Capitolo 10

Esercizio 10

10.1 Traccia

Progettare ed implementare in VHDL una macchina aritmetica sequenziale a scelta fra le seguenti:

- moltiplicatore di Robertson, per effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna;
- moltiplicatore di Booth, per effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna;
- divisore non-restoring, per effettuare la divisione intera fra due stringhe A e B di 4 bit ciascuna;
- divisore restoring, per effettuare la divisione intera fra due stringhe A e B di 4 bit ciascuna;

In ogni caso, la macchina implementata deve essere sintetizzata su FPGA e deve poter essere testata

mediante l'utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di

sviluppo in dotazione al gruppo. La modalità di utilizzo degli stessi è a completa discrezione degli studenti.

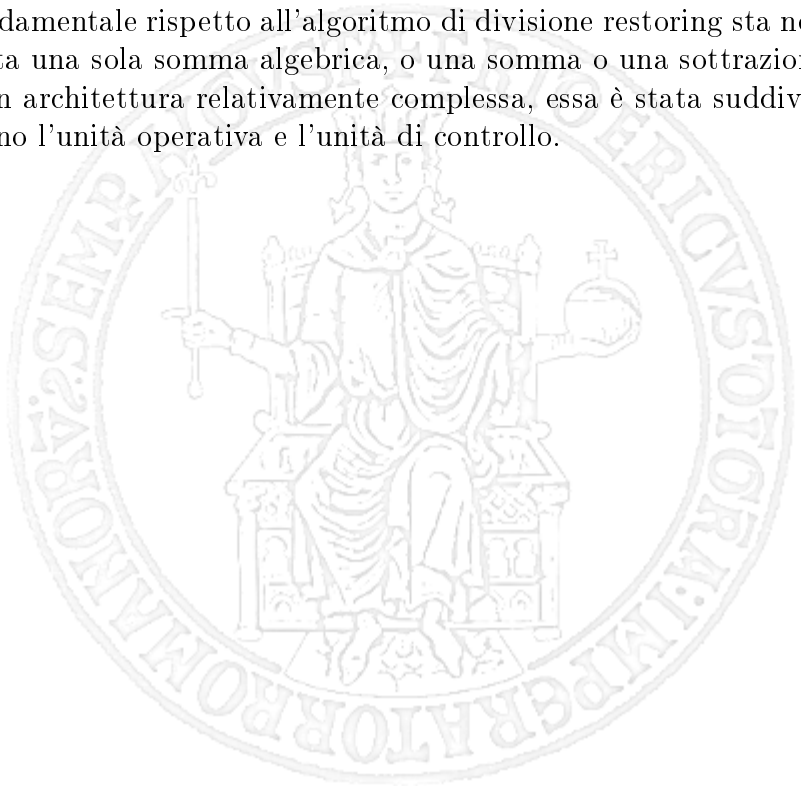
10.2 Soluzione

Si è deciso di implementare il divisore non restoring secondo il seguente algoritmo:

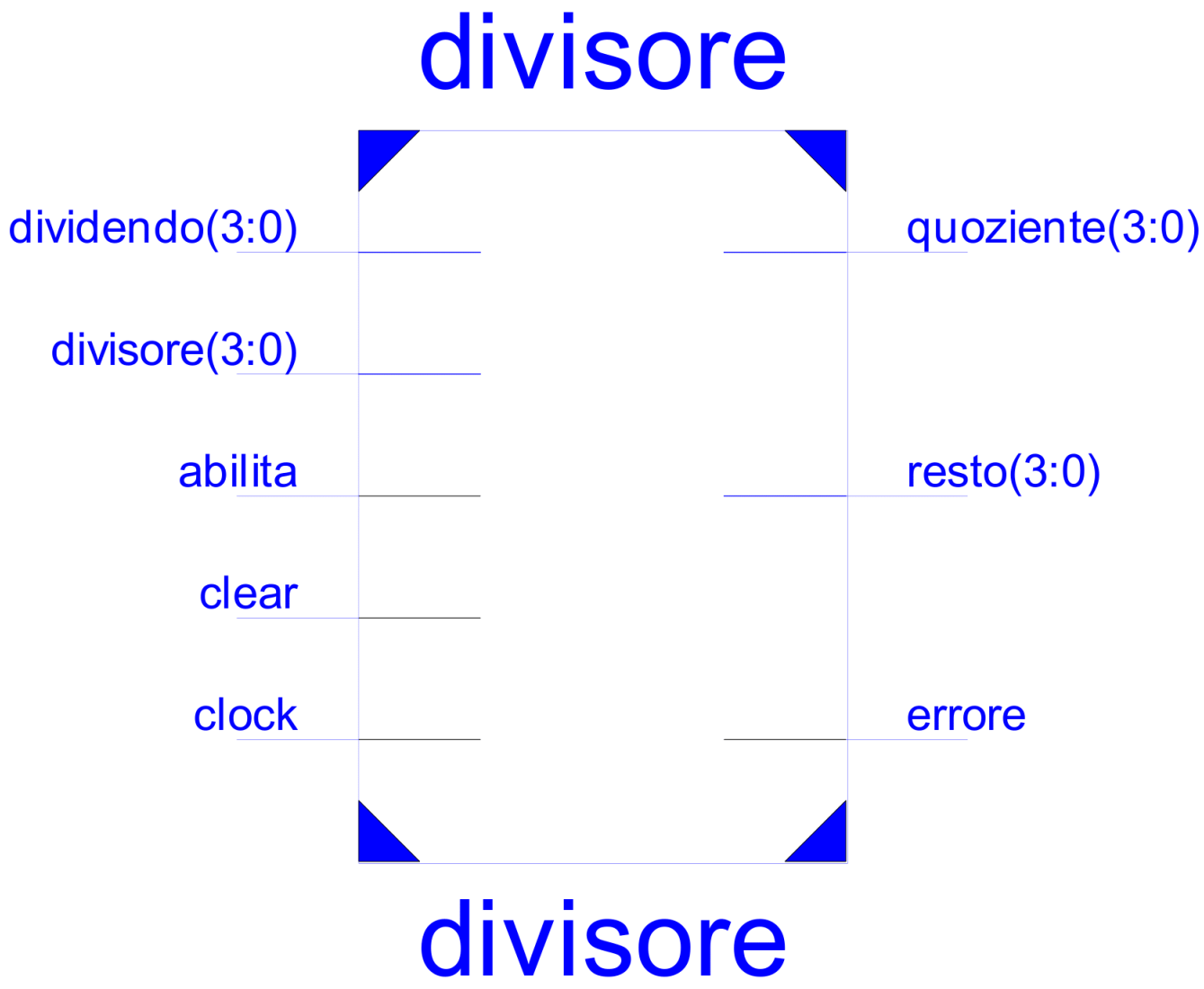
```
NRDivider:      (in:INBUS; OUT:OUTBUS)
                register S,A[n-1:0],M[n-1:0],Q[n-1:0],COUNT[log2n:0];
                bus INBUS[n-1:0], OUTBUS[n-1:0];
BEGIN:          COUNT:=0;S:=0;
INPUT:          A:=INBUS {carico la prima metà del dividendo D (0 in testa)}
                Q=INBUS {carico la seconda metà del dividendo D}
                M:=INBUS; {divisore V}

LSHIFT:         S.A.Q[n-1:1]=A.Q; {la prima volta S è 0, e dopo lo shift è ancora 0}

SUB:            if S==0 then
                        S.A:=S.A-M;
                else
SUM:                    S.A:=S.A+M;
                endif

SETq:           Q[0]:=not S;
                COUNT:=COUNT+1;

COUNT_TEST:     if COUNT< n then goto LSHIFT;
                endif

CORRECTION :    if S==1 then
                        S.A:=S.A+M;
                endif

OUTPUT:  OUTBUS;=Q, OUTBUS:=A;
END NRDivider;
```

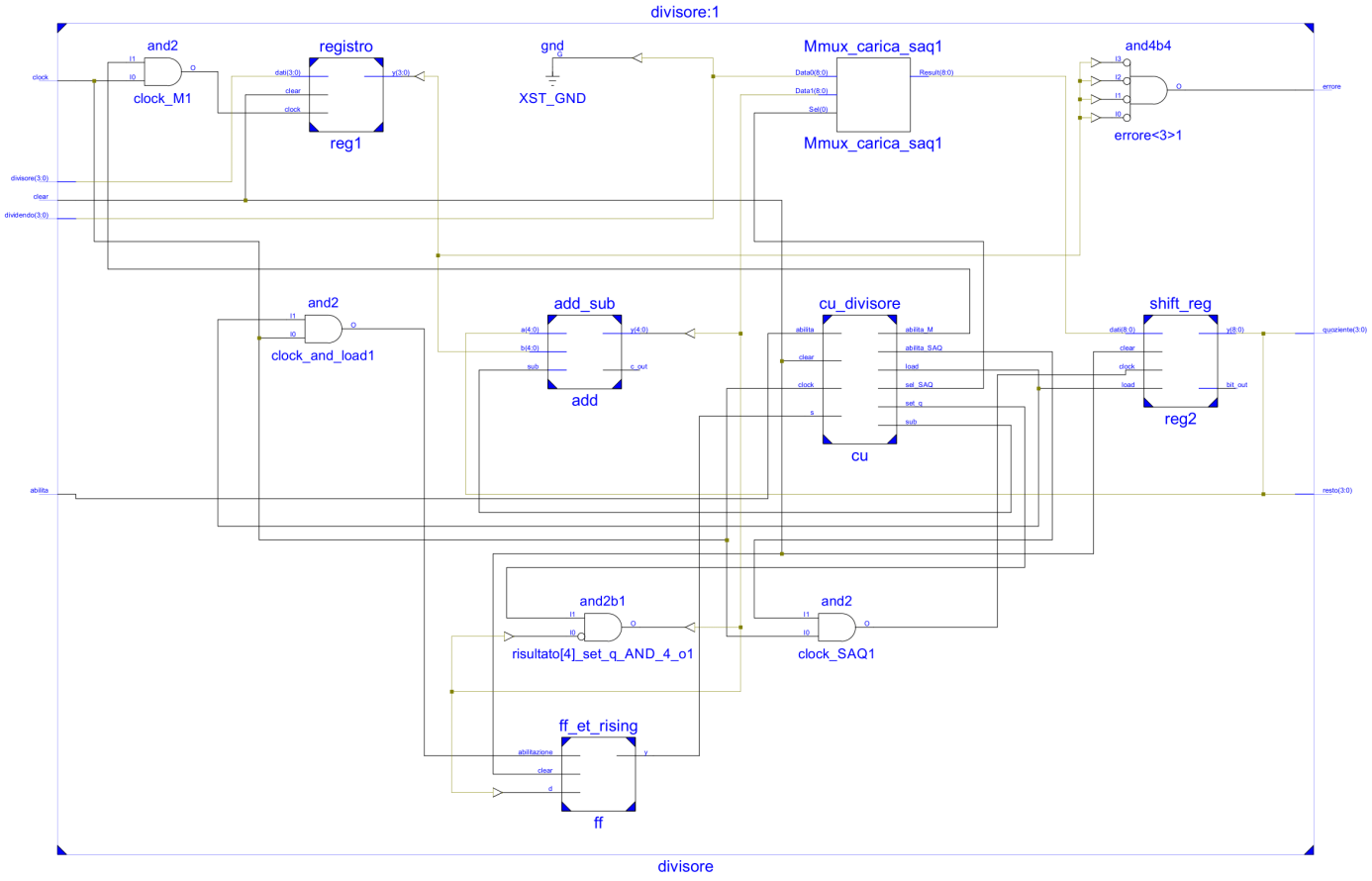
La differenza fondamentale rispetto all'algoritmo di divisione restoring sta nel fatto che ad ogni passo viene effettuata una sola somma algebrica, o una somma o una sottrazione.

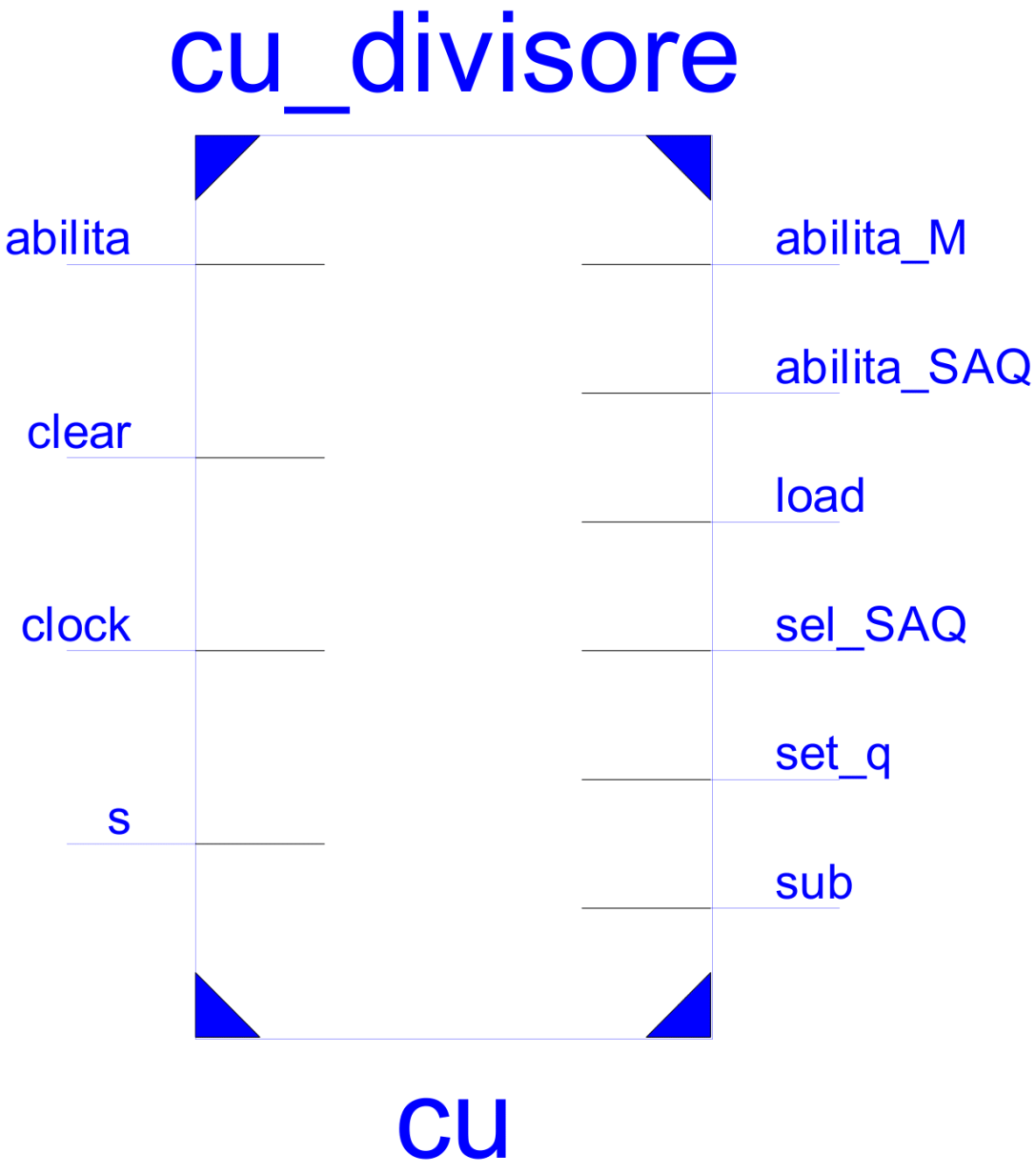
Trattandosi di un architettura relativamente complessa, essa è stata suddivisa nei due blocchi fondamentali che sono l'unità operativa e l'unità di controllo.

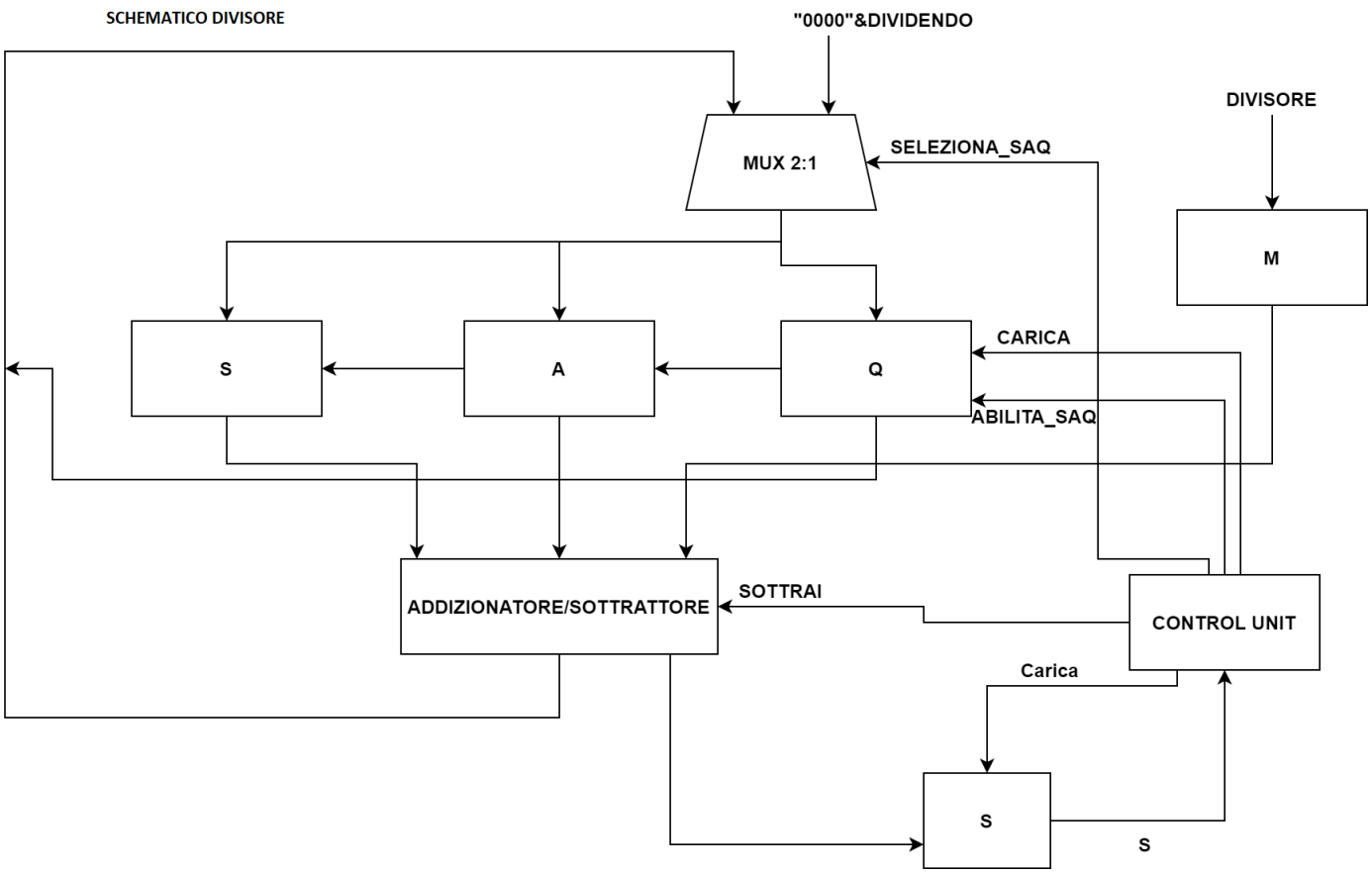


10.2.1 Schematici




divisore:1
and2
clock_M1
registro
reg1
gnd
XST_GND
Mmux_carica_saq1
Mmux_carica_saq1
and4b4
errore<3>1
clock
divisore(3:0)
clear
dividendo(3:0)
errore
and2
clock_and_load1
add_sub
add
cu_divisore
cu
shift_reg
reg2
quoziente(3:0)
resto(3:0)
abilita
and2b1
risultato[4]_set_q_AND_4_o1
and2
clock_SAQ1
ff_et_rising
ff
divisore


cu_divisore
abilita
clear
clock
s
abilita_M
abilita_SAQ
load
sel_SAQ
set_q
sub
cu



10.2.2 Codice

Unità operativa

L'unità operativa è formata da un registro a scorrimento denominato SAQ con ingresso parallelo, uscita parallela,caricamento e shift dei dati contenuti (segno,dividendo,resto) sul fronte di salita del segnale di clock, da un registro M che contiene il divisore ed è a caricamento sincronizzato sul fronte di salita del segnale di clock,da un addizionatore/sottrattore, da un convogliatore a 2 sorgenti ed 1 uscita per stabilire se mandare in ingresso al registro a scorrimento il risultato della divisione o il dividendo, da un flip flop edge triggered sul fronte di salita che viene utilizzato per mantenere il segno ad ogni passo cosa che si rende necessaria dal momento che il bit di segno, contenuto nel registro SAQ, potrebbe modificarsi a causa delle operazioni di shift associate al registro stesso. In uscita è presente un segnale denominato errore che assume il livello logico alto nel momento in cui il divisore è pari a zero. Si osservi che il registro a scorrimento denominato SAQ memorizza un numero di bit pari a 9 perchè quoziente e resto sono rappresentati al massimo su 4 bit e poi serve un ulteriore bit nel caso di risultati negativi uscenti dall'addizionatore/sottrattore. Il registro M memorizza 4 bit perchè, come imposto dalla traccia, il divisore è codificato su 4 bit. Il sommatore/sottrattore prende in ingresso operandi codificati su 5 bit, dunque anche il risultato è rappresentato su 5 bit e questo è il motivo per cui viene utilizzato il segnale M_1 che fa si che il bit più significativo del divisore sia uno zero.Il segnale A racchiude S e A del SAQ ed è fondamentale per connettere il registro a scorrimento al sommatore/sottrattore. Clock_SAQ,Clock_M e clock_and_load sono rispettivamente i segnali di tempificazione per il registro SAQ,per il registro M e per il flip flop che memorizza il bit di segno. Se la linea di selezione del multiplexer è alta in

ingresso al SAQ va una stringa di bit formata dalla concatenazione della somma algebrica prodotta dall'addizionatore, dei i bit del quoziente da 3 a 1 e del segno negato della somma algebrica in and con il segnale denominato set_q che fa si che il quoziente sia modificato secondo l'algoritmo proprio della divisione non restoring.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity divisore is
5  port( dividendo: in std_logic_vector(3 downto 0);
6      divisore: in std_logic_vector(3 downto 0);
7      clock: in std_logic;
8      abilita: in std_logic;
9      clear: in std_logic;
10     quoziente: out std_logic_vector(3 downto 0);
11     resto: out std_logic_vector(3 downto 0);
12     errore: out std_logic);
13 end divisore;
14
15
16 architecture structural of divisore is
17
18 component cu_divisore
19 port( clock: in std_logic;
20     clear: in std_logic;
21     abilita: in std_logic;
22     s: in std_logic;
23     abilita_SAQ: out std_logic;
24     abilita_M: out std_logic;
25     load: out std_logic;
26     sub: out std_logic;
27     set_q: out std_logic;
28     sel_SAQ: out std_logic);
29 end component;
30
31
32 component registro
33 port  ( clock: in std_logic;
34       clear: in std_logic;
35       dati: in std_logic_vector(3 downto 0);
36       y: out std_logic_vector(3 downto 0));
37 end component;
38
39 component shift_reg
40 port  ( clock: in std_logic;
41       clear: in std_logic;
42       load: in std_logic;
43       dati: in std_logic_vector(8 downto 0);
44       y: out std_logic_vector(8 downto 0);
```

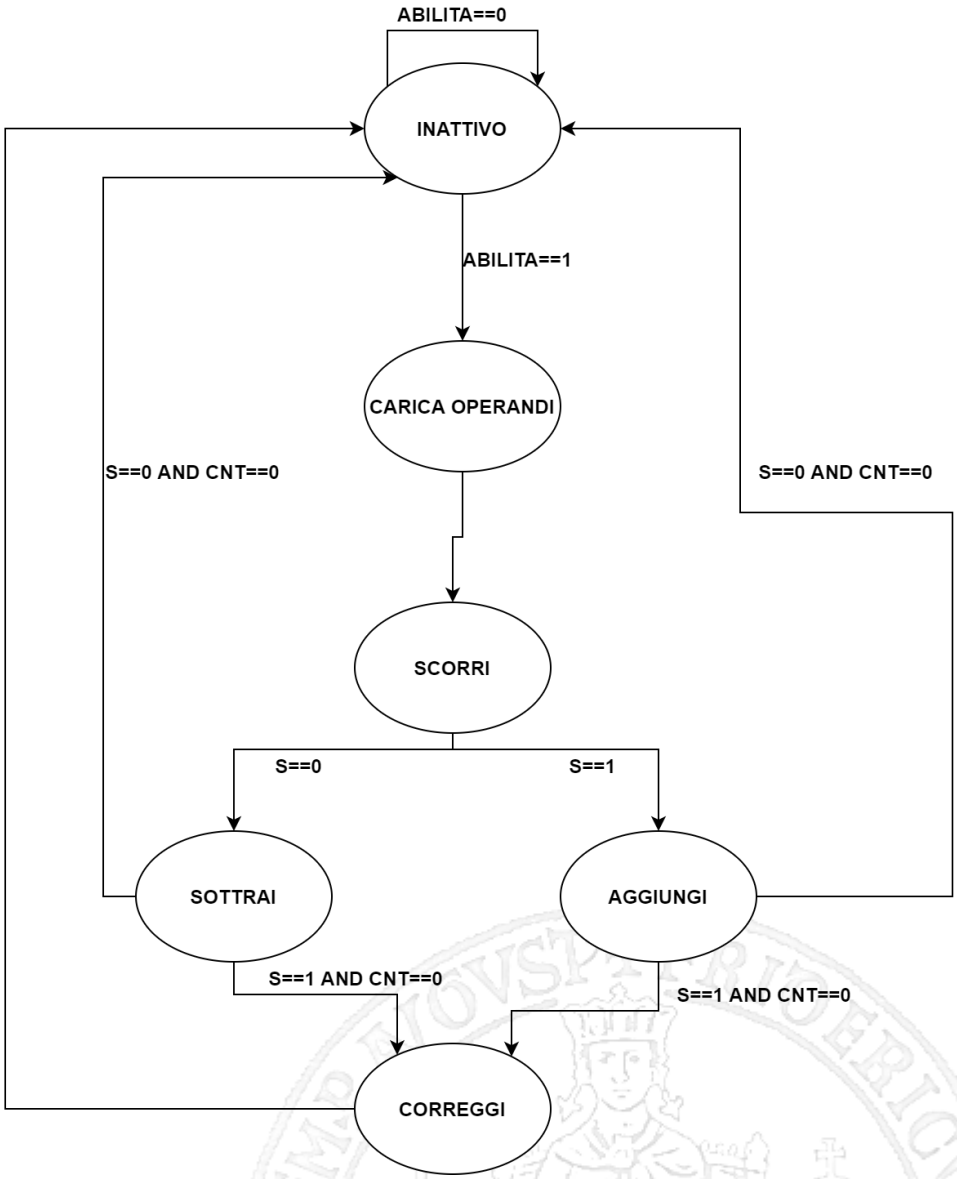
```
45      bit_out: out std_logic);
46  end component;
47
48
49  component add_sub
50  port ( a: in std_logic_vector(4 downto 0);
51       b: in std_logic_vector(4 downto 0);
52       sub: in std_logic;
53       y: out std_logic_vector(4 downto 0);
54       c_out: out std_logic);
55  end component;
56
57  component ff_et_rising
58  port ( d: in std_logic;
59       abilitazione: in std_logic;
60       clear: in std_logic;
61       y: out std_logic );
62  end component;
63
64  signal risultato: std_logic_vector(4 downto 0);
65  signal M_1: std_logic_vector(4 downto 0);
66  signal a: std_logic_vector(4 downto 0);
67  signal Q: std_logic_vector(3 downto 0);
68  signal carica_saq,saq: std_logic_vector(8 downto 0);
69  signal M: std_logic_vector(3 downto 0);
70  signal abilita_SAQ,abilita_M: std_logic;
71  signal load: std_logic;
72  signal sub: std_logic;
73  signal s: std_logic;
74  signal sel_SAQ: std_logic;
75  signal set_q: std_logic;
76  signal clock_M,clock_and_load,clock_SAQ: std_logic;
77
78  begin
79  errore<='1' when M="0000" else '0';
80  clock_M <= clock and abilita_M;
81  clock_SAQ <= clock and abilita_SAQ;
82  clock_and_load <= clock and load;
83  M_1<= '0'&M;
84
85  carica_saq <= risultato (4 downto 0) & Q (3 downto 1) & ( not ( risultato
        (4)) and set_q ) when sel_SAQ='1' else
86        "00000"&dividendo when sel_SAQ='0' else
87        "XXXXXXXXX";
88  a<=saq(8 downto 4);
89  Q<=saq(3 downto 0);
90  quoziente<=Q;
91  resto<=a(3 downto 0);
92
```

```
93 cu: cu_divisore port map(clock,clear,abilita,s,abilita_SAQ,abilita_M,load,
       sub,set_q,sel_SAQ);
94 add: add_sub port map(a,M_1,sub,risultato,open);
95 reg1: registro port map(clock_M,clear,divisore,M);
96 ff: ff_et_rising port map(risultato(4),clock_and_load,clear,s);
97 reg2: shift_reg port map(clock_SAQ,clear,load,carica_saq,SAQ,open);
98 end structural;
```

Codice Componente 10.1: Divisore

Unità di controllo

L'unità di controllo, come noto, controlla e comanda ciò che avviene nel sistema, fornisce all'unità operativa gli opportuni segnali di controllo affinche essa realizzi il comportamento desiderato per il sistema. Come si può osservare dallo schematico presente nella sezione superiore,in ingresso l'unità di controllo prende il segnale di clock,il segnale di clear, il segnale di abilitazione e il bit relativo al segno del dividendo parziale, fornito dal flip flop presente nell'unità operativa.In uscita l'unità di controllo produce il segnale abilita_M relativo al caricamento dei 4 bit del divisore nel registro M, il segnale abilita_SAQ che , se asserito, rende operativo il registro a scorrimento in termini sia di load sia di shift, il segnale sub che impone all'addizionatore/sottrattore di effettuare una somma oppure una sottrazione,set_q per fare distinzione tra la correzione del resto che non richiede la modifica del quoziente e l'addizione con modifica del bit del quoziente, e sel_SAQ che rappresenta la linea di selezione del multiplexer precedentemente descritto. L'unità di controllo realizza il seguente automa a stati finiti:



L'unità di controllo fa uso di un contatore modulo 4 ed è descritta al livello di astrazione comportamentale. Mentre l'unità di controllo produce i segnali di abilitazione in corrispondenza del fronte di discesa del clock, l'unità operativa funziona in corrispondenza del fronte di salita del clock.Sono presenti due process, uno che si occupa di aggiornare lo stato attuale della macchina e l'altro che invece imposta i livelli logici dei vari segnali di controllo in base allo stato corrente. Nello stato "inattivo" tutti i segnali di controllo sono al livello logico basso e si passa allo stato "carica operandi" soltanto quando il segnale abilita diventa alto.Nello stato "carica operandi" vengono abilitati i due registri M e SAQ, la linea di selezione del multiplexer sta al livello logico basso, e viene inviato al SAQ il segnale di load. L'unica differenza tra lo stato "aggiungi" e lo stato "sottrai" risiede nel bit inviato al sommatore/sottrattore che specifica se fare, appunto, somma o sottrazione.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
```

```
5  entity cu_divisore is
6  port( clock: in std_logic;
7      clear: in std_logic;
8      abilita: in std_logic;
9      s: in std_logic;
10     abilita_SAQ: out std_logic;
11     abilita_M: out std_logic;
12     load: out std_logic;
13     sub: out std_logic;
14     set_q: out std_logic;
15     sel_SAQ: out std_logic);
16 end cu_divisore;
17
18 architecture behavioural of cu_divisore is
19
20 TYPE stato is (inattivo,carica_operandi,scorri,sottrai,aggiungi,correggi);
21 signal stato_corrente: stato := inattivo;
22 signal stato_prossimo: stato := inattivo;
23 signal val_count: std_logic_vector(1 downto 0);
24 signal abilita_cont,cnt_0,cnt_or_sign,clear_n: std_logic;
25
26 component counter_mod4
27 port( clock: in std_logic;
28       clear_n: in std_logic;
29     abilita: in std_logic;
30     conteggio: out std_logic_vector(1 downto 0));
31 end component;
32
33
34 begin
35   clear_n <= not clear;
36   cnt_or_sign <= '1' when (( to_integer ( unsigned ( val_count )) > 0) or s
       = '1') else '0';
37   cnt_0 <= '1' when ( to_integer ( unsigned ( val_count )) = 0) else '0' ;
38   c4: counter_mod4 port map(clock,clear_n,abilita_cont,val_count);
39
40   proc: process(clock)
41   begin
42     if clock'event and clock='0' then
43       stato_corrente<=stato_prossimo;
44     end if;
45   end process;
46
47   proc2: process(stato_corrente,abilita,s)
48   begin
49     if stato_corrente=inattivo then
50       if abilita='1' then
51         stato_prossimo<=carica_operandi;
52       else
```

```
53          stato_prossimo<=inattivo;
54        end if;
55        abilita_SAQ<='0';
56        sel_SAQ<='0';
57        abilita_M<='0';
58        abilita_cont<='0';
59        load<='0';
60        set_q<='0';
61        sub<='0';
62      elsif stato_corrente=carica_operandi then
63        stato_prossimo<=scorri;
64        abilita_SAQ<='1';
65        abilita_M<='1';
66        sel_SAQ<='0';
67        abilita_cont<='0';
68        set_q<='0';
69        load<='1';
70        sub<='0';
71      elsif stato_corrente=scorri then
72        if s='1' then
73          stato_prossimo<=aggiungi;
74        else
75          stato_prossimo<=sottrai;
76        end if;
77        abilita_SAQ<='1';
78        sel_SAQ<='0';
79        abilita_cont<='0';
80        abilita_M<='0';
81        set_q<='0';
82        load<='0';
83        sub<='0';
84      elsif stato_corrente=sottrai then
85        if cnt_0='0' then
86          stato_prossimo<=scorri;
87        elsif cnt_0='1' and s='1' then
88          stato_prossimo<=correggi;
89        else
90          stato_prossimo<=inattivo;
91        end if;
92        abilita_SAQ<='1';
93        abilita_M<='0';
94        abilita_cont<='1';
95        sel_SAQ<='1';
96        set_q<='1';
97        load<='1';
98        sub<='1';
99      elsif stato_corrente=aggiungi then
100       if cnt_0='0' then
101         stato_prossimo<=scorri;
```

```
102       elsif cnt_0='1' and s='1' then
103         stato_prossimo<=correggi;
104       else
105         stato_prossimo<=inattivo;
106       end if;
107       abilita_SAQ<='1';
108       abilita_M<='0';
109       abilita_cont<='1';
110       sel_SAQ<='1';
111       set_q<='1';
112       load<='1';
113       sub<='0';
114     elsif stato_corrente=correggi then
115       stato_prossimo<=inattivo;
116       abilita_SAQ<='1';
117       abilita_M<='0';
118       abilita_cont<='0';
119       sel_SAQ<='1';
120       set_q<='0';
121       load<='1';
122       sub<='0';
123 
124     end if;
125 
126   end process;
127 
128 end behavioural;
```

Codice Componente 10.2: Control Unit divisore

10.3 Simulazione

Codice testbench divisore:

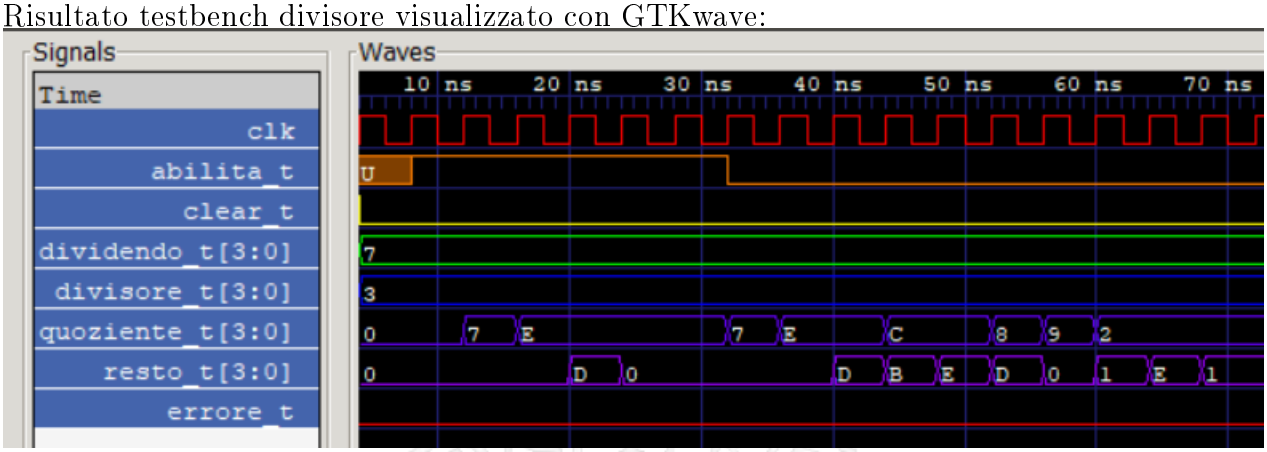
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  
4  entity divisore_tb is
5  
6  end divisore_tb;
7  
8  
9  
10 architecture behavioural of divisore_tb is
11 component divisore
12 port( dividendo: in std_logic_vector(3 downto 0);
13     divisore: in std_logic_vector(3 downto 0);
14     clock: in std_logic;
```

```
15      abilita: in std_logic;
16      clear: in std_logic;
17      quoziente: out std_logic_vector(3 downto 0);
18      resto: out std_logic_vector(3 downto 0);
19      errore: out std_logic);
20  end component;
21
22  constant clk_period: time := 4 ns;
23  signal dividendo_t,divisore_t: std_logic_vector(3 downto 0);
24  signal clk,abilita_t,clear_t,errore_t: std_logic;
25  signal quoziente_t,resto_t: std_logic_vector(3 downto 0);
26
27  begin
28
29  uut: divisore port map(dividendo_t,divisore_t,clk,abilita_t,clear_t,
        quoziente_t,resto_t,errore_t);
30
31  clk_process: process
32  begin
33    clk<='1';
34    wait for clk_period/2;
35    clk<='0';
36    wait for clk_period/2;
37  end process;
38
39  stim_proc: process
40  begin
41
42    wait for 2 ns;
43    clear_t<='1';
44    wait for 2 ns;
45    clear_t<='0';
46    dividendo_t<="0111"; --15
47    divisore_t<="0011"; --14
48    wait for 4 ns;
49    abilita_t<='1';
50    wait for 24 ns;
51    abilita_t<='0';
52
53
54
55    wait for 150 ns;
56
57    assert quoziente_t="0010"
58    report "errore nel quoziente"
59    severity failure;
60
61    assert resto_t="0001"
62    report "errore nel resto"
```

```
63    severity failure;
64
65    assert false
66    report "fine"
67    severity failure;
68
69
70  end process;
71
72
73
74  end behavioural;
```

Codice Componente 10.3: Control Unit divisore

Risultato testbench divisore visualizzato con GTKwave:



10.4 Sintesi su board FPGA

Per la sintesi su board, dividendo e divisore sono stati mappati rispettivamente sui primi e sugli ultimi 4 switch a partire da destra, clear e abilita su due bottoni, quoziente e resto rispettivamente sui primi e sugli ultimi 4 LED a partire da destra.