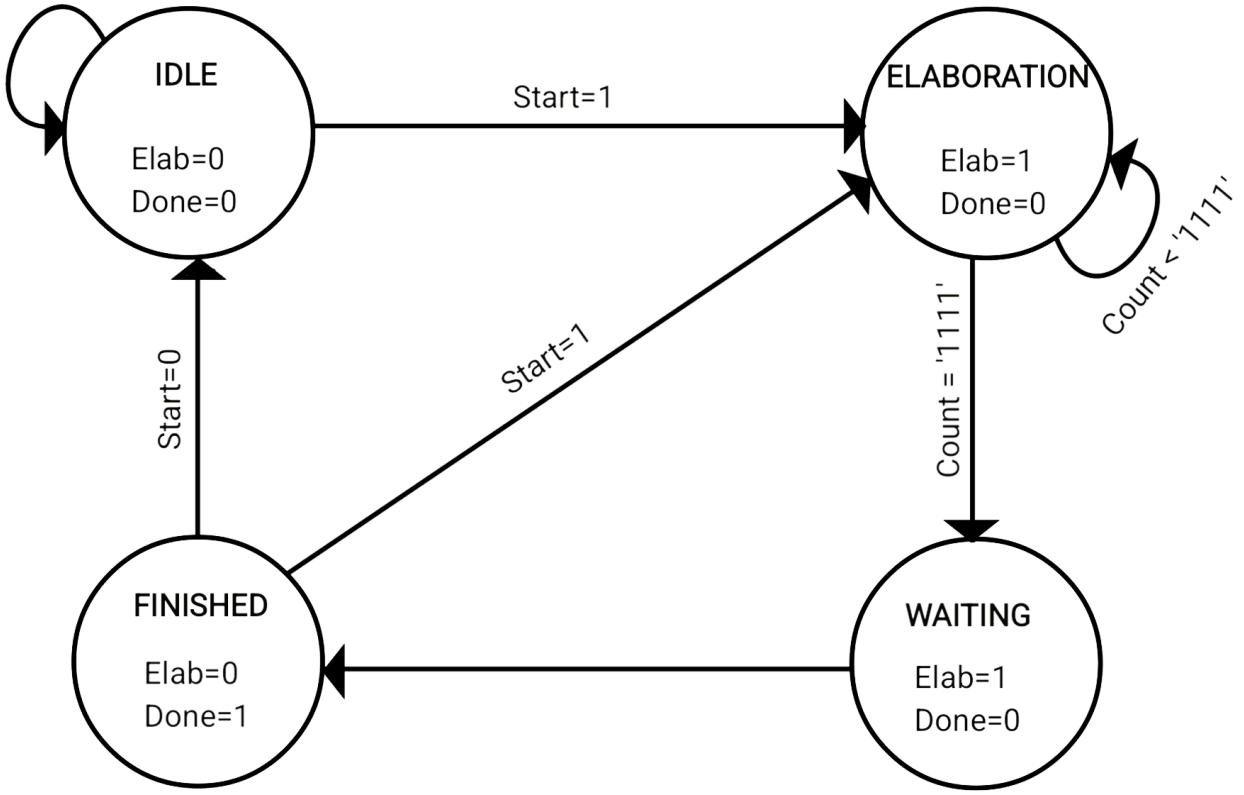


Figura 2: Macchina a stati finiti

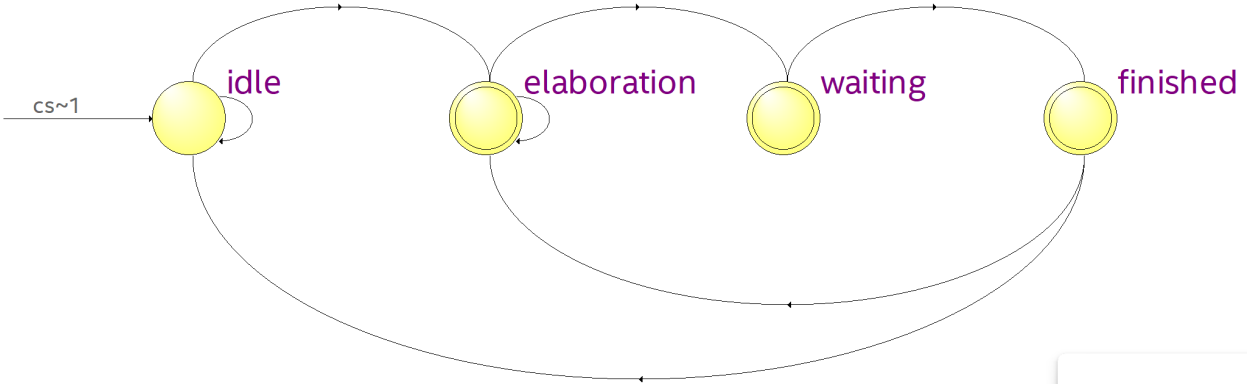


Figura 3: FSM fornita da Quartus

Complessivamente sono richiesti 87 bit di registri con reset asincrono attivo basso, rispettivamente 16 per il campionamento del divisore, 31 (15+16) per campionamento ed estensione del dividendo, 32 per i registri di uscita, 4 per il contatore e 4 per la macchina a stati. I registri di campionamento di ingresso sono con enable (start).

Per mantenere la struttura canonica della rete digitale fornita, si è ottenuta una latenza di 1 Tck superiore rispetto alla soluzione a 3 stati precedentemente adottata, e un conseguente peggioramento del throughput: pertanto la latenza risulta $17\,T_{ck}$ e il throughput di $\frac{1}{18} f_{ck}$.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity divider_serial is
7   port (
8   clk : in  std_logic;
9   resetn : in  std_logic;
10  IN_A     : in  std_logic_vector(15 downto 0); -- dividend
11  IN_B     : in  std_logic_vector(15 downto 0); -- divider
12  OUT_DIV  : out std_logic_vector(15 downto 0); -- quotient
13  OUT_REM  : out std_logic_vector(15 downto 0); -- reminder
14  start    : in  std_logic;
15  elab     : out std_logic;
16  done     : out std_logic
17  );
18  end divider_serial;
19
20 architecture behavioral of divider_serial is
21
22 signal in_a_reg: std_logic_vector (14 downto 0);
23 signal in_b_reg: std_logic_vector (15 downto 0);
24 signal a_pad_d, a_pad_q: std_logic_vector (15 downto 0);
25 signal div_d, div_q, rem_d, rem_q: std_logic_vector(15 downto 0);
26 signal shift, reset_cont: std_logic;
27
28 signal sub: std_logic_vector(16 downto 0);
29
30 type stato is (idle, elaboration, waiting, finished);
31 signal cs, ns: stato;
32
33 signal cont: std_logic_vector(3 downto 0);
34
35
36 component COUNTER
37      generic(cw: natural:=16);
38      port (
39         CLK    : in  std_logic;
40         RESET  : in  std_logic;
41         ENABLE : in std_logic;
42         COUNT  : out std_logic_vector(cw-1 downto 0)
43       );
44 end component;
45
46 begin
47
```

```
48    reset_cont <= not(resetn and shift);
49
50     cont0 : COUNTER
51    generic map(cw => 4)
52    port map (
53                    CLK   => clk,
54                    RESET => reset_cont,
55              ENABLE => '1',
56                    COUNT => cont
57    );
58
59  -- Registro di 15 bit con enable (start) e reset asincrono di campionamento divisore (escluso MSB) con rete di shift
60    process(clk, resetn)
61    begin
62      if resetn = '0' then
63        in_a_reg <= (others => '0');
64      elsif clk'event and clk='1' then
65        if start='1' then
66          in_a_reg <= IN_A(14 downto 0);
67        elsif shift='1' then
68          in_a_reg <= in_a_reg(13 downto 0) & '0';
69        end if;
70      end if;
71    end process;
72
73  -- Registro con enable (start) e reset asincrono di campionamento MSB dell'ingresso e shift con registro di campionamento
       del divisore (in_a_reg)
74    process(clk, resetn)
75    begin
76      if resetn = '0' then
77        a_pad_q <= (others => '0');
78      elsif clk'event and clk='1' then
79        if start='1' then
80          a_pad_q <= "000000000000000" & IN_A(15);
81        elsif shift='1' then
82          a_pad_q <= a_pad_d(14 downto 0) & in_a_reg(14);
83        else
84          a_pad_q <= a_pad_d;
85        end if;
86      end if;
87    end process;
88
89  -- Collegamento uscita registro di appoggio a registro di uscita
90    rem_d <= a_pad_q;
91
92  -- Registro con enable (start) e reset asincrono di campionamento dividendo
93    process(clk, resetn)
94    begin
95      if resetn = '0' then
96        in_b_reg <= (others => '0');
97      elsif clk'event and clk='1' and start='1' then
98        in_b_reg <= IN_B;
99      end if;
100   end process;
101
102 -- registro di stato di gestione FSM
103   process(clk, resetn)
104   begin
105     if resetn = '0' then
106       cs <= idle;
107     elsif clk'event and clk='1' then
108       cs <= ns;
109     end if;
110   end process;
111
112 -- Registro di campionamento dell'uscita OUT_REM
113   process(clk, resetn)
114   begin
115     if resetn='0' then
116       rem_q <= (others => '0');
117     elsif clk'event and clk='1' then
118       rem_q <= rem_d;
119     end if;
120   end process;
121
122 -- Assegnamento uscita OUT_REM
123   OUT_REM <= rem_q;
124
125 -- Registro di campionamento dell'uscita OUT_DIV
126   process(clk, resetn)
127   begin
128     if resetn='0' then
129       div_q <= (others => '0');
130     elsif clk'event and clk='1' then
131       div_q <= div_d;
132     end if;
133   end process;
134
135 -- Assegnamento uscita OUT_DIV
136   OUT_DIV <= div_q;
137
138 -- Rete combinatoria algoritmo di divisione
```

```
139   process(cs, start, div_q, a_pad_q, in_b_reg, sub)
140   begin
141     case cs is
142       when elaboration =>
143         -- Shift verso sx di div_q
144         div_d(15 downto 1) <= div_q(14 downto 0);
145         -- Calcolo sottrazione con segno
146         sub <= signed('0' & a_pad_q) - signed('0' & in_b_reg);
147         -- Assegnamento LSB a seconda dei casi + valutazione resto parziale divisione
148         -- oss.: lo shift di rem viene fatto nel registro
149         if sub(16) = '0' then -- sub >= 0
150           div_d(0) <= '1';
151           a_pad_d <= sub(15 downto 0);
152         else -- sub < 0
153           div_d(0) <= '0';
154           a_pad_d <= a_pad_q;
155         end if;
156       when waiting =>
157         -- Permanenza dei valori in attesa della propagazione del regnale a_pad_q in rem_q
158         div_d <= div_q;
159         a_pad_d <= a_pad_q;
160         sub <= (others => '0');
161       when others =>
162         div_d <= (others => '0');
163         a_pad_d <= (others => '0');
164         sub <= (others => '0');
165       end case;
166   end process;
167 
168 -- Assegnamento uscite elab e done
169   process(cs)
170   begin
171     elab <= '0';
172     done <= '0';
173     case cs is
174       when elaboration =>
175         elab <= '1';
176       when waiting =>
177         elab <= '1';
178       when finished =>
179         done <= '1';
180       when others =>
181         elab <= '0';
182         done <= '0';
183       end case;
184   end process;
185 
186 -- Rete combinatoria di gestione FSM (si rimanda al dds nella relazione di progetto)
187   process(cs, start, cont)
188   begin
189     case cs is
190       when idle =>
191         shift <= '0';
192         if start = '1' then
193           ns <= elaboration;
194         else
195           ns <= idle;
196         end if;
197       when elaboration =>
198         if cont < "1111" then
199           shift <= '1';
200         else
201           shift <= '0';
202         end if;
203         if cont = "1111" then
204           ns <= waiting;
205         else
206           ns <= cs;
207         end if;
208       when waiting =>
209         ns <= finished;
210         shift <= '0';
211       when finished =>
212         shift <= '0';
213         if start = '1' then
214           ns <= elaboration;
215         else
216           ns <= idle;
217         end if;
218       when others =>
219         shift <= '0';
220         ns <= idle;
221       end case;
222   end process;
223 
224 end behavioral;
```

2 Verifica funzionale

La differenza sostanziale tra le testbench di divisore combinatorio e seriale sta nella frequenza con cui sono applicabili gli ingressi: per quanto riguarda il combinatorio è possibile applicare ingressi diversi ogni ciclo di clock (figura 4), mentre per il seriale al più ogni 18 Tck. Abbiamo avuto premura di utilizzare valori di ingresso che portassero almeno intuitivamente a un maggior numero di transizioni delle celle combinatorie e dei registri, e quindi il coefficiente di attività, in maniera tale da avere una stima più affidabile del consumo di potenza.

2.1 Divisore combinatorio (codice e simulazione)

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_arith.all;
5
6  entity Testbench is
7  end Testbench;
8
9  Architecture A of Testbench is
10
11 component divider is
12      port (
13         clk         : in std_logic;
14         resetn   : in std_logic;
15         IN_A     : in  std_logic_vector(15 downto 0);
16         IN_B     : in  std_logic_vector(15 downto 0);
17         OUT_DIV  : out std_logic_vector(15 downto 0);
18         OUT_REM  : out std_logic_vector(15 downto 0)
19      );
20 end component;
21
22   signal CLK, RESETN: std_logic;
23   signal IN_A, IN_B: std_logic_vector(15 downto 0);
24   signal OUT_DIV, OUT_REM: std_logic_vector(15 downto 0);
25
26   constant clk_period : time := 73 ns;
27
28 begin
29
30   UUT : divider
31         port map (clk, resetn, IN_A, IN_B, OUT_DIV, OUT_REM);
32
33    xsclock_engine : process
34     begin
35       clk <= '0';
36       wait for clk_period/2;
37       clk <= '1';
38       wait for clk_period/2;
39     end process;
40
41     reset_engine : process
42       begin
43   resetn <= '1';
44         wait for clk_period;
45         resetn <='0';
46         wait for clk_period;
47         resetn <= '1';
48         wait;
49     end process;
50
51     input_engine: process
52   begin
53     wait for clk_period;
54     IN_A <= conv_std_logic_vector(34562, 16);
55     IN_B <= conv_std_logic_vector(29536, 16);
56     wait for clk_period;
57     IN_A <= conv_std_logic_vector(61283, 16);
58     IN_B <= conv_std_logic_vector(11342, 16);
59     wait for clk_period;
60     IN_A <= conv_std_logic_vector(59876, 16);
61     IN_B <= conv_std_logic_vector(13, 16);
62     wait for clk_period;
63     IN_A <= conv_std_logic_vector(26229, 16);
64     IN_B <= conv_std_logic_vector(9580, 16);
65     wait for clk_period;
66     IN_A <= conv_std_logic_vector(48543, 16);
67     IN_B <= conv_std_logic_vector(2, 16);
68     wait for clk_period;
69     IN_A <= conv_std_logic_vector(27632, 16);
70     IN_B <= conv_std_logic_vector(65535, 16);
71     wait for clk_period;
72     IN_A <= conv_std_logic_vector(65535, 16);
```

```
    IN_B <= conv_std_logic_vector(1, 16);
    wait for clk_period;
    IN_A <= conv_std_logic_vector(28465, 16);
    IN_B <= conv_std_logic_vector(9422, 16);
    wait for clk_period;
    IN_A <= conv_std_logic_vector(63231, 16);
    IN_B <= conv_std_logic_vector(3, 16);
    wait for clk_period;
    IN_A <= conv_std_logic_vector(32490, 16);
    IN_B <= conv_std_logic_vector(1907, 16);
    wait for clk_period;
    IN_A <= conv_std_logic_vector(45682, 16);
    IN_B <= conv_std_logic_vector(8, 16);
  end process;

end A;
```

Come si può osservare in figura, il segnale viene correttamente campionato una volta tornato alto il reset asincrono e l'uscita corretta è riportata in uscita il clock successivo.

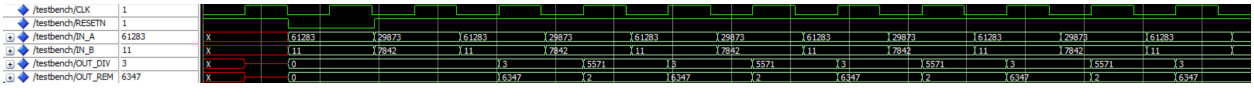


Figura 4: Simulazione divisore combinatorio

2.2 Divisore seriale (codice e simulazione)

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Testbench is
end Testbench;

Architecture A of Testbench is

 component divider_serial is
 port (
 clk : in  std_logic;
 resetn : in  std_logic;
 IN_A     : in  std_logic_vector(15 downto 0); -- dividend
 IN_B     : in  std_logic_vector(15 downto 0); -- divider
 OUT_DIV  : out std_logic_vector(15 downto 0); -- quotient
 OUT_REM  : out std_logic_vector(15 downto 0); -- reminder
 start    : in  std_logic;
 elab     : out std_logic;
 done     : out std_logic
 );
 end component;


  signal clk, resetn, start, elab, done: std_logic;
  signal IN_A, IN_B: std_logic_vector(15 downto 0);
  signal OUT_DIV, OUT_REM: std_logic_vector(15 downto 0);

  constant clk_period : time := 8 ns;

begin

  UUT : divider_serial
        port map (clk, resetn, IN_A, IN_B, OUT_DIV, OUT_REM, start, elab, done);

   xsclock_engine : process
    begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
    end process;

    reset_engine : process
      begin
  resetn <= '1';
        wait for clk_period;
        resetn <='0';
        wait for clk_period;
        resetn <= '1';
        wait;
    end process;

    input_engine: process
```

```
56    begin
57      wait for 2*clk_period;
58      start <= '1';
59      IN_A <= conv_std_logic_vector(34562, 16);
60      IN_B <= conv_std_logic_vector(29536, 16);
61      wait for clk_period;
62      start <= '0';
63      wait for 17*clk_period;
64      start <= '1';
65      IN_A <= conv_std_logic_vector(61283, 16);
66      IN_B <= conv_std_logic_vector(11342, 16);
67      wait for clk_period;
68      start <= '0';
69      wait for 18*clk_period;
70      start <= '1';
71      IN_A <= conv_std_logic_vector(59876, 16);
72      IN_B <= conv_std_logic_vector(13, 16);
73      wait for clk_period;
74      start <= '0';
75      wait for 17*clk_period;
76      start <= '1';
77      IN_A <= conv_std_logic_vector(26229, 16);
78      IN_B <= conv_std_logic_vector(9580, 16);
79      wait for clk_period;
80      start <= '0';
81      wait for 17*clk_period;
82      start <= '1';
83      IN_A <= conv_std_logic_vector(48543, 16);
84      IN_B <= conv_std_logic_vector(2, 16);
85      wait for clk_period;
86      start <= '0';
87      wait for 17*clk_period;
88      start <= '1';
89      IN_A <= conv_std_logic_vector(27632, 16);
90      IN_B <= conv_std_logic_vector(65535, 16);
91      wait for clk_period;
92      start <= '0';
93      wait for 17*clk_period;
94      start <= '1';
95      IN_A <= conv_std_logic_vector(65535, 16);
96      IN_B <= conv_std_logic_vector(1, 16);
97      wait for clk_period;
98      start <= '0';
99      wait for 17*clk_period;
100     start <= '1';
101     IN_A <= conv_std_logic_vector(28465, 16);
102     IN_B <= conv_std_logic_vector(9422, 16);
103     wait for clk_period;
104     start <= '0';
105     wait for 17*clk_period;
106     start <= '1';
107     IN_A <= conv_std_logic_vector(63231, 16);
108     IN_B <= conv_std_logic_vector(3, 16);
109     wait for clk_period;
110     start <= '0';
111     wait for 17*clk_period;
112     start <= '1';
113     IN_A <= conv_std_logic_vector(32490, 16);
114     IN_B <= conv_std_logic_vector(1907, 16);
115     wait for clk_period;
116     start <= '0';
117     wait for 17*clk_period;
118     start <= '1';
119     IN_A <= conv_std_logic_vector(45682, 16);
120     IN_B <= conv_std_logic_vector(8, 16);
121     wait for clk_period;
122     start <= '0';
123     wait for 15*clk_period;
124   end process;
125
126 end A;
```

Nelle figure seguenti sono mostrati due esempi di transizioni del segnale di start nelle condizioni per cui si passi da FINISHED a ELABORATION e da IDLE a ELABORATION, per verificare la correttezza della FSM. L'uscita è libera di commutare durante l'elaborazione, e il segnale di uscita viene considerato valido quando il flag done è alto.

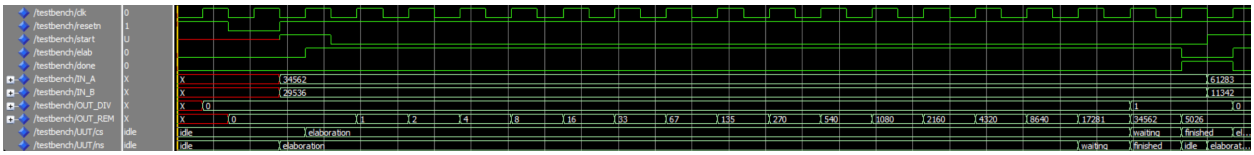


Figura 5: Simulazione divisore sequenziale: ingressi applicati a distanza di 17 Tck (saltato stato di idle)

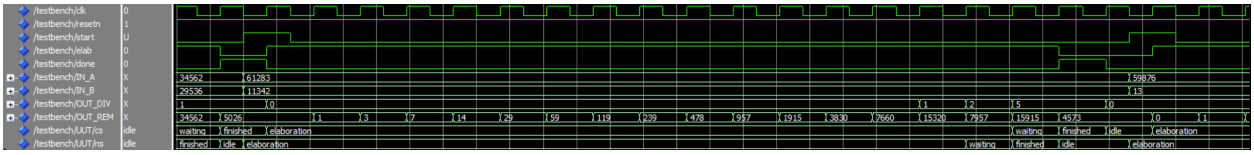


Figura 6: Simulazione divisore sequenziale: ingressi applicati a distanza di 18 Tck (passaggio per idle)

3 Confronto soluzioni FPGA e SC

3.1 Analisi su FPGA

Per la compilazione di entrambi i divisori è stata utilizzata la FPGA EP4CE6E22C9L, appartenente alla famiglia Cyclone IV-E. L'ambiente di sviluppo Quartus è di default impostato in modalità bilanciata velocità/consumo, per cui il tool cerca di minimizzare dove possibile la potenza di leakage senza ridurre le prestazioni (tipicamente nei cammini non critici).

3.1.1 Timing analisys su FPGA

Nelle figure seguenti sono riportati i valori di timing analisys ottenuti da simuazione su Quartus. Il fine dello studio è verificare il periodo minimo di funzionamento del circuito, per poter procedere a valutare il consumo di potenza alla massima frequenza raggiungibile (caso peggiore). Come facilmente intuibile, la necessità di effettuare l'operazione di divisione in un unico ciclo di clock comporta un numero di logical elements molto superiore al caso seriale, e inoltre una frequenza massima di funzionamento molto minore ($\approx 14MHz$ contro i $\approx 125MHz$). Per quanto riguarda il controllo del rispetto dei vincoli di setup e hold, la teoria ci suggerisce che la condizione con la maggior probabilità di violazione del tempo di setup è quella ad alta temperatura (che riduce la mobilità e quindi la corrente) e processo slow, mentre vale il caso opposto per il tempo di hold (Fast, $0°C$). La ragione di ciò è legata, per il setup, all'aumento dei tempi di propagazione del cammino critico della rete combinatoria e, per l'hold, alla riduzione del tempo di propagazione del cammino più rapido della stessa rete.

Period (ns)	Freq. (MHz)	Latency (ns)	Throughput (MS/s)	Max freq. (MHz) (w.c)	Setup check		Hold check		Total LE	Total registers	Total pins
					P, V,T	Slack (ns)	P, V,T	Slack (ns)			
78	12,8205	78	12,82051282	13,97	slow, 1V, 85°C	6,436	fast, 1V, 0°C	0,572	586	64	66
74	13,5135	74	13,51351351	13,56	slow, 1V, 85°C	0,228	fast, 1V, 0°C	0,74	587	64	66
73	13,6986	73	13,69863014	13,71	slow, 1V, 85°C	0,055	fast, 1V, 0°C	0,803	586	64	66
72	13,8889	72	13,88888889		slow, 1V, 85°C	-0,948					

Period (ns)	Freq. (MHz)	Latency (ns)	Throughput (MS/s)	Max freq. (MHz) (w.c)	Setup check		Hold check		Total LE	Total registers	Total pins
					P, V,T	Slack (ns)	P, V,T	Slack (ns)			
10	100	170	5,555555556	124,05	slow, 1V, 85°C	1,939	fast, 1V, 0°C	0,279	142	87	69
9	111,111	153	6,172839506	119,67	slow, 1V, 85°C	0,644	fast, 1V, 0°C	0,281	141	87	69
8	125	136	6,944444444	133,19	slow, 1V, 85°C	0,492	fast, 1V, 0°C	0,281	140	87	69
7	142,857	119	7,936507937		slow, 1V, 85°C	-0,441					

3.1.2 Power analisys su FPGA

L'analisi di potenza è stata effettuata, come anticipato, alla massima frequenza di funzionamento, in maniera da avere una valutazione di caso peggiore per quanto concerne la potenza dinaminca. Le casistiche analizzate sono:

1. Toggle rate fissato e temperatura di giunzione auto determinata;
2. Toggle rate fissato e temperatura di giunzione imposta a 85°;

3. Toggle rate determinato da simulazione su Modelsim e temperatura di giunzione auto determinata. Il vantaggio di questo tipo di analisi è quello di fornire risultati più affidabili rispetto a una stima vectorless con coefficiente di attività fissato.

Tutti i casi non prevedono l'utilizzo di un dissipatore di calore.
Le principali osservazioni sui risultati ottenuti sono:

- Dipendenza della potenza dinamica dalla frequenza di funzionamento;
- Dipendenza della potenza statica dalla temperatura;
- Il toggle rate influenza direttamente il consumo di potenza dinamica, quindi sia di celle combinatorie che registri;
- Migliore energia per svolgere l'operazione della soluzione combinatoria rispetto quella seriale (legato alla bassa latenza del combinatorio);
- Consumo delle celle combinatorie della soluzione seriale molto inferiore rispetto quella combinatoria (rete combinatoria più semplice);
- Minor consumo di clock control e register cell della soluzione combinatoria (legato al minor numero di registri).

	T_{CK} (ns)	freq (MHz)	LE/Ltot	toggle rate α/T_{CK}		Temperature (°C)		Summary (mW)				By clock domain (mW)	By block type (mW)				Power by clock domain/freq	Energia per svolgere operazione En/op	Power estimation confidence
					computed average toggle rate (Million of Transitions/s)	auto compute junction temperature	Tjun	Total	Core dyn	Core static	I/O		combinational cell	clock control	register cell	I/O	(mW/MHz)	(mW/MHz)	
Tck,min,comb	73	13,699	586/6272	vectorless, default toggle rate for I/O signals = 10%	10	si	27,1	76,23	1,45	41,23	33,55	1,45	1,03	0,18	0,21	15,58	0,1059	0,1059	low
Tck,min,comb	73	13,699	586/6272	vectorless, default toggle rate for I/O signals = 10%	10	no	85	81,97	1,45	46,97	33,55	1,45	1,03	0,18	0,21	15,6	0,1059	0,1059	low
Tck,min,comb	73	13,699	586/6272	from simulation results	50,8	si	27,2	81,49	2,55	41,23	37,7	2,55	1,58	0,19	0,63	19,86	0,1862	0,1862	high

	T_{CK} (ns)	freq (MHz)	LE/Ltot	toggle rate α/T_{CK}		Temperature (°C)		Summary (mW)				By clock domain (mW)	By block type (mW)				Power by clock domain/freq	Energia per svolgere operazione En/op	Power estimation confidence
					computed average toggle rate (Million of Transitions/s)	auto compute junction temperature	Tjun	Total	Core dyn	Core static	I/O		combinational cell	clock control	register cell	I/O	(mW/MHz)	(mW/MHz)	
Tck,min,seriale	8	125	85/6272	vectorless, default toggle rate for I/O signals = 10%	10	si	27,4	89,63	2,87	41,21	45,56	2,87	0,3	1,35	0,92	27,86	0,0230	0,3903	low
Tck,min,seriale	8	125	85/6272	vectorless, default toggle rate for I/O signals = 10%	10	no	85	95,29	2,87	46,87	45,56	2,87	0,3	1,35	0,92	27,86	0,0230	0,3903	low
Tck,min,seriale	8	125	85/6272	from simulation results	16,6	si	27,3	83,92	3,03	41,2	39,69	3,03	0,49	1,35	1,1	21,78	0,0242	0,4121	high

3.2 Sintesi su Standard Cell

La sintesi su Standard Cell è stata effettuata attraverso la libreria di celle standard NANDGATE-Opencell, la quale contiene 115 celle full custom con processo a 45nm. Il tool di sintesi converte il codice RTL in una netlist che cerca di soddisfare i vincoli di timing minimizzando quanto possibile l'area occupata e la potenza di leakage dissipata. A differenza del caso su FPGA, nella sintesi su SC non viene generato l'albero di clock, per cui la clock uncertainty deve essere specificata esplicitamente. Sono riportate a seguire le tabelle della sintesi di divisore combinatorio e seriale.

Le osservazioni principali riguardo i risultati sono:

- Aumentando la frequenza di funzionamento diminuisce in numero di livelli di logica della rete, al fine di ridurre il più possibile il tempo di propagazione lungo il cammino critico;
- L'architettura del divisore combinatorio è quella di un divisore realizzato con Carry-look-ahead adder, spostandosi verso una versione più rapida allo scendere del Tck;
- L'ottimizzazione privilegiata sul divisore seriale è quella di Area dove possibile, spontandosi verso Area/Speed in caso di vincoli più stringenti sulla frequenza di funzionamento.
- La potenza dinamica sale all'aumentare della frequenza (come atteso);
- La potenza statica sale all'aumentare della frequenza, probabilmente legato al fatto che il tool cerca di realizzare schematiche che soddisfino il vincolo di setup a prezzo di un peggioramento dal punto di vista della corrente di leakage;
- Il divisore combinatorio occupa molta più area, è più lento in termini di frequenza massima, dissipa maggior potenza statica (legato alla maggior complessità della rete combinatoria), ma consuma meno energia per svolgere l'operazione (legato alla bassa latenza).

clock uncertainty (ns)	Period (ns)	Freq. MHz	Latency in numero di cicli di clock	Throughput (MS/s)	slack (ns)	Area (um^2)	Area (Kgate,eq)	Leakage (uW)	Dynamic Power (uW)	Total Power (uW)	DynP/freq (uW/MHz)	Energia per svolgere l'operazione (uW/MHz)	n. livelli di logica nel critical path	ottimizzazione macro-celle
							area nand2 = 0,798 um^2							
0,2	14	71,43	14	71,43	0	2710,810	3,397	45,120	85,610	130,730	1,19854	1,199	122	cla
0,2	13	76,92	13	76,92	0	3338,030	4,183	55,220	108,190	163,400	1,40647	1,406	117	cla2/cla
0,2	12	83,33	12	83,33	0	3342,560	4,189	54,920	121,990	176,910	1,46388	1,464	94	cla2
0,2	11	90,91	11	90,91	0	3449,750	4,323	57,470	138,370	195,840	1,52207	1,522	87	cla2
0,2	10	100,00	10	100,00	0	3622,920	4,540	60,950	152,830	213,780	1,5283	1,528	87	cla2
0,2	9	111,11	9	111,11	-0,07		0,000				0	0,000		

clock uncertainty (ns)	Period (ns)	Freq. MHz	Latency in numero di cicli di clock	Throughput (MS/s)	slack (ns)	Area (um^2)	Area (Kgate,eq)	Leakage (uW)	Dynamic Power (uW)	Total Power (uW)	DynP/freq (uW/MHz)	Energia per svolgere l'operazione (uW/MHz)	n. livelli di logica nel critical path	ottimizzazione macro-celle
							area nand2 = 0,798 um^2							
0,2	9	111,11	153	6,17	3,08	645,050	0,808	7,020	59,130	66,150	0,532	9,047	20	area
0,2	5	200,00	85	11,11	0,05	692,400	0,868	7,900	107,940	115,840	0,540	9,175	29	area
0,2	4	250,00	68	13,89	0,16	681,220	0,854	7,720	135,740	143,260	0,543	9,230	21	area/speed
0,2	3	333,33	51	18,52	0,15	693,460	0,869	8,010	181,480	189,490	0,544	9,255	16	area/speed
0,2	2	500,00	34	27,78	0,01	726,980	0,911	8,940	274,180	283,120	0,548	9,322	11	area/speed
0,2	1	1000,00	17	55,56	-0,73									

3.3 Confronto dei risultati FPGA vs SC

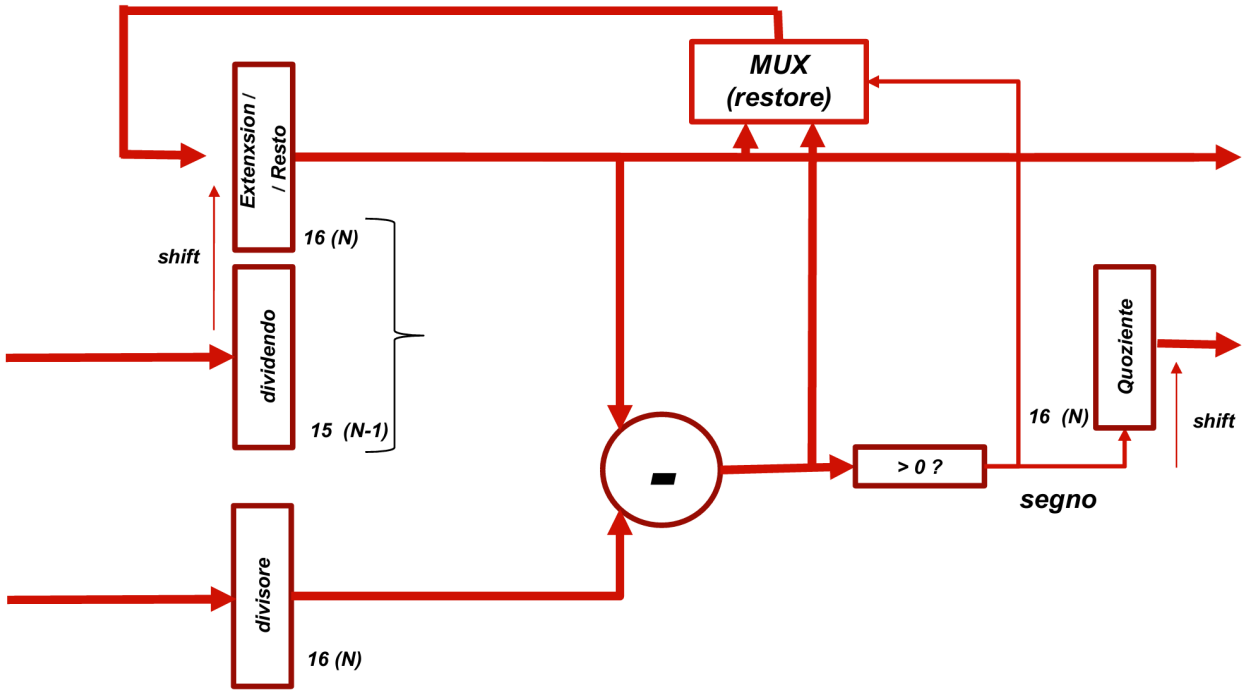
Il confronto tra le soluzioni combinatorie e tra le soluzioni seriali porta alle medesime conclusioni, che elenchiamo a seguire:

- Frequenza di funzionamento molto maggiore sulla soluzione cell based;
- Dissipazione di potenza molto minore sulla soluzione cell based (fattore 10^3).

Come prevedibile i circuiti realizzati con metodologia semi-custom cell based hanno in generale performance molto migliori da ogni punto di vista rispetto alla sintesi su FPGA. Tuttavia si ricorda che la sintesi su standard cell richiede alti costi NRE legati alla generazione delle maschere non richiesti invece nel progetto su FPGA, per il quale è richesto solo determinare il bit stream di programmazione. La scelta di realizzare un IC su standard cell è motivata in caso di reti critiche in termini di velocità o necessità di basso consumo, e diventa una soluzione vantaggiosa quanto ai costi solo in caso di alti volumi di vendita.

4 Soluzione seriale alternativa

Oltre alla soluzione canonica rappresentata dal diagramma in figura 1 è stato scritto un codice RTL rappresentato dal seguente datapath:

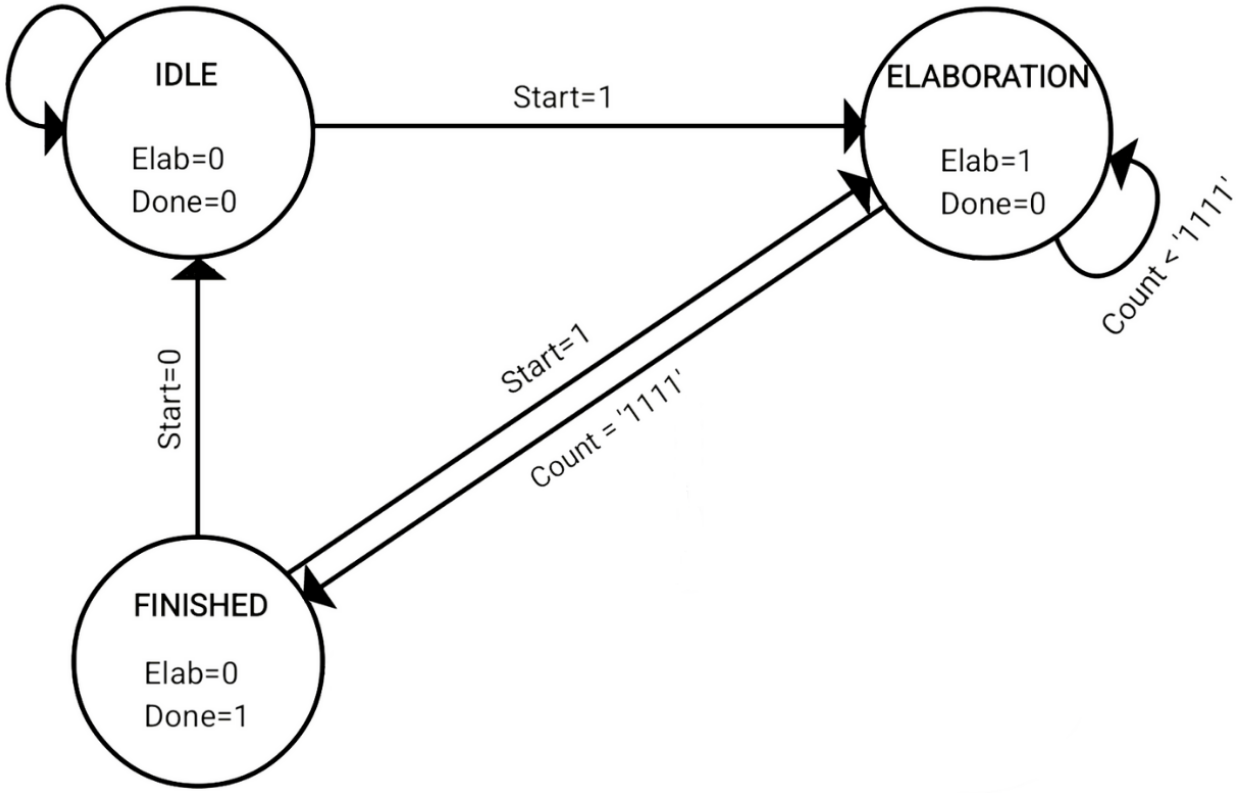


Questa soluzione alternativa ci permette di ottenere vantaggi sotto diversi punti di vista rispetto alla precedente:

- Necessita di un registro da 16 bit in meno, visto che l'uscita è presa direttamente dal registro contenente il resto parziale;
- L'unità di controllo risulta più semplice visto che sono stato necessari solo tre stati (uno in meno della soluzione canonica);
- i cicli di latenza sono stati ridotti da 17 a 16;
- il troughput e aumentato da $\frac{f_{ck}}{18}$ a $\frac{f_{ck}}{17}$;

Per via del risparmio di un registro a 16 bit, si osserva un consumo di potenza statica ridotto e un'area occupata minore sulla soluzione sintetizzata su celle standard. Questa scelta di non inserire un registro ad hoc per l'uscita porta quindi alla possibilità di non attendere la propagazione del segnale dal registro del resto parziale al registro di uscita del resto; questo risparmio di un ciclo di clock è la causa che ha portato ai restanti tre punti sopra elencati.

Si riportano dunque la macchina a stati finiti e il codice della soluzione in analisi:




```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity divider_serial is
7   port (
8   clk : in  std_logic;
9   resetn : in  std_logic;
10  IN_A     : in  std_logic_vector(15 downto 0); -- dividend
11  IN_B     : in  std_logic_vector(15 downto 0); -- divider
12  OUT_DIV  : out std_logic_vector(15 downto 0); -- quotient
13  OUT_REM  : out std_logic_vector(15 downto 0); -- reminder
14  start    : in  std_logic;
15  elab     : out std_logic;
16  done     : out std_logic
17  );
18  end divider_serial;
19
20 architecture behavioral of divider_serial is
21
22 signal in_a_reg: std_logic_vector (14 downto 0);
23 signal in_b_reg: std_logic_vector (15 downto 0);
24 signal rem_d, rem_q: std_logic_vector (15 downto 0);
25 signal div_d, div_q: std_logic_vector(15 downto 0);
26 signal shift, reset_cont: std_logic;
27
28 signal sub: std_logic_vector(15 downto 0);
29
30 type stato is (idle, elaboration, finished);
31 signal cs, ns: stato;
32
33 signal cont: std_logic_vector(3 downto 0);
34
35
36 component COUNTER
37      generic(cw: natural:=16);
38      port (
39         CLK    : in  std_logic;
40         RESET  : in  std_logic;
41         ENABLE : in std_logic;
```

```
42          COUNT : out std_logic_vector(cw-1 downto 0)
43        );
44  end component;
45
46  begin
47
48    reset_cont <= not(resetn and shift);
49
50     cont0 : COUNTER
51    generic map(cw => 4)
52    port map (
53                    CLK   => clk,
54                    RESET => reset_cont,
55        ENABLE => '1',
56                    COUNT => cont
57    );
58
59  -- Registro con enable (start) e reset asincrono di campionamento dividendo
60    process(clk, resetn)
61    begin
62      if resetn = '0' then
63        in_b_reg <= (others => '0');
64      elsif clk'event and clk='1' then
65        if start = '1' then
66          in_b_reg <= IN_B;
67        end if;
68      end if;
69    end process;
70
71  -- Registro di 15 bit con enable (start) e reset asincrono di campionamento divisore (escluso MSB) con rete di shift
72    process(clk, resetn)
73    begin
74      if resetn = '0' then
75        in_a_reg <= (others => '0');
76      elsif clk'event and clk='1' then
77        if start='1' then
78          in_a_reg <= IN_A(14 downto 0);
79        elsif shift='1' then
80          in_a_reg <= in_a_reg(13 downto 0) & '0';
81        end if;
82      end if;
83    end process;
84
85  -- Registro con enable (start) e reset asincrono di campionamento MSB dell'ingresso e shift con registro di campionamento
        del divisore (in_a_reg)
86  -- oss.: registro di campionamento dell'uscita OUT_REM (al termine dell'elaborazione, rem_q corrisponde al risultato di
        resto)
87    process(clk, resetn)
88    begin
89      if resetn = '0' then
90        rem_q <= (others => '0');
91      elsif clk'event and clk='1' then
92        if start='1' then
93          rem_q <= "000000000000000" & IN_A(15);
94        elsif shift='1' then
95          rem_q <= rem_d(14 downto 0) & in_a_reg(14);
96        else
97          rem_q <= rem_d;
98        end if;
99      end if;
100   end process;
101
102 -- Assegnamento uscita OUT_REM
103   OUT_REM <= rem_q;
104
105 -- Registro di campionamento dell'uscita OUT_DIV
106   process(clk, resetn)
107   begin
108     if resetn='0' then
109       div_q <= (others => '0');
110     elsif clk'event and clk='1' then
111       div_q <= div_d;
112     end if;
113   end process;
114
115 -- Assegnamento uscita OUT_DIV
116   OUT_DIV <= div_q;
117
118 -- Rete combinatoria algoritmo di divisione
119   process(cs, start, div_q, rem_q, in_b_reg, sub)
120   begin
121     case cs is
122       when elaboration =>
123         -- Shift verso sx di div_q
124         div_d(15 downto 1) <= div_q(14 downto 0);
125         sub <= unsigned(rem_q) - unsigned(in_b_reg);
126         -- Assegnamento LSB a seconda dei casi + valutazione resto parziale divisione
127         -- oss.: lo shift di rem viene fatto nel registro
128         if unsigned(rem_q)>= unsigned(in_b_reg) then
129           div_d(0) <= '1';
130           rem_d <= sub;
131         else
```

```
132           div_d(0) <= '0';
133           rem_d <= rem_q;
134         end if;
135       when others =>
136         div_d <= (others => '0');
137         rem_d <= (others => '0');
138         sub <= (others => '0');
139       end case;
140   end process;
141
142 -- Assegnamento uscite elab e done
143   process(cs)
144   begin
145     elab <= '0';
146     done <= '0';
147     case cs is
148       when elaboration =>
149         elab <= '1';
150       when finished =>
151         done <= '1';
152       when others =>
153         elab <= '0';
154         done <= '0';
155       end case;
156   end process;
157
158 -- registro di stato di gestione FSM
159   process(clk, resetn)
160   begin
161     if resetn = '0' then
162       cs <= idle;
163     elsif clk'event and clk='1' then
164       cs <= ns;
165     end if;
166   end process;
167
168 -- Rete combinatoria di gestione FSM (si rimanda al dds nella relazione di progetto)
169   process(cs, start, cont)
170   begin
171     case cs is
172       when idle =>
173         shift <= '0';
174         if start = '1' then
175           ns <= elaboration;
176         else
177           ns <= idle;
178         end if;
179       when elaboration =>
180         if cont < "1111" then
181           shift <= '1';
182         else
183           shift <= '0';
184         end if;
185         if cont = "1111" then
186           ns <= finished;
187         else
188           ns <= cs;
189         end if;
190       when finished =>
191         shift <= '0';
192         if start = '1' then
193           ns <= elaboration;
194         else
195           ns <= idle;
196         end if;
197       when others =>
198         shift <= '0';
199         ns <= idle;
200       end case;
201   end process;
202
203 end behavioral;
```

4.1 Verifica funzionale

Si riportano gli screenshot della simulazione ModelSim, e si nota come con la macchina a stati sopra riportata permette il risparmio di un ciclo di latenza rispetto alla soluzione seriale proposta nei capitoli precedenti.

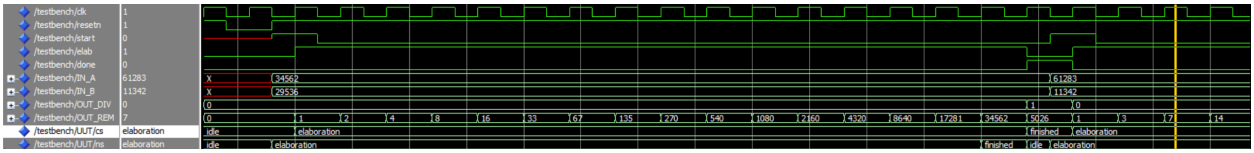


Figura 7: Simulazione Modelsim con ingressi forniti dopo 16 clock dai precedenti (saltato stato di idle)

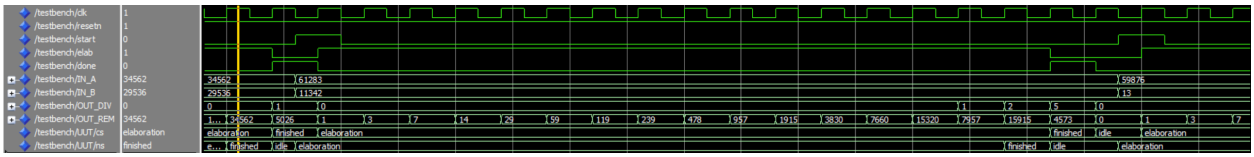


Figura 8: Simulazione Modelsim con ingressi forniti dopo 17 clock dai precedenti (passaggio per idle)

Per quanto riguarda la testbench utilizzata è esattamente analoga a quella riportata nel capitolo sulla precedente soluzione seriale riducendo di uno i periodi di clock tra le varie transizioni degli ingressi.

4.2 Risultati FPGA e SC

Si riportano i risultati ottenuti dalle timing e power analysis ottenute al termine del processo di programmazione dell'FPGA e di sintesi su celle standard.

Period (ns)	Freq. (MHz)	Latency (ns)	Throughput (MS/s)	Max freq. (MHz) (w.c)	Setup check		Hold check		Total LE	Total registers	Total pins
					P, V,T	Slack (ns)	P, V,T	Slack (ns)			
10	100	160	5,882352941	125,85	slow, 1V, 85°C	2,054	fast, 1V, 0°C	0,279	134	69	69
9	111,111	144	6,535947712	124,67	slow, 1V, 85°C	0,979	fast, 1V, 0°C	0,286	133	69	69
8	125	128	7,352941176	126,28	slow, 1V, 85°C	0,081	fast, 1V, 0°C	0,28	133	69	69
7	142,857	112	8,403361345		slow, 1V, 85°C	-0,999					

	T_{CK} (ns)	freq (MHz)	LE/Ltot	toggle rate αT_{CK}		Temperature (°C)		Summary (mW)				By clock domain (mW)	By block type (mW)				Power by clock domain/freq	Energia per svolgere operazione En/op	Power estimation confidence
					computed average toggle rate (Million of Transitions/s)	auto compute junction temperature	Tjun	Total	Core dyn	Core static	I/O		combinational cell	clock control	register cell	I/O	(mW/MHz)	(mW/MHz)	
Tck,min,seriale	8	125	85/6272	vectorless, default toggle rate for I/O signals = 10%	10	si	27,4	89,54	2,44	41,2	45,9	2,44	0,25	1,2	0,68	28,2	0,0195	0,3123	low
Tck,min,seriale	8	125	85/6272	vectorless, default toggle rate for I/O signals = 10%	10	no	85	95,2	2,44	46,86	45,9	2,44	0,25	1,2	0,68	28,2	0,0195	0,3123	low
Tck,min,seriale	8	125	85/6272	from simulation results	19,5	si	27,3	84,22	2,67	41,2	40,35	2,67	0,53	1,2	0,83	22,5	0,0214	0,3418	high

clock uncertainty (ns)	Period (ns)	Freq. MHz	Latency in numero di cicli di clock	Throughput (MS/s)	slack (ns)	Area (um^2)	Area (Kgate,eq)	Leakage (uW)	Dynamic Power (uW)	Total Power (uW)	DynP/freq (uW/MHz)	Energia per svolgere l'operazione (uW/MHz)	n. livelli di logica nel critical path	ottimizzazione macro-celle
							area nand2 = 0,798 um^2							
0,2	5	200,00	80	11,76	0,01	604,620	0,758	6,980	84,660	91,640	0,423	6,773	23	area
0,2	3	333,33	48	19,61	0,23	630,690	0,790	7,640	143,210	150,840	0,430	6,874	17	area/speed
0,2	2	500,00	32	29,41	0	686,010	0,860	8,540	217,980	226,510	0,436	6,975	10	area/speed
0,2	1	1000,00	16	58,82	-0,75		0,000				0,000	0,000		

4.3 Confronto rispetto alla soluzione seriale precedente

Dai risultati ottenuti dalle due soluzioni su FPGA si nota principalmente una riduzione potenza dissipata dai registri e della potenza dinamica per quanto riguarda la soluzione alternativa, mentre non cambia sensibilmente la potenza totale dissipata dal dispositivo. La riduzione del consumo dinamico, assieme al risparmio di un ciclo di latenza, portano anche ad una riduzione dell'energia per svolgere l'operazione del 17% nell'analisi post-fit.

Per quanto riguarda il confronto delle due soluzioni seriali implementate tramite librerie di celle standard si evince che la seconda proposta presenta una riduzione dell'area del 5.6% e una riduzione della potenza totale dissipata del 20% nelle condizioni di massima frequenza di funzionamento, portando a un miglioramento rilevante in entrambi consumo e occupazione di area.