



**UNIVERSITÀ<sup>DEGLI</sup> STUDI DI  
NAPOLI FEDERICO II**

Scuola Politecnica e delle Scienze di Base  
Corso di Laurea in Ingegneria Informatica

Elaborato in Architettura dei Sistemi Digitali

***Elaborato finale***

Anno Accademico 2024/25

Studenti

**Filomena Vigliotti**  
matr. M63001734  
**Ciro Scognamilgio**  
matr. M6300  
**Antonio Sirignano**  
matr. M63001732

# Indice

<b>1 Esercizio 1</b>	<b>1</b>
1.1 Multiplexer 16:1 . . . . .	1
1.1.1 Progetto e architettura . . . . .	3
1.1.2 Implementazione . . . . .	8
1.1.3 Simulazione . . . . .	12
1.1.4 Implementazione 2.0 . . . . .	15
1.2 Rete di interconnessione a 16 ingressi e 4 uscite . . . . .	19
1.2.1 Progettazione . . . . .	20
1.2.2 Implementazione . . . . .	23
1.2.3 Simulazione . . . . .	29
1.3 Implementazione su board del punto precedente . . . . .	34
1.3.1 Traccia . . . . .	35
1.3.2 Implementazione . . . . .	36
1.3.3 Funzionamento . . . . .	40
<b>2 Esercizio 2 - Sistema ROM+M</b>	<b>42</b>
2.1 Progettazione . . . . .	43
2.2 Implementazione . . . . .	43
2.3 Simulazione . . . . .	47

2.4	Implementazione su board . . . . .	50
2.4.1	Traccia . . . . .	50
2.4.2	Implementazione . . . . .	50
<b>3</b>	<b>Esercizio 3</b>	<b>53</b>
3.1	Riconoscitore di sequenze . . . . .	53
3.1.1	Progettazione e architettura . . . . .	54
3.1.2	Implementazione . . . . .	55
3.1.3	Simulazione . . . . .	58
3.2	Implementazione su board del punto precedente . . . . .	62
3.2.1	Traccia . . . . .	62
3.2.2	Implementazione . . . . .	62
<b>4</b>	<b>Esercizio 4</b>	<b>68</b>
4.1	Shift Register - Approccio comportamentale . . . . .	68
4.1.1	Progetto e architettura . . . . .	68
4.1.2	Implementazione . . . . .	70
4.1.3	Simulazione . . . . .	71
4.2	Shift Register - Approccio strutturale . . . . .	74
4.2.1	Progetto e architettura . . . . .	74
4.2.2	Implementazione . . . . .	77
4.2.3	Simulazione . . . . .	81
<b>5</b>	<b>Esercizio 5</b>	<b>86</b>
5.1	Cronometro . . . . .	86
5.1.1	Progettazione . . . . .	86
5.1.2	Implementazione . . . . .	93

5.1.3	Simulazione . . . . .	105
5.2	Implementazione su board del punto precedente . . . . .	108
5.2.1	Traccia . . . . .	108
5.2.2	Implementazione . . . . .	108
<b>6</b>	<b>Esercizio 6</b>	<b>127</b>
6.1	Sistema di lettura - elaborazione - scrittura PO_PC . .	127
6.1.1	Traccia . . . . .	127
6.1.2	Progettazione . . . . .	128
6.1.3	Implementazione . . . . .	130
6.1.4	Simulazione . . . . .	137
6.2	Implementazione su board del punto precedente . . . . .	140
6.2.1	Traccia . . . . .	140
6.2.2	Implementazione . . . . .	140
<b>7</b>	<b>Esercizio 7</b>	<b>149</b>
7.1	Moltiplicatore di Booth . . . . .	149
7.1.1	Progettazione . . . . .	149
7.1.2	Implementazione . . . . .	153
7.1.3	Simulazione . . . . .	164
7.2	Implementazione su board del punto precedente . . . . .	166
7.2.1	Traccia . . . . .	166
7.2.2	Implementazione . . . . .	166
<b>8</b>	<b>Esercizio 8.1</b>	<b>171</b>
8.1	Comunicazione con handshaking . . . . .	171
8.1.1	Traccia . . . . .	171

8.1.2	Progettazione . . . . .	172
8.1.3	Implementazione . . . . .	174
8.1.4	Simulazione . . . . .	192
<b>9 Esercizio 9</b>		<b>196</b>
9.1	Il processore Mic-1 . . . . .	196
9.1.1	Unità operativa . . . . .	197
9.1.2	Microistruzioni . . . . .	199
9.1.3	Unità di Controllo . . . . .	200
9.2	La microistruzione ISUB . . . . .	201
9.2.1	Simulazione . . . . .	203
9.3	La microistruzione IOR . . . . .	205
9.3.1	Simulazione . . . . .	207
9.4	Implmentazione della microistruzione IXOR . . . . .	208
<b>10 Esercizio 8</b>		<b>213</b>
10.1	Prova di esame del 19 dicembre 2024 . . . . .	213
10.1.1	Traccia . . . . .	213
10.1.2	Progettazione . . . . .	214
10.1.3	Implementazione . . . . .	216
10.1.4	Simulazione . . . . .	233
<b>Bibliografia</b>		<b>237</b>

# Capitolo 1

## Esercizio 1

### 1.1 Multiplexer 16:1

Un multiplexer è una **macchina combinatoria**, ovvero una macchina la cui uscita in un determinato istante di tempo dipende solo dall'ingresso nel medesimo istante, e quindi realizza una funzione del tipo:

$$U = f(I)$$

dove  $I$  e  $U$  rappresentano rispettivamente gli insiemi limitati dei valori di ingresso e di uscita.

Il Multiplexer realizza una connessione  $n:1$ , ovvero connette  $n$  sorgenti a un'unica destinazione sulla base di segnali di selezione.

Un **Multiplexer lineare** è composto da  $n$  segnali in ingresso e  $n$  segnali di selezione. Tale dispositivo convoglia uno specifico segnale in ingresso verso l'uscita solo se il corrispondente segnale di selezione è

alto. Uno svantaggio di un dispositivo di questo tipo è il numero eccessivo di fili per i segnali di selezione. Per risolvere ciò si può aggiungere un **Decoder**, un altro dispositivo notevole, che riceve in ingresso una parola codice di  $n$  bit e presenta in uscita la sua rappresentazione decodificata di  $2^n$  bit.

Unendo un Multiplexer lineare a un Decoder, l'architettura diventa quella in figura, e si ottiene un componente definito **Multiplexer indirizzabile**, che diversamente da quello lineare, prende solo 2 segnali di selezione in ingresso. Un MUX indirizzabile è a sua volta una macchina notevole, caratterizzata da  $2^n$  ingressi,  $n$  ingressi di selezione e un'unica uscita.

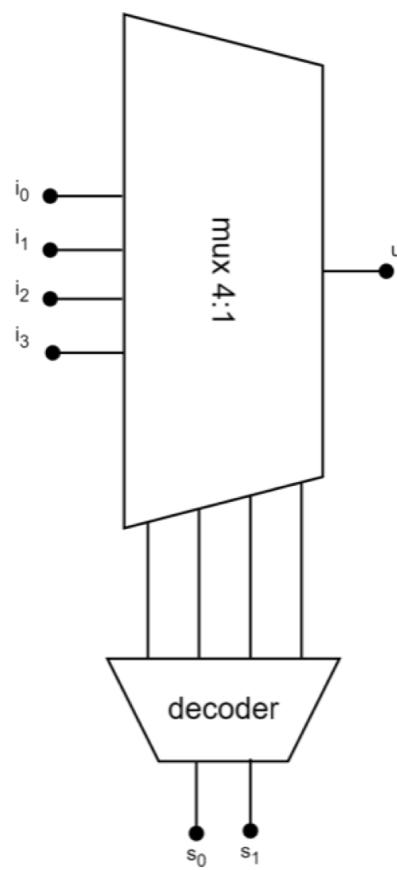


Figura 1.1: multiplexer indirizzabile

Si vuole ora progettare un multiplexer indirizzabile 16:1, utilizzando un approccio per composizione, a partire da multiplexer 4:1.  
Tale multiplexer è rappresentato di seguito.

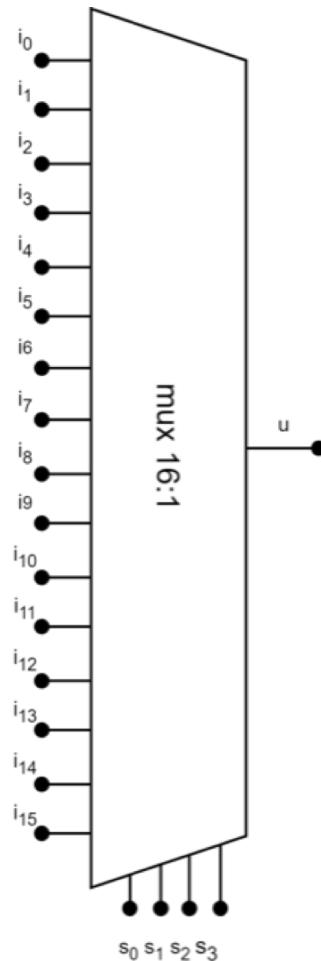


Figura 1.2: multiplexer 16:1

### 1.1.1 Progetto e architettura

Dapprima si utilizza un approccio per composizione per realizzare un multiplexer 4:1 con multiplexer 2:1.

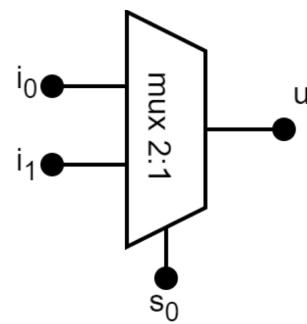


Figura 1.3: multiplexer 2:1

Il primo componente che si realizza è un multiplexer 2:1, caratterizzato dalla seguente tabella di verità:

<b>s<sub>0</sub></b>	<b>i<sub>1</sub></b>	<b>i<sub>0</sub></b>	<b>u</b>
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Tabella 1.1: Tabella di verità di un Mux 2:1

da cui si ottiene l'equazione:

$$u = (i_0 \text{ AND } \bar{s}_0) \text{ OR } (i_1 \text{ AND } s_0)$$

Il successivo componente da costruire è un multiplexer 4:1.

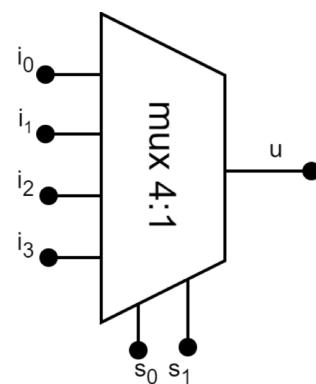


Figura 1.4: multiplexer 4:1

Per composizione, a partire da 3 multiplexer 2:1, si può ottenere un multiplexer 4:1

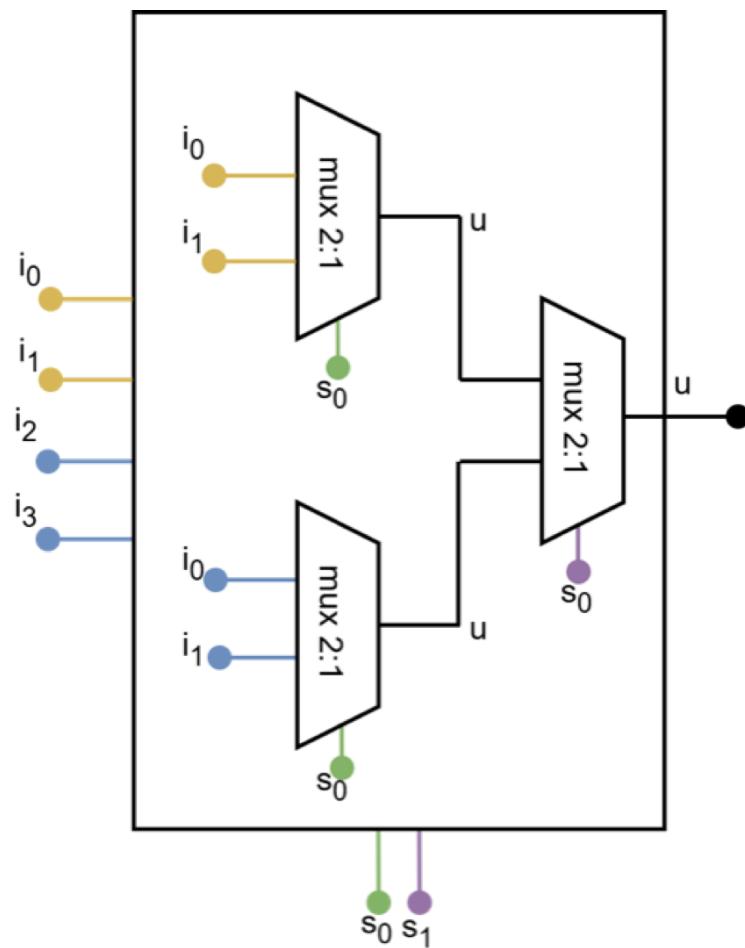


Figura 1.5: multiplexer 4:1 per composizione di multiplexer 2:1

I 4 ingressi entrano in due multiplexer 2:1, che prendono due ingressi e producono un'uscita ciascuno; tali uscite vengono immesse nel terzo multiplexer, che produrrà l'unico output finale. Concettualmente, si divide la selezione in ingresso al multiplexer esterno in due parti:

- la parte meno significativa (indicata dal colore verde)  $s_0$ , viene posta in ingresso ai multiplexer del primo stadio e seleziona per ciascuno un filo in uscita;
- la parte più significativa (indicata dal colore viola)  $s_1$  entra nel multiplexer del secondo stadio e decide quale dei due fili, provenienti dai due blocchi precedenti, sarà immessa in uscita.

In maniera analoga si procede con la progettazione del multiplexer 16:1.

Anche in questo caso, sono stati usati dei colori per identificare i collegamenti tra le componenti.

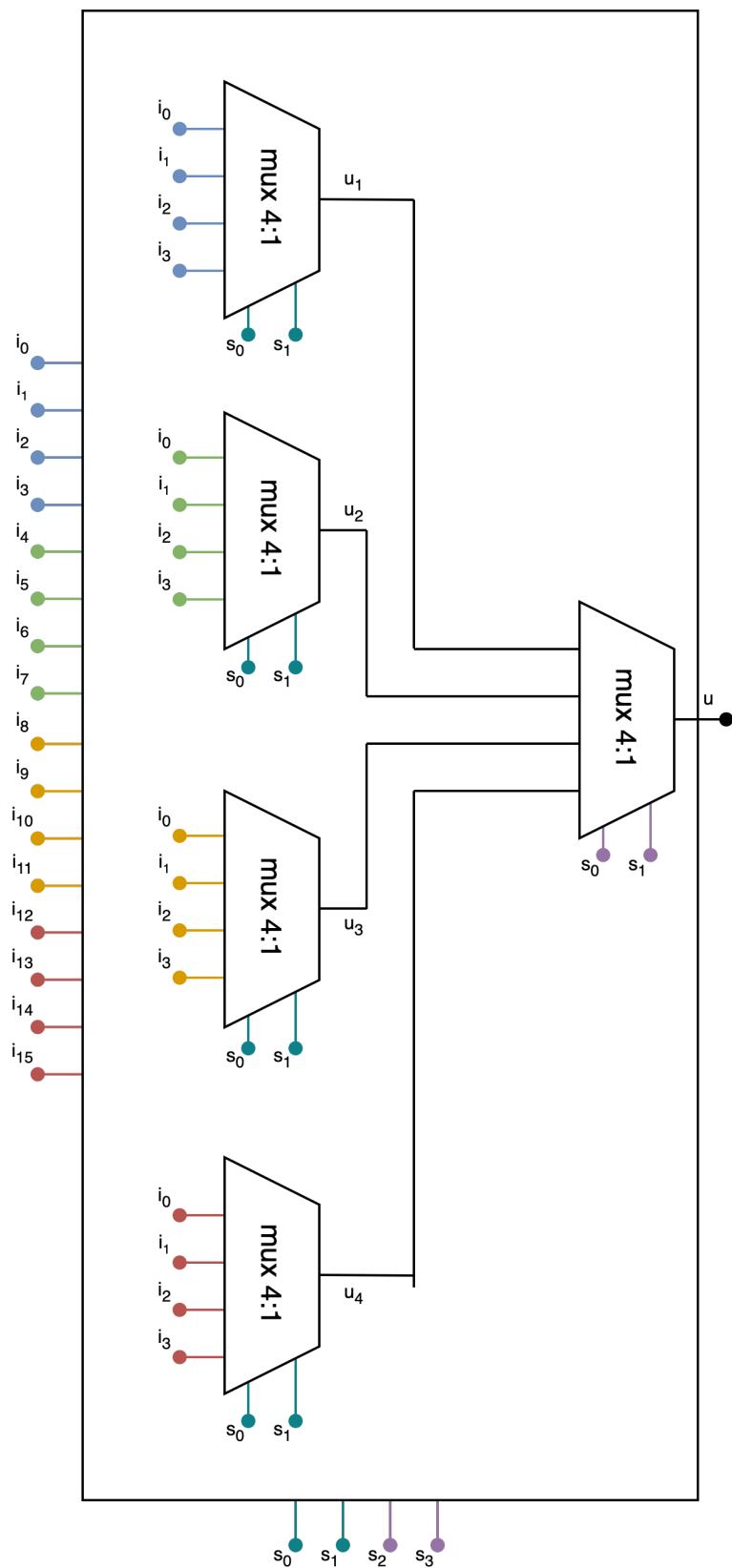


Figura 1.6: multiplexer 16:1 per composizione di multiplexer 4:1

### 1.1.2 Implementazione

Per l'implementazione si procede con un approccio di tipo strutturale, iniziando quindi dalla codifica del multiplexer 2:1, e, a partire da questo si compongono dispositivi sempre più complessi fino ad arrivare all'obiettivo del multiplexer 16:1.

**Mux 2:1** Di seguito il codice riguardante il Mux 2:1.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity mux_21 is
5     port
6     (
7         i0, i1: in std_logic;
8         s0: in std_logic;
9         u: out std_logic
10    );
11 end mux_21;
12
13 architecture rtl of mux_21 is
14 begin
15     u <= i0 when s0='0' else
16         i1 when s0='1' else
17             ' ';
18 end rtl;
```

---

Code 1.1: Multiplexer 2:1 in VHDL

L'interfaccia del componente ha come ingressi `i0` ed `i1`, come selezione `s0` e come uscita `u`.

Al seguito della definizione dell'interfaccia, si definisce il comportamento dell'entità, che risponde alla tabella della verità 1.1.

**Mux 4:1** Si prosegue con il Mux 4:1.

Come anticipato, viene costruito a partire da tre mux 2:1.

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity mux_41 is
5     port
6     (
7         i: in std_logic_vector(0 to 3);
8         s: in std_logic_vector(1 downto 0);
9         u: out std_logic
10    );
11 end mux_41;
12
13 architecture structural of mux_41 is
14     signal u_mid: std_logic_vector(0 to 1);
15
16     component mux_21 is
17         port (
18             i0, i1: in std_logic;
19             s0: in std_logic;
20             u: out std_logic
21         );
22     end component;
23
24 begin
25     mux0to1: FOR k IN 0 TO 1 GENERATE
26         m: mux_21
27             port map
28             (
29                 i(k*2),
30                 i(k*2 + 1),
31                 s(0),
32                 u_mid(k)
33             );
34     end GENERATE;
35
36     mux_2: mux_21
37         port map
38         (
39             u_mid(0),
40             u_mid(1),
41             s(1),
```

```
42           u
43       );
44
45 end structural;
46
```

Code 1.2: Multiplexer 4:1 in VHDL

In quest'entità, l'interfaccia è dichiarata come segue:

- Il parametro `i` vettore di 4 elementi, ognuno corrispondente ad un ingresso del mux 4:1.
- Il parametro `s` vettore di 2 elementi, ognuno corrispondente ad un ingresso di selezione.
- Il parametro `u` corrispondente all'uscita del multiplexer.

A seguire si definisce la struttura del mux 4:1, utilizzando mux 2:1 come componenti.

Con il ciclo `for`, vengono stanziati i primi due mux 2:1, i quali riceveranno in ingresso rispettivamente, gli ingressi del mux 4:1 e la loro uscita è il vettore d'appoggio `u_mid`, il quale è talvolta l'ingresso del terzo mux 2:1.

**Mux 16:1** In maniera analoga si procede con la costruzione del mux 16:1. Il codice è il seguente:

---

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity mux_161 is
```

```
5      port
6      (
7          i: in std_logic_vector(0 to 15);
8          s: in std_logic_vector(3 downto 0);
9          u: out std_logic
10     );
11 end mux_161;
12
13 architecture structural of mux_161 is
14
15     signal u_mid: std_logic_vector(0 to 3);
16
17     component mux_41 is
18         port
19         (
20             i: in std_logic_vector(0 to 3);
21             s: in std_logic_vector(0 to 1);
22             u: out std_logic
23         );
24     end component;
25
26 begin
27     mux0to3: FOR k IN 0 TO 3 GENERATE
28         m: mux_41
29         port map
30         (
31             i => i((k*4) to (k*4 + 3)),
32             s => s(1 downto 0),
33             u => u_mid(k)
34         );
35     end GENERATE;
36
37     mux_2: mux_41
38         port map
39         (
40             i => u_mid,
41             s => s(3 downto 2),
42             u => u
43         );
44
45 end structural;
```

Code 1.3: Multiplexer 16:1 in VHDL

### 1.1.3 Simulazione

Per la simulazione, vi è la necessità di un testbench, il quale generiamo in maniera automatica tramite software appositi.

In tale progetto la generazione viene effettuata tramite ChatGPT ed il codice è il seguente:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.numeric_std.all; -- Libreria necessaria per `to_unsigned`  

4
5 entity tb_mux_161 is
6 end tb_mux_161;  

7
8 architecture behavior of tb_mux_161 is
9     -- Component declaration
10    component mux_161
11        port (
12            i: in std_logic_vector(0 to 15);
13            s: in std_logic_vector(3 downto 0);
14            u: out std_logic
15        );
16    end component;  

17
18    -- Signals for testing
19    signal i: std_logic_vector(0 to 15);
20    signal s: std_logic_vector(3 downto 0);
21    signal u: std_logic;  

22
23 begin
24     -- Instantiate the unit under test (UUT)
25     uut: mux_161
26         port map (
27             i => i,
28             s => s,
29             u => u
30         );
31
32     -- Test process
33     stim_proc: process
34         variable expected_output: std_logic; -- Variabile per il  

35             controllo
```

```
35 begin
36     -- Initialize inputs
37     i <= (others => '0');
38     s <= "0000";
39     wait for 10 ns;
40
41     -- Apply test cases
42     for sel in 0 to 15 loop
43         -- Set the ith bit of i to '1'
44         i <= (others => '0');
45         i(sel) <= '1';
46
47         -- Set the selector
48         s <= std_logic_vector(to_unsigned(sel, 4));
49
50         -- Aspetta che l'uscita si stabilizzi
51         wait for 10 ns;
52
53         -- Calcola l'uscita attesa
54         expected_output := i(sel);
55
56         -- Controlla se l'uscita è corretta
57         if u = expected_output then
58             report "Test passed for s = " & integer'image(sel) &
59                     ", u = " & std_logic'image(u);
60         else
61             report "Test failed for s = " & integer'image(sel) &
62                     ": expected = " &
63                     ← std_logic'image(expected_output) &
64                     ", got = " & std_logic'image(u)
65                     severity error;
66         end if;
67     end loop;
68
69     -- Fine simulazione
70     report "All tests completed";
71     wait;
72 end process;
73 end behavior;
```

Code 1.4: Testbench multiplexer 16:1 in VHDL

Una volta generato ciò, utilizzando i software GHDL e GTKWAVE,

vengono eseguiti i seguenti comandi:

```
[antoniosirignano@Antonios-MacBook-Pro Problema_1 % ghdl -a mux_2_1.vhdl
[antoniosirignano@Antonios-MacBook-Pro Problema_1 % ghdl -a mux_4_1.vhdl
[antoniosirignano@Antonios-MacBook-Pro Problema_1 % ghdl -a mux_16_1.vhdl
[antoniosirignano@Antonios-MacBook-Pro Problema_1 % ghdl -a tb_mux_16_1.vhdl
[antoniosirignano@Antonios-MacBook-Pro Problema_1 % ghdl -e tb_mux_161
antoniosirignano@Antonios-MacBook-Pro Problema_1 % ghdl -r tb_mux_161 --wave=mux_16_1.ghw

tb_mux_16_1.vhdl:58:17:@20ns:(report note): Test passed for s = 0, u = '1'
tb_mux_16_1.vhdl:58:17:@30ns:(report note): Test passed for s = 1, u = '1'
tb_mux_16_1.vhdl:58:17:@40ns:(report note): Test passed for s = 2, u = '1'
tb_mux_16_1.vhdl:58:17:@50ns:(report note): Test passed for s = 3, u = '1'
tb_mux_16_1.vhdl:58:17:@60ns:(report note): Test passed for s = 4, u = '1'
tb_mux_16_1.vhdl:58:17:@70ns:(report note): Test passed for s = 5, u = '1'
tb_mux_16_1.vhdl:58:17:@80ns:(report note): Test passed for s = 6, u = '1'
tb_mux_16_1.vhdl:58:17:@90ns:(report note): Test passed for s = 7, u = '1'
tb_mux_16_1.vhdl:58:17:@100ns:(report note): Test passed for s = 8, u = '1'
tb_mux_16_1.vhdl:58:17:@110ns:(report note): Test passed for s = 9, u = '1'
tb_mux_16_1.vhdl:58:17:@120ns:(report note): Test passed for s = 10, u = '1'
tb_mux_16_1.vhdl:58:17:@130ns:(report note): Test passed for s = 11, u = '1'
tb_mux_16_1.vhdl:58:17:@140ns:(report note): Test passed for s = 12, u = '1'
tb_mux_16_1.vhdl:58:17:@150ns:(report note): Test passed for s = 13, u = '1'
tb_mux_16_1.vhdl:58:17:@160ns:(report note): Test passed for s = 14, u = '1'
tb_mux_16_1.vhdl:58:17:@170ns:(report note): Test passed for s = 15, u = '1'
tb_mux_16_1.vhdl:69:9:@170ns:(report note): All tests completed
[antoniosirignano@Antonios-MacBook-Pro Problema_1 % gtkwave mux_16_1.ghw

GTKWave Analyzer v3.4.0 (w)1999-2022 BSI

[0] start time.
[170000000] end time.
2024-11-25 18:54:06.970 gtkwave[78165:3049537] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-25 18:54:06.970 gtkwave[78165:3049537] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

Figura 1.7: Comandi per la simulazione

Con l'esecuzione dell'ultimo comando, vi si apre una nuova finestra che permette la visualizzazione delle onde:

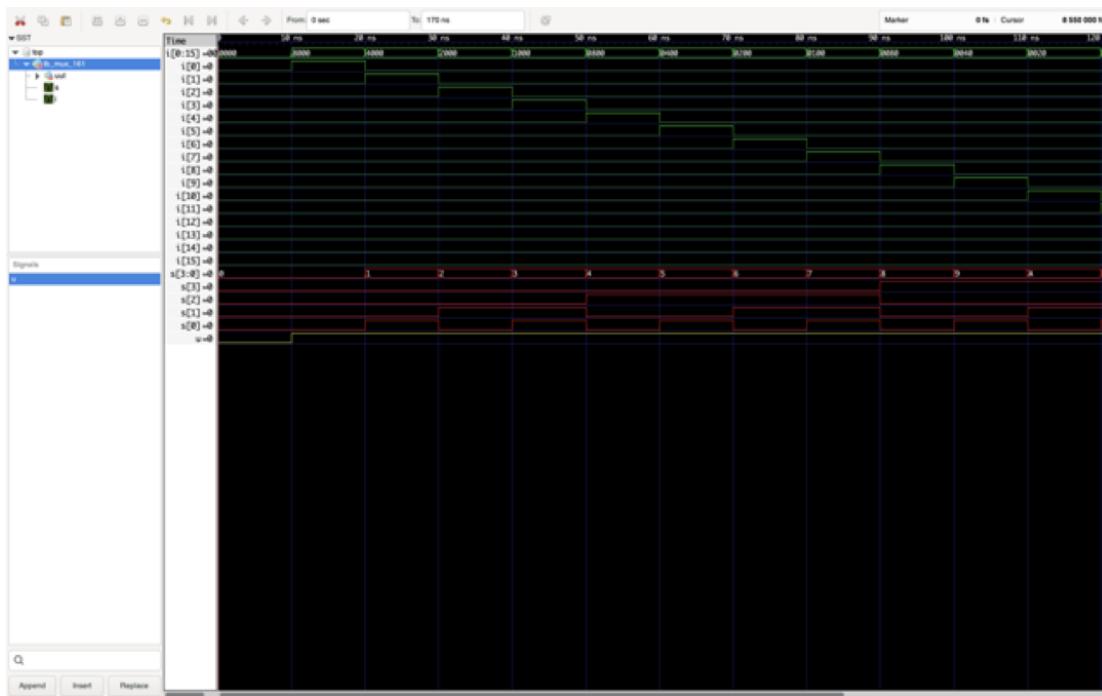


Figura 1.8: Risultati della simulazione: waveform

#### 1.1.4 Implementazione 2.0

In alcuni casi, può essere utile specificare i singoli ingressi, in particolare quando gli ingressi provengono da fonti diverse; in tal caso, è preferibile l'implementazione che segue:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity mux16_1 is
5     port(      i0 : in STD_LOGIC;
6                 i1 : in STD_LOGIC;
7                 i2 : in STD_LOGIC;
8                 i3 : in STD_LOGIC;
9                 i4 : in STD_LOGIC;
10                i5 : in STD_LOGIC;
11                i6 : in STD_LOGIC;
12                i7 : in STD_LOGIC;
13                i8 : in STD_LOGIC;
14                i9 : in STD_LOGIC;
```

```

15                               i10 : in STD_LOGIC;
16                               i11 : in STD_LOGIC;
17                               i12 : in STD_LOGIC;
18                               i13 : in STD_LOGIC;
19                               i14 : in STD_LOGIC;
20                               i15 : in STD_LOGIC;
21                               s0 : in STD_LOGIC;
22                               s1 : in STD_LOGIC;
23                               s2 : in STD_LOGIC;
24                               s3 : in STD_LOGIC;
25                               y0 : out STD_LOGIC
26 );
27 end mux16_1;
28
29 architecture structural of mux16_1 is
30     signal u0 : STD_LOGIC := '0';
31     signal u1 : STD_LOGIC := '0';
32     signal u2 : STD_LOGIC := '0';
33     signal u3 : STD_LOGIC := '0';
34
35 component mux_2_1
36     port(          a0      : in STD_LOGIC;
37                      a1      : in STD_LOGIC;
38                      s       : in STD_LOGIC;
39                      y       : out STD_LOGIC
40 );
41 end component;
42
43 component mux_4_1
44     port(      b0 : in STD_LOGIC;
45                 b1 : in STD_LOGIC;
46                 b2 : in STD_LOGIC;
47                 b3 : in STD_LOGIC;
48                 s0 : in STD_LOGIC;
49                 s1 : in STD_LOGIC;
50                 y0 : out STD_LOGIC
51 );
52 end component;
53
54 begin
55     mux_0: mux_4_1
56         Port map(   b0 => i0,
57                         b1 => i1,
58                         b2 => i2,
59                         b3 => i3,
60                         s0 => s0,

```

```
61           s1 => s1,
62           y0 => u0
63       );
64
65   mux_1: mux_4_1
66   Port map(  b0 => i4,
67               b1 => i5,
68               b2 => i6,
69               b3 => i7,
70               s0 => s0,
71               s1 => s1,
72               y0 => u1
73 );
74   mux_2: mux_4_1
75   Port map(  b0 => i8,
76               b1 => i9,
77               b2 => i10,
78               b3 => i11,
79               s0 => s0,
80               s1 => s1,
81               y0 => u2
82 );
83
84
85   mux_3: mux_4_1
86   Port map(  b0 => i12,
87               b1 => i13,
88               b2 => i14,
89               b3 => i15,
90               s0 => s0,
91               s1 => s1,
92               y0 => u3
93 );
94
95   mux_4: mux_4_1
96   Port map(  b0 => u0,
97               b1 => u1,
98               b2 => u2,
99               b3 => u3,
100              s0 => s2,
101              s1 => s3,
102              y0 => y0
103 );
104
105 end structural;
```

Code 1.5: Multiplexer 16:1 in VHDL: ingressi trattati separatamente

Ovviamente, la macchina sarà fatta allo stesso modo, come si può vedere dallo schematico generato da Vivado:

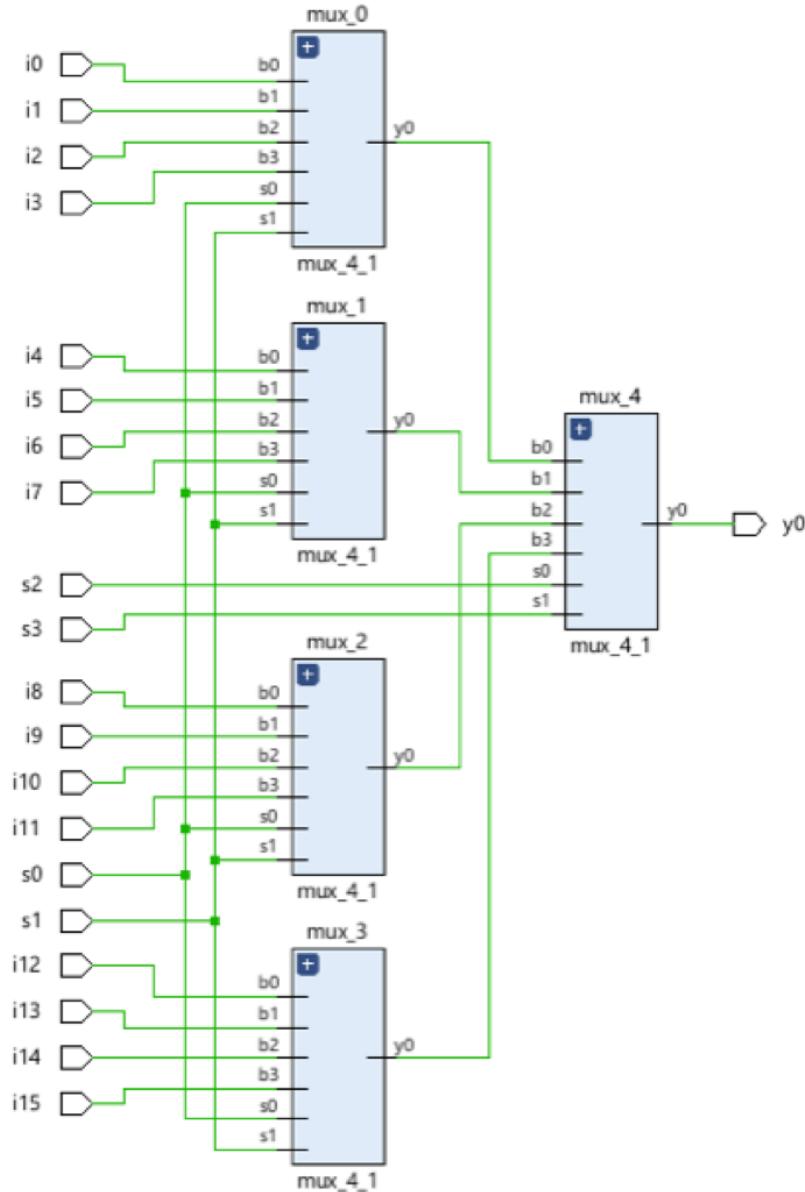


Figura 1.9: Schematic Vivado: Mux 16:1

Il multiplexer lavorerà allo stesso modo, con gli stessi risultati simulativi.

## 1.2 Rete di interconnessione a 16 ingressi e 4 uscite

Una rete di interconnessione è un tipo di rete di commutazione che permette di instradare i segnali da un insieme di ingressi a un insieme più ridotto di uscite. Tale rete può essere progettata attraverso un adeguato utilizzo di Multiplexer e Demultiplexer.

Nel caso in esame, si vuole progettare una rete che prenda 16 ingressi e restituisca 4 uscite. Si utilizza anche in questo caso un approccio per composizione, a partire dal Multiplexer 16:1 implementato nell'esercizio precedente, la cui uscita sarà posta in ingresso a un Demultiplexer 1:4.

La rete complessiva sarà fatta in questo modo:

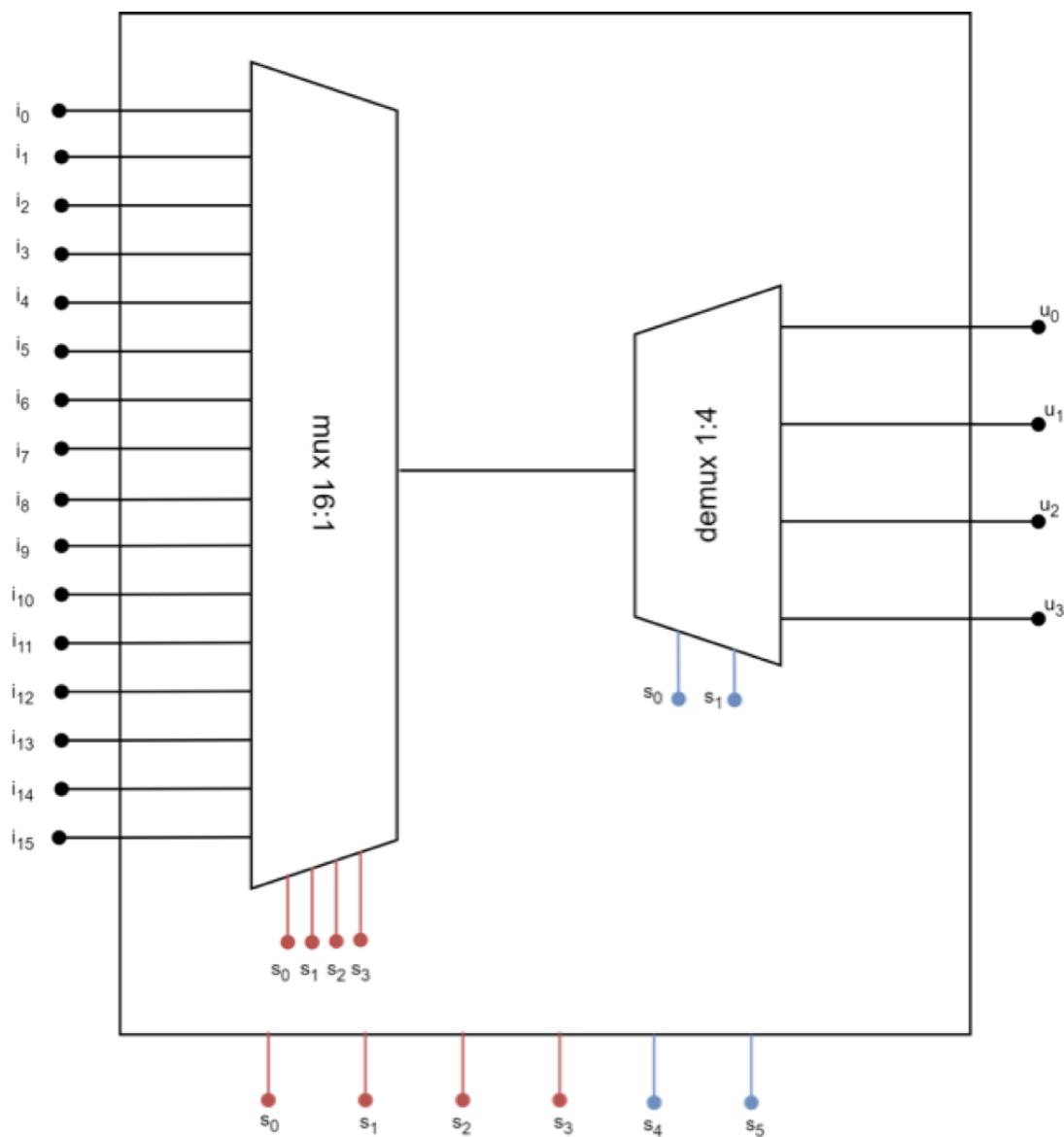


Figura 1.10: Rete di interconnessione

### 1.2.1 Progettazione

Anche in questo caso, prima di procedere all'implementazione della rete nel complesso, si costruisce il Demultiplexer 4:1 a partire da Demultiplexer 2:1.

Un Demultiplexer  $1 : u$  è un dispositivo che prende un solo segnale di

ingresso, due segnali di selezione e a partire da essi restituisce  $u$  uscite.

Un Demultiplexer 2:1 è un dispositivo fatto in questo modo:

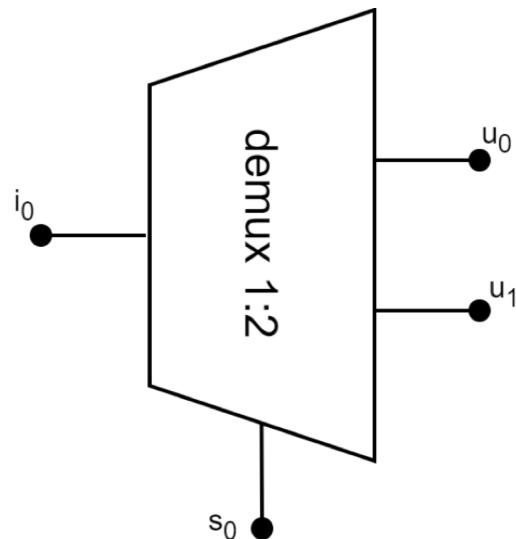


Figura 1.11: Demux 1:2

Tale componente è caratterizzato dalla seguente tabella di verità:

<b>s<sub>0</sub></b>	<b>i<sub>0</sub></b>	<b>u<sub>0</sub></b>	<b>u<sub>1</sub></b>
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

Tabella 1.2: Tabella di verità di un Demux 2:1

Da cui si ricavano le seguenti equazioni relative alle uscite:

$$u_0 = (i_0 \text{ AND } \bar{s}_0)$$

$$u_1 = (i_0 \text{ AND } s_0)$$

A partire dalla composizione di dispositivi di questo tipo, si può realizzare un Demultiplexer 1:4, come rappresentato in figura.

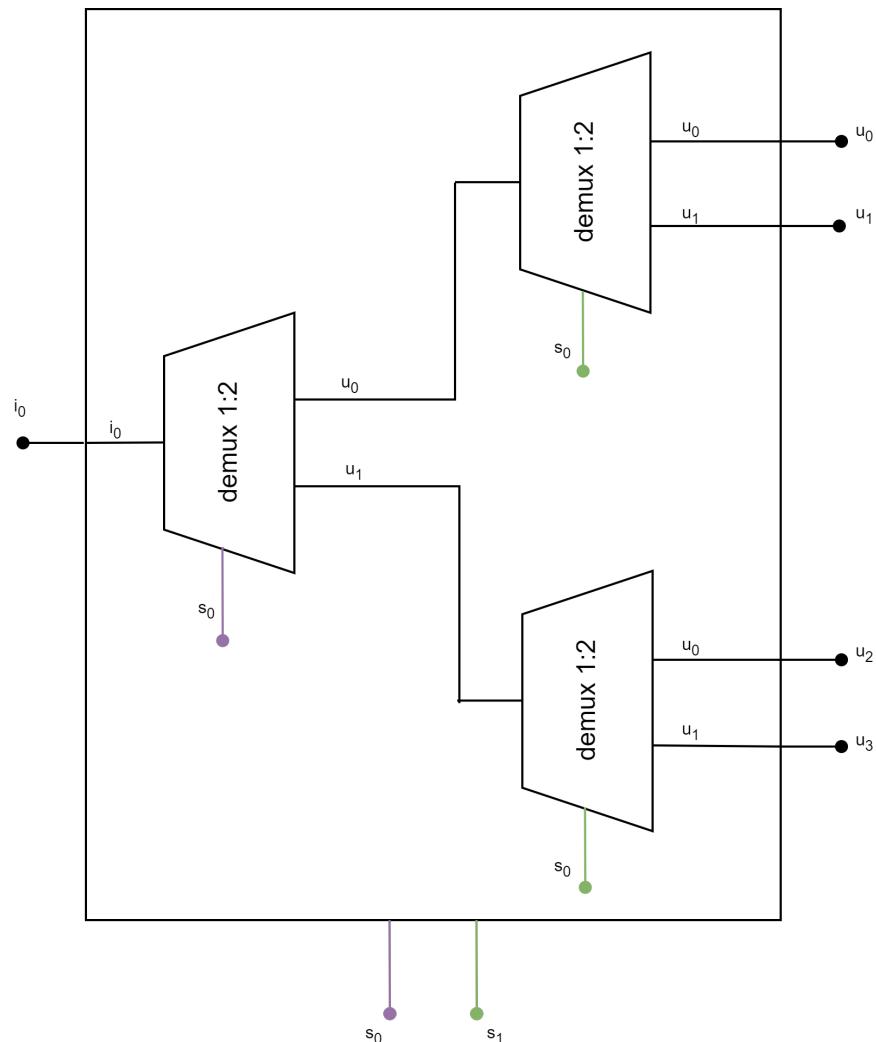


Figura 1.12: Demux 1:4 composto a partire da Demux 1:2

Utilizzando il Demultiplexer appena progettato, al cui ingresso si fa corrispondere l'uscita del Multiplexer 16:1, progettato nell'esercizio precedente, si ottiene la rete di interconnessione, così formata:

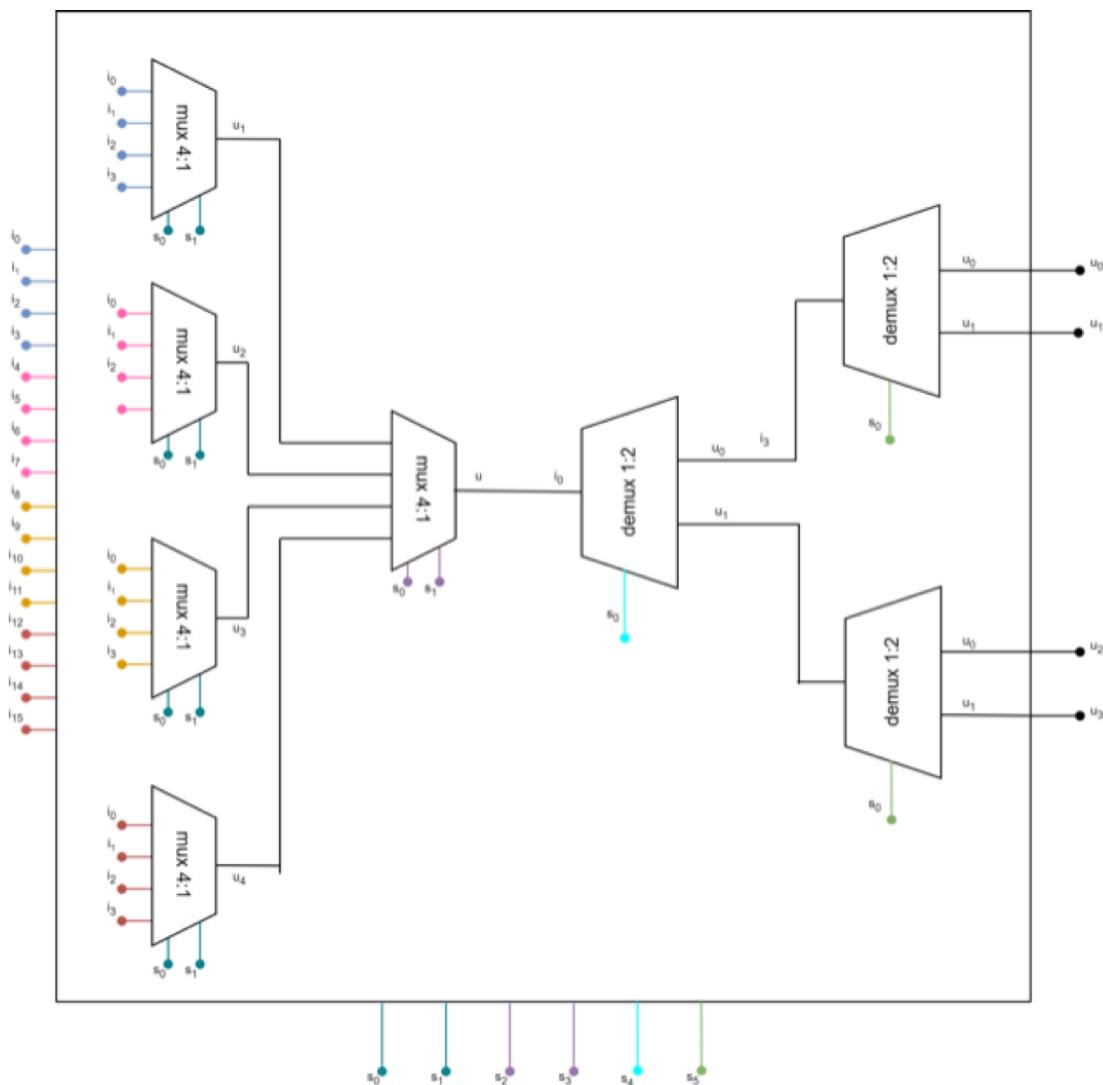


Figura 1.13: Rete di interconnessione: funzionamento interno

Nell'immagine, i colori sono stati usati per rendere più chiari i collegamenti tra segnali.

### 1.2.2 Implementazione

Si inizia mostrando l'implementazione del Demultiplexer 1:2, fatta seguendo un'architettura di tipo Dataflow.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity demux1_2 is
5     port(
6         a0 :in STD_LOGIC;
7         s0 :in STD_LOGIC;
8         y0 :out STD_LOGIC;
9         y1 :out STD_LOGIC
10    );
11 end demux1_2;
12
13
14
15 architecture dataflow of demux1_2 is
16
17 begin
18     y0 <= (not s0 AND a0);
19     y1 <= (s0 AND a0);
20
21
22 end dataflow;
```

Code 1.6: Demultiplexer 1:2

Come mostrato dalla figura 1.12 presente nella fase di progettazione, a partire da 3 demux 1:2 si può realizzare un demux 1:4 seguendo un approccio di tipo strutturale. Segue il codice:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity demux1_4 is
5     port(
6         i0 :in STD_LOGIC;
7         s0 :in STD_LOGIC;
8         s1 :in STD_LOGIC;
9         y0 :out STD_LOGIC;
10        y1 :out STD_LOGIC;
11        y2 :out STD_LOGIC;
12        y3 :out STD_LOGIC
13    );
```

```
14 end demux1_4;
15
16 architecture structural of demux1_4 is
17     signal u0: STD_LOGIC := '0';
18     signal u1: STD_LOGIC := '0';
19
20
21 component demux1_2
22     port(
23         a0: in STD_LOGIC;
24         s0: in STD_LOGIC;
25         y0: out STD_LOGIC;
26         y1: out STD_LOGIC
27     );
28 end component;
29
30 begin
31
32     demux0: demux1_2
33         Port map(
34             a0 =>i0,
35             s0 =>s0,
36             y0 =>u0,
37             y1 =>u1
38         );
39     demux1: demux1_2
40         Port map(
41             a0 =>u0,
42             s0 =>s1,
43             y0 =>y0,
44             y1 =>y1
45         );
46     demux2: demux1_2
47         Port map(
48             a0 =>u1,
49             s0 =>s1,
50             y0 =>y2,
51             y1 =>y3
52         );
53
54 end structural;
```

Code 1.7: Demultiplexer 1:4

Tramite un'appropriata connessione del Multiplexer realizzato nell'esercizio precedente e il Demux 1:4, si ottiene la rete di interconnessione richiesta:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity interc16_4 is
5     port( i0 : in STD_LOGIC;
6             i1 : in STD_LOGIC;
7             i2 : in STD_LOGIC;
8             i3 : in STD_LOGIC;
9             i4 : in STD_LOGIC;
10            i5 : in STD_LOGIC;
11            i6 : in STD_LOGIC;
12            i7 : in STD_LOGIC;
13            i8 : in STD_LOGIC;
14            i9 : in STD_LOGIC;
15            i10 : in STD_LOGIC;
16            i11 : in STD_LOGIC;
17            i12 : in STD_LOGIC;
18            i13 : in STD_LOGIC;
19            i14 : in STD_LOGIC;
20            i15 : in STD_LOGIC;
21            s0 : in STD_LOGIC;
22            s1 : in STD_LOGIC;
23            s2 : in STD_LOGIC;
24            s3 : in STD_LOGIC;
25            s4 : in STD_LOGIC;
26            s5 : in STD_LOGIC;
27            y0 : out STD_LOGIC;
28            y1 : out STD_LOGIC;
29            y2 : out STD_LOGIC;
30            y3 : out STD_LOGIC
31        );
32    end interc16_4;
33
34 architecture structural of interc16_4 is
35 signal a0 : STD_LOGIC;
36
37 component mux16_1
38     port(
39             i0 : in STD_LOGIC;
```

```
40          i2 : in STD_LOGIC;
41          i3 : in STD_LOGIC;
42          i4 : in STD_LOGIC;
43          i5 : in STD_LOGIC;
44          i6 : in STD_LOGIC;
45          i7 : in STD_LOGIC;
46          i8 : in STD_LOGIC;
47          i9 : in STD_LOGIC;
48          i10 : in STD_LOGIC;
49          i11 : in STD_LOGIC;
50          i12 : in STD_LOGIC;
51          i13 : in STD_LOGIC;
52          i14 : in STD_LOGIC;
53          i15 : in STD_LOGIC;
54          s0 : in STD_LOGIC;
55          s1 : in STD_LOGIC;
56          s2 : in STD_LOGIC;
57          s3 : in STD_LOGIC;
58          y0 : out STD_LOGIC
59      );
60  end component;
61
62 component demux1_4
63     port(      i0      : in STD_LOGIC;
64                 s0      : in STD_LOGIC;
65                 s1      : in STD_LOGIC;
66                 y0      : out STD_LOGIC;
67                 y1      : out STD_LOGIC;
68                 y2      : out STD_LOGIC;
69                 y3      : out STD_LOGIC
70      );
71  end component;
72
73
74
75 begin
76     mux_0: mux16_1
77         Port map(
78             i0 => i0,
79             i1 => i1,
80             i2 => i2,
81             i3 => i3,
82             i4 => i4,
83             i5 => i5,
84             i6 => i6,
85             i7 => i7,
```

```
86          i8 => i8,
87          i9 => i9,
88          i10 => i10,
89          i11 => i11,
90          i12 => i12,
91          i13 => i13,
92          i14 => i14,
93          i15 => i15,
94          s0 => s0,
95          s1 => s1,
96          s2 => s2,
97          s3 => s3,
98          y0 => a0
99      );
100
101     demux0: demux1_4
102     Port map(
103         i0 => a0,
104         s0 => s4,
105         s1 => s5,
106         y0 => y0,
107         y1 => y1,
108         y2 => y2,
109         y3 => y3
110     );
111
112 end structural;
113
```

Code 1.8: Rete di interconnessione 16:4 in VHDL

La rete realizzata è osservabile come schematic generato da Vivado:

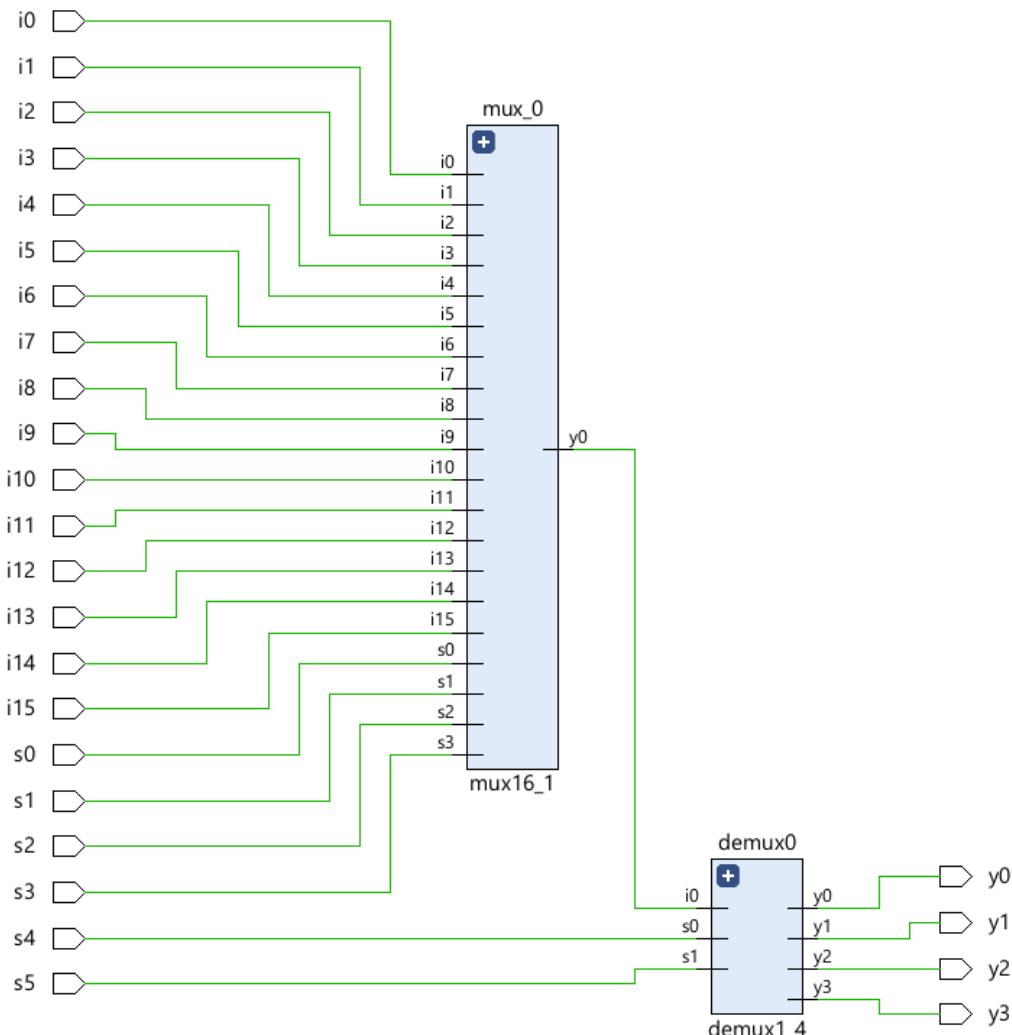


Figura 1.14: Rete di interconnessione: schematic

### 1.2.3 Simulazione

Per procedere con la simulazione della rete realizzata, si utilizza un tesbench. Tale testebnch è stato realizzato tramite il sito Doulos, e sono stati manualmente aggiunti diversi casi di test:

```

1 library IEEE;
2 use IEEE.Std_logic_1164.all;
3 use IEEE.Numeric_Signed.all;
4
5 entity interc16_4_tb is

```

```
6 end;
7
8 architecture bench of interc16_4_tb is
9
10 component interc16_4
11     port( i0 : in STD_LOGIC;
12             i1 : in STD_LOGIC;
13             i2 : in STD_LOGIC;
14             i3 : in STD_LOGIC;
15             i4 : in STD_LOGIC;
16             i5 : in STD_LOGIC;
17             i6 : in STD_LOGIC;
18             i7 : in STD_LOGIC;
19             i8 : in STD_LOGIC;
20             i9 : in STD_LOGIC;
21             i10 : in STD_LOGIC;
22             i11 : in STD_LOGIC;
23             i12 : in STD_LOGIC;
24             i13 : in STD_LOGIC;
25             i14 : in STD_LOGIC;
26             i15 : in STD_LOGIC;
27             s0 : in STD_LOGIC;
28             s1 : in STD_LOGIC;
29             s2 : in STD_LOGIC;
30             s3 : in STD_LOGIC;
31             s4 : in STD_LOGIC;
32             s5 : in STD_LOGIC;
33             y0 : out STD_LOGIC;
34             y1 : out STD_LOGIC;
35             y2 : out STD_LOGIC;
36             y3 : out STD_LOGIC
37         );
38     end component;
39
40 signal i0: STD_LOGIC;
41 signal i1: STD_LOGIC;
42 signal i2: STD_LOGIC;
43 signal i3: STD_LOGIC;
44 signal i4: STD_LOGIC;
45 signal i5: STD_LOGIC;
46 signal i6: STD_LOGIC;
47 signal i7: STD_LOGIC;
48 signal i8: STD_LOGIC;
49 signal i9: STD_LOGIC;
50 signal i10: STD_LOGIC;
51 signal i11: STD_LOGIC;
```

```
52  signal i12: STD_LOGIC;
53  signal i13: STD_LOGIC;
54  signal i14: STD_LOGIC;
55  signal i15: STD_LOGIC;
56  signal s0: STD_LOGIC;
57  signal s1: STD_LOGIC;
58  signal s2: STD_LOGIC;
59  signal s3: STD_LOGIC;
60  signal s4: STD_LOGIC;
61  signal s5: STD_LOGIC;
62  signal y0: STD_LOGIC;
63  signal y1: STD_LOGIC;
64  signal y2: STD_LOGIC;
65  signal y3: STD_LOGIC ;
66
66
67 begin
68
69  uut: interc16_4 port map ( i0    => i0,
70                            i1    => i1,
71                            i2    => i2,
72                            i3    => i3,
73                            i4    => i4,
74                            i5    => i5,
75                            i6    => i6,
76                            i7    => i7,
77                            i8    => i8,
78                            i9    => i9,
79                            i10   => i10,
80                            i11   => i11,
81                            i12   => i12,
82                            i13   => i13,
83                            i14   => i14,
84                            i15   => i15,
85                            s0    => s0,
86                            s1    => s1,
87                            s2    => s2,
88                            s3    => s3,
89                            s4    => s4,
90                            s5    => s5,
91                            y0    => y0,
92                            y1    => y1,
93                            y2    => y2,
94                            y3    => y3 ) ;
95
96  stimulus: process
97  begin
```

```

98
99    -- Inizializzazione segnali
100   i0 <= '0'; i1 <= '0'; i2 <= '0'; i3 <= '0';
101   i4 <= '0'; i5 <= '0'; i6 <= '0'; i7 <= '0';
102   i8 <= '0'; i9 <= '0'; i10 <= '0'; i11 <= '0';
103   i12 <= '0'; i13 <= '0'; i14 <= '0'; i15 <= '0';
104   s0 <= '0'; s1 <= '0'; s2 <= '0'; s3 <= '0';
105   s4 <= '0'; s5 <= '0';
106   wait for 10 ns;
107
108   -- Test Case 1: Selezione i0
109   i0 <= '1'; s0 <= '0'; s1 <= '0'; s2 <= '0'; s3 <= '0'; s4 <= '0';
110   ↵ s5 <= '0';
111   wait for 10 ns;
112
113   -- Test Case 2: Selezione i3
114   i3 <= '1'; s0 <= '1'; s1 <= '1'; s2 <= '0'; s3 <= '0'; s4 <= '0';
115   ↵ s5 <= '0';
116   wait for 10 ns;
117
118   -- Test Case 3: Selezione i7
119   i7 <= '1'; s0 <= '1'; s1 <= '1'; s2 <= '1'; s3 <= '0'; s4 <= '0';
120   ↵ s5 <= '0';
121   wait for 10 ns;
122
123
124   -- Test Case 4: Selezione i12
125   i12 <= '1'; s0 <= '0'; s1 <= '0'; s2 <= '0'; s3 <= '1'; s4 <=
126   ↵ '1'; s5 <= '0';
127   wait for 10 ns;
128
129   -- Test Case 5: Selezione i15
130   i15 <= '1'; s0 <= '1'; s1 <= '1'; s2 <= '1'; s3 <= '1'; s4 <=
131   ↵ '1'; s5 <= '0';
132   wait for 10 ns;
133
134   -- Test Case 6: Nessun ingresso attivo
135   i0 <= '0'; i1 <= '0'; i2 <= '0'; i3 <= '0';
136   i4 <= '0'; i5 <= '0'; i6 <= '0'; i7 <= '0';
137   i8 <= '0'; i9 <= '0'; i10 <= '0'; i11 <= '0';
138   i12 <= '0'; i13 <= '0'; i14 <= '0'; i15 <= '0';
139   s0 <= '0'; s1 <= '0'; s2 <= '0'; s3 <= '0'; s4 <= '0'; s5 <=
140   ↵ '0';
141   wait for 10 ns;
142
143
144   -- Test Case 7: Selezione i5

```

```
137      i5 <= '1'; s0 <= '1'; s1 <= '0'; s2 <= '1'; s3 <= '0'; s4 <= '0';
138      ↵ s5 <= '0';
139      wait for 10 ns;
140
140      -- Test Case 8: Selezione i10
141      i10 <= '1'; s0 <= '0'; s1 <= '1'; s2 <= '0'; s3 <= '1'; s4 <=
142          ↵ '0'; s5 <= '0';
142      wait for 10 ns;
143
144      -- Test Case 9: Selezione i6
145      i6 <= '1'; s0 <= '1'; s1 <= '0'; s2 <= '1'; s3 <= '1'; s4 <=
146          ↵ s5 <= '0';
146      wait for 10 ns;
147
148      -- Test Case 10: Selezione i9
149      i9 <= '1'; s0 <= '0'; s1 <= '1'; s2 <= '1'; s3 <= '0'; s4 <=
150          ↵ s5 <= '0';
150      wait for 10 ns;
151
152      -- Test Case 11: Selezione i14
153      i14 <= '1'; s0 <= '1'; s1 <= '1'; s2 <= '0'; s3 <= '1'; s4 <=
154          ↵ '1'; s5 <= '0';
154      wait for 10 ns;
155
156      -- Fine del test
157      wait;
158      end process;
159
160 end;
```

Code 1.9: Testbench: Rete di interconnessione 16:4

I risultati di tale simulazione sono osservabili nella seguente waveform realizzata dal tool di Vivado.

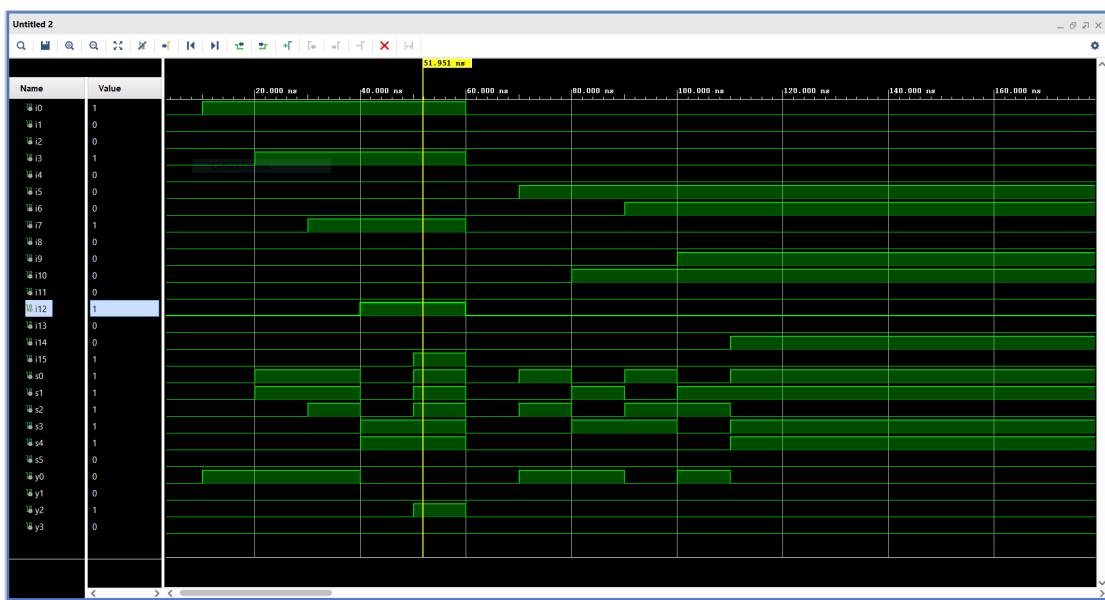


Figura 1.15: Rete di interconnessione: waveform

### 1.3 Implementazione su board del punto precedente

La board utilizzata è la **Nexys A7**, una scheda di sviluppo basata su FPGA progettata da Digilent.

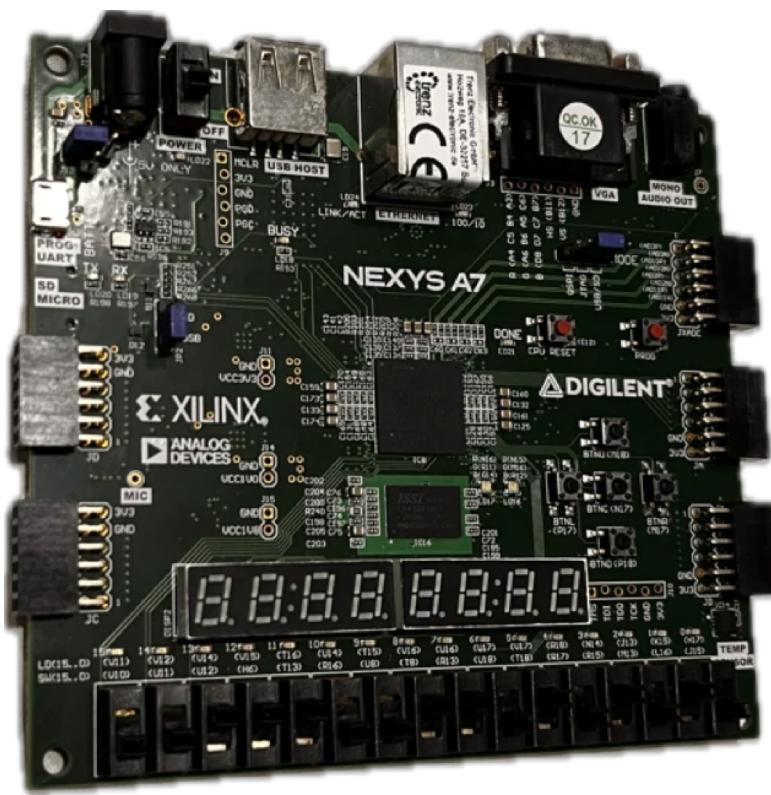


Figura 1.16: Board Nexys A7

### 1.3.1 Traccia

Sintetizzare ed implementare su board il progetto della rete di interconnessione sviluppato al punto 1.2, utilizzando gli switch per fornire gli input di selezione e i led per visualizzare i 4 bit di uscita. Per quanto riguarda i 16 bit dato in input, essi devono essere immessi mediante switch, 8 bit alla volta, sviluppando un'apposita “rete di controllo” per l’acquisizione che utilizzi due bottoni della board per caricare rispettivamente la prima e la seconda metà del dato in ingresso.

### 1.3.2 Implementazione

Per permettere lo sviluppo sulla board, è stato necessario gestire gli input in modo appropriato; per fare ciò che viene richiesto, si è scelto di usare il bottone *BTNL* per il caricamento della prima metà degli ingressi, il bottone *BTNR* per il caricamento della seconda metà degli ingressi, e il bottone *BTNU* per il caricamento dei segnali di selezione; inoltre è stato previsto un bottone per il reset, *BTNC*. Gli ingressi sono stati gestiti con gli switch, e le uscite sono visualizzabili tramite i led. I primi 8 switch (da 0 a 7) sono stati utilizzati per gli ingressi, mentre i successivi 6 (da 8 a 13) per le selezioni. I led utilizzati per le uscite sono invece i primi 4 (da 0 a 3). Per permettere opportune connessioni tra i componenti hardware e i segnali utilizzati nella rete di interconnessione, è stata implementata una unità di controllo, che ha gestito gli ingressi in due fasi distinte, oltre che i segnali di selezione.

Segue il codice dell'unità di controllo:

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity control_unit is
5     Port (
6         clock : in STD_LOGIC;
7         reset : in STD_LOGIC;
8         load_first_part : in STD_LOGIC;
9         load_second_part : in STD_LOGIC;
10        load_selection: in STD_LOGIC;
11        value8_in : in STD_LOGIC_VECTOR(7 downto 0);
12        --valore acquisito dai primi 8 switch
13        value16_out: out STD_LOGIC_VECTOR(15 downto 0);
14        selection_in: in STD_LOGIC_VECTOR(5 downto 0);
15        sel_out: out STD_LOGIC_VECTOR(5 downto 0)
```

---

```
15          );
16 end control_unit;
17
18 architecture Behavioral of control_unit is
19
20 signal reg_value : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
21 signal selection_value: STD_LOGIC_VECTOR(5 downto 0);
22
23 begin
24 value16_out <= reg_value;
25 sel_out <= selection_value;
26
27 main: process(clock)
28 begin
29
30     if clock'event and clock = '1' then
31         if reset = '1' then
32             reg_value <= (others => '0');
33         else
34             if load_first_part = '1' then
35                 reg_value(7 downto 0) <= value8_in;
36             elsif load_second_part = '1' then
37                 reg_value(15 downto 8) <= value8_in;
38             elsif load_selection = '1' then
39                 selection_value <= selection_in;
40             end if;
41         end if;
42     end if;
43
44 end process;
45
46
47 end Behavioral;
```

Code 1.10: Control unit

Inoltre, per consentire il funzionamento del sistema sulla board, è stato implementato il seguente codice:

---

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



```
42          i6 : in STD_LOGIC;
43          i7 : in STD_LOGIC;
44          i8 : in STD_LOGIC;
45          i9 : in STD_LOGIC;
46          i10 : in STD_LOGIC;
47          i11 : in STD_LOGIC;
48          i12 : in STD_LOGIC;
49          i13 : in STD_LOGIC;
50          i14 : in STD_LOGIC;
51          i15 : in STD_LOGIC;
52          s0 : in STD_LOGIC;
53          s1 : in STD_LOGIC;
54          s2 : in STD_LOGIC;
55          s3 : in STD_LOGIC;
56          s4 : in STD_LOGIC;
57          s5 : in STD_LOGIC;
58          y0 : out STD_LOGIC;
59          y1 : out STD_LOGIC;
60          y2 : out STD_LOGIC;
61          y3 : out STD_LOGIC
62      );
63 end component;
64
65 signal cu_value: STD_LOGIC_VECTOR(15 downto 0);
66 signal cu_sel: STD_LOGIC_VECTOR( 5 downto 0);
67
68 begin
69 cu: control_unit
70     port map(
71         clock => clock,
72         reset => reset,
73         load_first_part => load_first_part,
74         load_second_part => load_second_part,
75         load_selection => load_selection,
76         value8_in => value8_in,
77         value16_out => cu_value,
78         selection_in => selection_in,
79         sel_out => cu_sel
80     );
81
82 ri: interc16_4
83     port map(
84         i0 => cu_value(0),
85         i1 => cu_value(1),
86         i2 => cu_value(2),
87         i3 => cu_value(3),
```

```
88     i4 => cu_value(4),
89     i5 => cu_value(5),
90     i6 => cu_value(6),
91     i7 => cu_value(7),
92     i8 => cu_value(8),
93     i9 => cu_value(9),
94     i10 => cu_value(10),
95     i11 => cu_value(11),
96     i12 => cu_value(12),
97     i13 => cu_value(13),
98     i14 => cu_value(14),
99     i15 => cu_value(15),
100    s0 => cu_sel(0),
101    s1 => cu_sel(1),
102    s2 => cu_sel(2),
103    s3 => cu_sel(3),
104    s4 => cu_sel(4),
105    s5 => cu_sel(5),
106    y0 => y0,
107    y1 => y1,
108    y2 => y2,
109    y3 => y3
110  );
111
112 end structural;
```

---

Code 1.11: Implementazione: Rete di interconnessione on Board

### 1.3.3 Funzionamento

Di seguito si mostra l'esecuzione su board di uno dei casi di test visti in precedenza nella fase di simulazione. In particolare è stato testato ciò che avveniva a 51 ns, e si può vedere che il led acceso corrisponde con l'uscita attesa  $y2$ .

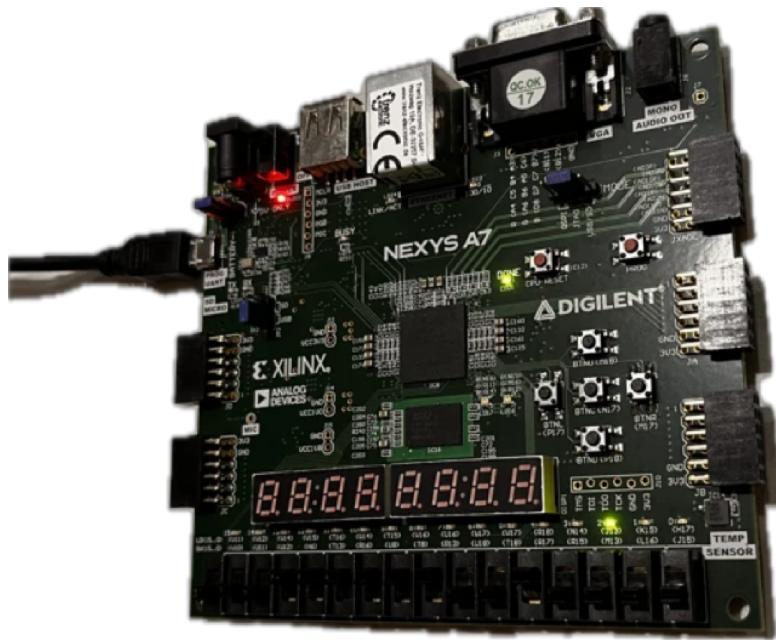


Figura 1.17: Uscita y2 attiva

# Capitolo 2

## Esercizio 2 - Sistema ROM+M

Il sistema che si vuole costruire consiste in due elementi principali: una ROM (Read-Only-Memory) puramente combinatoria e una macchina combinatoria M, che esegue una trasformazione sui dati letti da M e li pone in uscita. La ROM si compone di 16 locazioni di memoria, ciascuna contenente una stringa di 8 bit. Il sistema prende in ingresso un indirizzo di 4 bit, che permetterà di accedere a una delle locazioni della ROM; il dato in tale locazione viene posto in uscita alla ROM, e quindi in ingresso alla macchina M. La macchina M deve effettuare una trasformazione sulla stringa di 8 bit, in modo da restituire in uscita una stringa di 4 bit. La trasformazione scelta consiste nel sommare i 4 bit più significativi della stringa con i 4 bit meno significativi, la stringa di 4 bit risultante sarà restituita come uscita all'intero sistema.

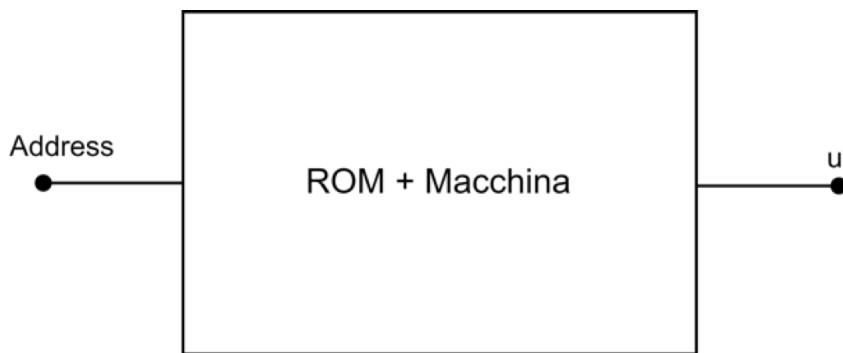


Figura 2.1: ROM + M

## 2.1 Progettazione

La progettazione consiste nella realizzazione dei due componenti fondamentali del sistema: ROM e M.

## 2.2 Implementazione

Dapprima si implementa la ROM, in cui sono memorizzati 16 elementi, ciascuno da 8 bit. Il codice sottostante crea l'entità ROM, al cui ingresso è presente un vettore da 4 bit di `std_logic` che rappresenta l'indirizzo, e in uscita restituisce un vettore di 8 bit. Vengono poi definite le stringhe di bit contenute nella ROM. Nel processo `main`, si pone in uscita l'elemento corrispondente alla locazione `address`.

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5
```

```

6
7 entity ROM is
8     port(
9         address: in STD_LOGIC_VECTOR(3 downto 0);
10        dout: out STD_LOGIC_VECTOR(7 downto 0)
11    );
12 end entity ROM;
13
14 architecture RTL of ROM is
15
16 type MEMORY is array(15 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
17     --memoria da N locazioni che contengono 8 bit
18 constant ROM_N: MEMORY := (
19     "01000000", -- in locazione 15
20     "01000001",
21     "01000010",
22     "01000011",
23     "00010100",
24     "01000101",
25     "00000110",
26     "01000111",
27     "00001000",
28     "00001001",
29     "01001010",
30     "00001011",
31     "00001100",
32     "00001101",
33     "10001010",
34     "00001001" --in locazione 0
35 );
36
37 begin
38 main: process(address)
39 begin
40 dout<= ROM_N(TO_INTEGER(unsigned(address))); --lettura dalla rom
41 end process main;
42 end architecture RTL;

```

Code 2.1: Implementazione ROM in VHDL

Si procede poi con l'implementazione del componente M, che effettua la trasformazione.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity M is
6     port(
7         ingresso: in std_logic_vector(7 downto 0);
8         uscita: out std_logic_vector(3 downto 0)
9     );
10
11 end entity M;
12
13 architecture Behavioral of M is
14 begin
15     process(ingresso)
16     begin
17         -- Somma dei 4 bit piu' significativi e dei 4 meno
18         -- significativi
19         uscita <= std_logic_vector(unsigned(ingresso(7 downto 4)) +
20             unsigned(ingresso(3 downto 0)));
21     end process;
22 end Behavioral;

```

Code 2.2: Macchina M

Nel processo si pone come uscita della macchina la somma tra i bit più significativi dell'ingresso (dal bit 7 al 4) e dei bit meno significativi (dal bit 3 allo 0).

Le due componenti sono parte del sistema S che è così implementato:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity ROMplusM is
6     Port (
7         A: in std_logic_vector(3 downto 0); --indirizzo in ingresso
8             -- al sistema

```

```

8         bout: out std_logic_vector(3 downto 0) --uscita complessiva
9             → del sistema
10        );
11    end ROMplusM;
12
13 architecture structural of ROMplusM is
14 signal u0 : std_logic_vector(7 downto 0) := "00000000";
15
16 component ROM
17     port(
18         address: in STD_LOGIC_VECTOR(3 downto 0);
19         dout: out STD_LOGIC_VECTOR(7 downto 0)
20     );
21     end component;
22 component M
23     port(
24         ingresso: in std_logic_vector(7 downto 0);
25         uscita: out std_logic_vector(3 downto 0)
26     );
27     end component;
28
29 begin
30     -- Istanza della ROM
31     rom_instance: ROM
32     port map(
33         address => A,
34         dout => u0
35     );
36     -- Istanza della macchina combinatoria M
37     transform: M
38     port map(
39         ingresso => u0,
40         uscita => bout
41     );
42 end structural;

```

Code 2.3: Sistema S

Tale sistema è stato costruito come structural: sono stati dichiarati i componenti, e ne sono state definite le istanze. Si è utilizzato un segnale di supporto  $u_0$ , che funge da segnale intermedio tra l'uscita della ROM e l'ingresso della macchina.

Si osserva lo schematic fornito dall'ambiente di sviluppo Vivado:



Figura 2.2: Schematic di S

## 2.3 Simulazione

Per procedere alla simulazione si realizza un testbench, con diversi casi di test, che permettano di osservare il comportamento del sistema.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity ROMplusM_tb is
6     -- Un testbench non ha porte, e' un'entita' vuota.
7 end ROMplusM_tb;
8
9 architecture behavior of ROMplusM_tb is
10
11     -- Component declaration for the unit under test (UUT)
12     component ROMplusM
13         Port (
14             A: in std_logic_vector(3 downto 0);
15             bout: out std_logic_vector(3 downto 0)
16         );
17     end component;
18
19     -- Signals to connect to the UUT
20     signal A_tb: std_logic_vector(3 downto 0) := (others => '0'); --
21     -- Ingresso inizializzato a 0
22     signal bout_tb: std_logic_vector(3 downto 0); -- Uscita

```

```

23 begin
24
25     -- Instantiation of the UUT (Unit Under Test)
26     uut: ROMplusM
27     Port map (
28         A => A_tb,
29         bout => bout_tb
30     );
31
32     -- Stimulus process to provide inputs and check outputs
33     stimulus_process: process
34     begin
35         -- Test 1: Indirizzo A = 0
36         A_tb <= "0000";
37         wait for 10 ns;
38
39         -- Test 2: Indirizzo A = 1
40         A_tb <= "0001";
41         wait for 10 ns;
42
43         -- Test 3: Indirizzo A = 2
44         A_tb <= "0010";
45         wait for 10 ns;
46
47         -- Test 4: Indirizzo A = 3
48         A_tb <= "0011";
49         wait for 10 ns;
50
51         -- Test 5: Indirizzo A = 5
52         A_tb <= "0101";
53         wait for 10 ns;
54
55         -- Test 6: Indirizzo A = 7
56         A_tb <= "0111";
57         wait for 10 ns;
58
59         -- Test 7: Indirizzo A = 10
60         A_tb <= "1010";
61         wait for 10 ns;
62
63         -- Test 8: Indirizzo A = 255
64         A_tb <= "1111";
65         wait for 10 ns;
66
67         -- Fine simulazione
68         wait;

```

```

69      end process;
70
71 end behavior;
```

Code 2.4: Testbench

La seguente figura permette la visualizzazione delle waveform.

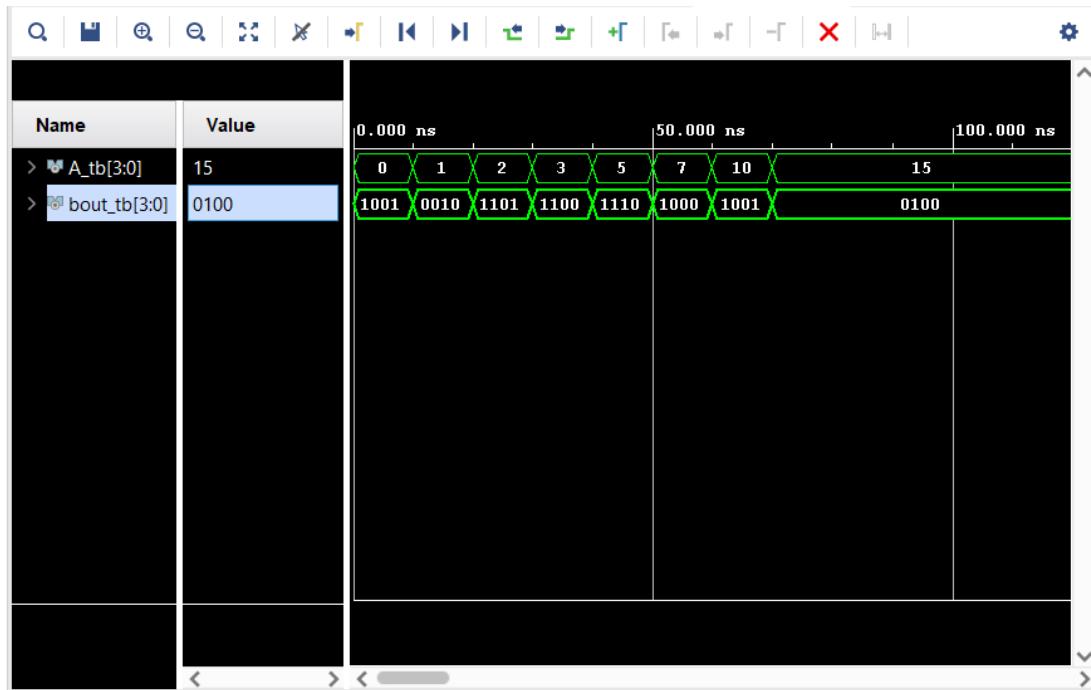


Figura 2.3: Waveform della simulazione di S

Si procede con dei test effettuati manualmente per mostrare la correttezza nel funzionamento del sistema S. Per consentire una maggiore leggibilità, si è scelto di visualizzare gli indirizzi come Unsigned Decimal.

Nel caso  $A = 0$ , si accede alla stringa 00001001, sommando i bit meno significativi con quelli più significativi si ottiene  $0000 + 1001 = 1001$ ; nel caso  $A = 5$ , si accede alla stringa 01001010, e procedendo come sopra si ottiene  $0100 + 1010 = 1110$ .

Come si può vedere, i risultati di questi test coincidono con il comportamento atteso dal sistema e che sono mostrati nella waveform relativa alla simulazione.

## 2.4 Implementazione su board

### 2.4.1 Traccia

Sintetizzare ed implementare su board il progetto del sistema ROM+M sviluppato al punto 2.1, utilizzando gli switch per fornire l'indirizzo della ROM da cui leggere i valori da trasformare e i led per visualizzare i 4 bit di uscita.

### 2.4.2 Implementazione

In questo caso, per implementare il sistema sulla board, è stato sufficiente modificare il file `Nexys-A7-50T-Master.xdc`, collegando i primi 4 switch (da 0 a 3) all'indirizzo *A* in ingresso, e i led da 0 a 3 alle uscite bout della macchina.

In particolare, il file xdc è composto dalle seguenti righe utili:

```
#GESTIONE SWITCH
set_property -dict {PACKAGE_PIN J15 IO_STANDARD LVCMOS33} [get_ports{ A[0]}];
# IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict {PACKAGE_PIN L16 IO_STANDARD LVCMOS33} [get_ports{ A[1]}];
# IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict {PACKAGE_PIN M13 IO_STANDARD LVCMOS33} [get_ports{ A[2]}];
# IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict {PACKAGE_PIN R15 IO_STANDARD LVCMOS33} [get_ports{ A[3]}];
# IO_L13N_T2_MRCC_14 Sch=sw[3]
```

## CAPITOLO 2. ESERCIZIO 2 - SISTEMA ROM+M

```
#GESTIONE LED  
set_property -dict {PACKAGE_PIN H17 IO_STANDARD LVCMOS33} [get_ports {bout[0]}];  
#IO_L18P_T2_A24_15 Sch=led[0]  
  
set_property -dict {PACKAGE_PIN K15 IO_STANDARD LVCMOS33} [get_ports {bout[1]}];  
#IO_L24P_T3_RS1_15 Sch=led[1]  
  
set_property -dict {PACKAGE_PIN J13 IO_STANDARD LVCMOS33} [get_ports {bout[2]}];  
#IO_L17N_T2_A25_15 Sch=led[2]  
  
set_property -dict {PACKAGE_PIN N14 IO_STANDARD LVCMOS33} [get_ports {bout[3]}];  
#IO_L8P_T1_D11_14 Sch=led[3]
```

Si mostrano in seguito alcuni test eseguiti sulla board, che hanno confermato i risultati ottenuti dalla simulazione.

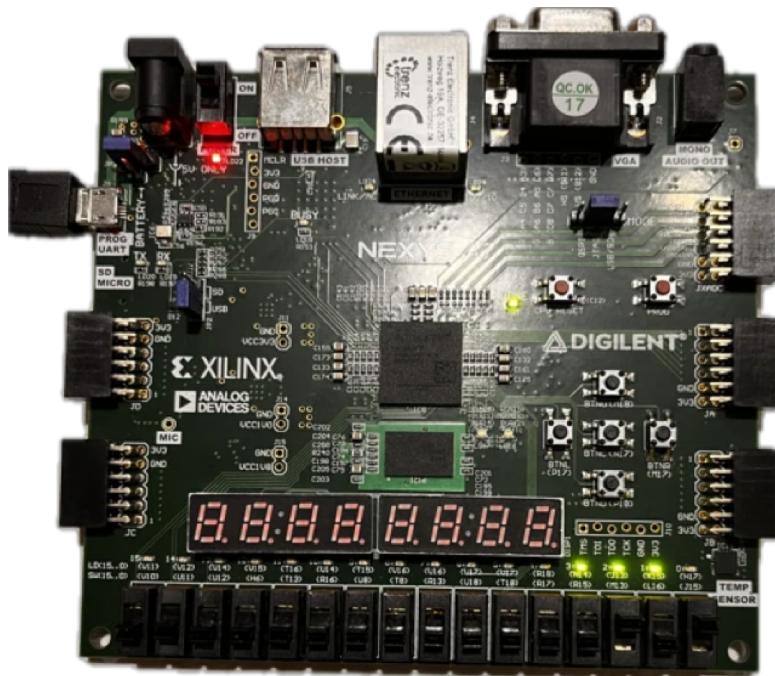


Figura 2.4: A = "0101", bout = "1110"

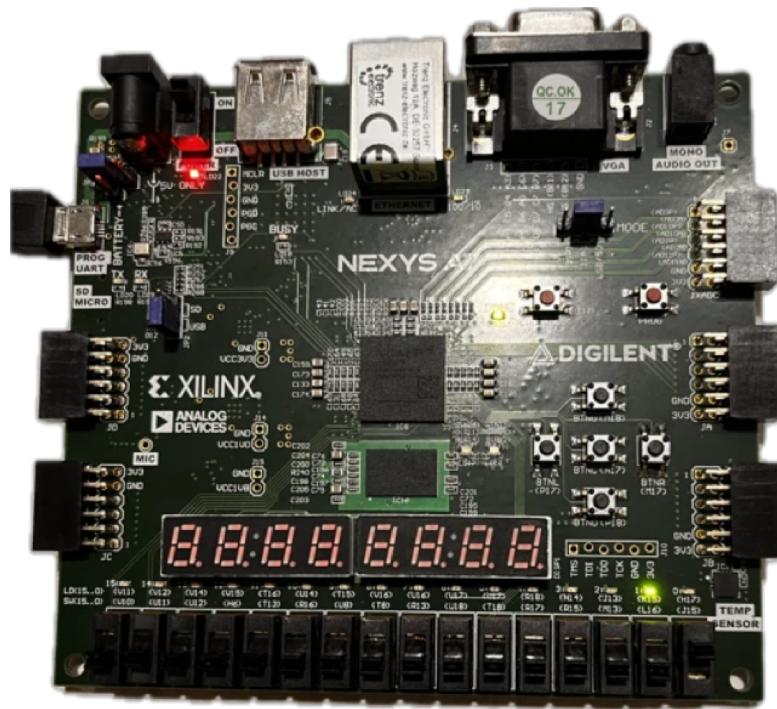


Figura 2.5:  $A = "0001"$ ,  $bout = "0010"$

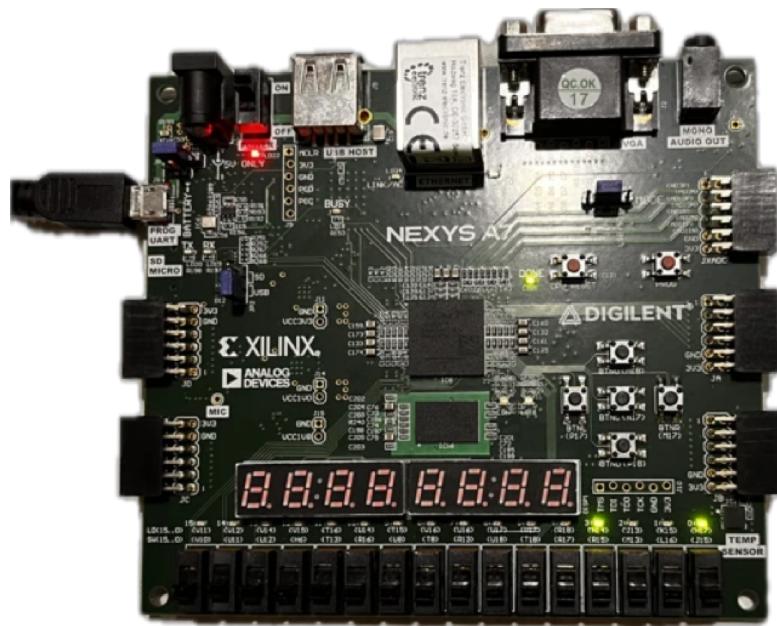


Figura 2.6:  $A = "1001"$ ,  $bout = "1001"$

Confrontando i risultati ottenuti con la waveform generata dalla simulazione si confermano le conclusioni precedenti.

# Capitolo 3

## Esercizio 3

### 3.1 Riconoscitore di sequenze

Un **riconoscitore di sequenze**, è una macchina sequenziale impulsiva<sup>1</sup> che riceve una sequenza di bit in ingresso e che, a seconda se tale sequenza sia uguale o non ad una data, ritorni i valori 1 e 0, rispettivamente.

In particolare si possono avere due tipi di riconoscitori:

1. **riconoscitori di sequenze non sovrapposte**: valuta i bit in ingresso a gruppi di  $n$  elementi alla volta;
2. **riconoscitori di sequenze parzialmente sovrapposte**: valuta i bit in ingresso a uno alla volta, tornando allo stato iniziale ogni qual volta la sequenza viene riconosciuta.

---

<sup>1</sup>Macchina in cui l'uscita è vera solo per un determinato stato e per un determinato ingresso, e poi torna ad essere falsa.

Nel caso in esame si vuole implementare un riconoscitore della sequenza **101**.

Oltre al dato, tale macchina ha in ingresso la tempificazione  $A$  e il valore  $M$ , che nel caso in cui  $M = 0$ , la macchina lavora come riconoscitore di sequenze non sovrapposte, mentre se  $M = 1$  lavora come riconoscitore di sequenze parzialmente sovrapposte.

### 3.1.1 Progettazione e architettura

Per progettare una macchina sequenziale, vi è bisogno dell'automa a stati finiti.

Nel caso in questione, vi è il seguente risultato

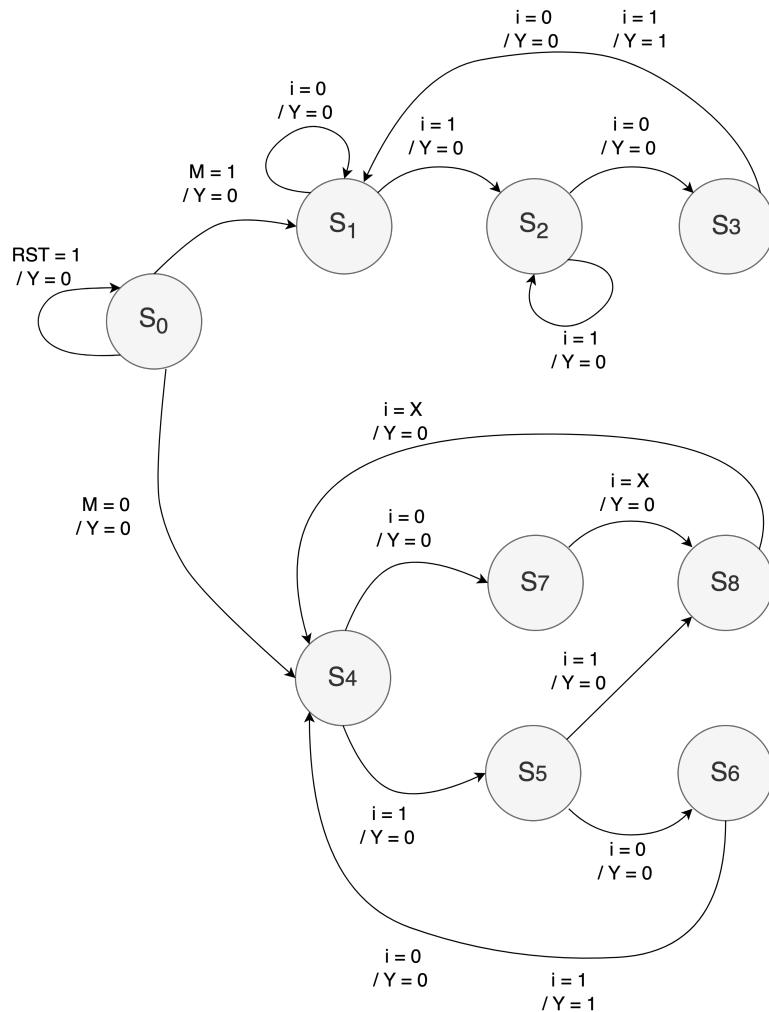


Figura 3.1: Automa riconoscitore di sequenza

### 3.1.2 Implementazione

Per l'implementazione VHDL dell'automa, si dichiarano dapprima gli ingressi

- RST: permette il reset della macchina, portandola allo stato  $S_0$ ;
- A: rappresenta l'abilitazione, ovvero il clock;
- $i$ : è l'ingresso;

- M: permette di selezionare con quale modalità far lavorare la macchina: se  $M = 0$  effettua il riconoscimento a gruppi di tre bit per volta; se  $M = 1$  effettua il riconoscimento un bit alla volta

L'uscita è rappresentata dal segnale Y.

L'architettura è costruita con un approccio comportamentale e vi è una variazione di stato ad ogni fronte di salita del clock (A).

Si vuole notare che il segnale RST è sincrono.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity riconoscitore is
5     port
6     (
7         RST:    in  std_logic;
8         i:      in  std_logic;
9         A:      in  std_logic;
10        M:      in  std_logic;
11        Y:      out std_logic
12    );
13 end riconoscitore;
14
15 architecture Behavioral of riconoscitore is
16     type state_type is (S0, S1, S2, S3, S4, S5 , S6, S7, S8);
17     signal current_state, next_state: state_type;
18     signal temp_y: std_logic;
19
20 begin
21     process(A, RST)
22     begin
23         if rising_edge(A) then          -- (A's event and A='1')
24             if RST = '1' then
25                 current_state <= S0;
26                 Y           <= '0';
27             else
28                 current_state <= next_state;
29                 Y           <= temp_y;
```

```
30         end if;
31     end if;
32 end process;
33
34 process (current_state, i, M)
35 begin
36     next_state          <= current_state;
37     --Y                 <= '0';
38
39     case current_state is
40     when S0 =>
41         if M = '1' then
42             next_state      <= S1;
43             temp_y          <= '0';
44         elsif M = '0' then
45             next_state      <= S4;
46             temp_y          <= '0';
47         end if;
48     when S1 =>
49         if i = '0' then
50             next_state      <= current_state;
51             temp_y          <= '0';
52         elsif i = '1' then
53             next_state      <= S2;
54             temp_y          <= '0';
55         end if;
56     when S2 =>
57         if i = '1' then
58             next_state      <= current_state;
59             temp_y          <= '0';
60         elsif i = '0' then
61             next_state      <= S3;
62             temp_y          <= '0';
63         end if;
64     when S3 =>
65         next_state          <= S1;
66         if i = '0' then
67             temp_y          <= '0';
68         elsif i = '1' then
69             temp_y          <= '1';
70         end if;
71
72     when S4 =>
73         if i = '0' then
74             next_state      <= S7;
75             temp_y          <= '0';
```

```

76      elsif i = '1' then
77          next_state      <=  S5;
78          temp_y         <=  '0';
79      end if;
80
81      when S5 =>
82          if i = '1' then
83              next_state      <=  S8;
84              temp_y         <=  '0';
85          elsif i = '0' then
86              next_state      <=  S6;
87              temp_y         <=  '0';
88          end if;
89      when S6 =>
90          next_state      <=  S4;
91          if i = '1' then
92              temp_y         <=  '1';
93          end if;
94      when S7 =>
95          next_state      <=  S8;
96          temp_y         <=  '0';
97      when S8 =>
98          next_state      <=  S4;
99          temp_y         <=  '0';
100     end case;
101 end process;
102 end Behavioral;
```

Code 3.1: riconoscitore.vhdl

### 3.1.3 Simulazione

Per effettuare la simulazione, è stato necessario il seguente testbench.

```

1 -- Testbench for riconoscitore (sequence 101 detection)
2 library IEEE;
3 use IEEE.Std_logic_1164.all;
4 use IEEE.Numeric_Std.all;
5
6 entity riconoscitore_tb is
```

```
7 end;
8
9 architecture bench of riconoscitore_tb is
10
11     component riconoscitore
12         port
13         (
14             RST:    in  std_logic;
15             i:      in  std_logic;   -- Input signal
16             A:      in  std_logic;   -- Clock signal
17             M:      in  std_logic;   -- Mode or another input (adjust as
18                 ← needed)
19             Y:      out std_logic   -- Output signal (detects "101")
20         );
21     end component;
22
23     signal RST: std_logic := '0';
24     signal i: std_logic := '0';
25     signal A: std_logic := '0';   -- Clock
26     signal M: std_logic := '0';
27     signal Y: std_logic;
28
29     constant clock_period: time := 10 ns;
30     signal stop_the_clock: boolean := false;
31
32 begin
33
34     uut: riconoscitore port map (
35         RST => RST,
36         i    => i,
37         A    => A,
38         M    => M,
39         Y    => Y
40     );
41
42     -- Clock generation
43     clocking: process
44     begin
45         while not stop_the_clock loop
46             A <= '0';
47             wait for clock_period/2;
48             A <= '1';
49             wait for clock_period/2;
50         end loop;
51         wait;
52     end process;
```

```
52
53    -- Stimulus process
54  stimulus: process
55  begin
56      -- Initialization
57      RST <= '1';
58      wait for 2 * clock_period; -- Hold reset for 2 clock cycles
59      RST <= '0';
60      wait for clock_period;
61
62      M <= '1';
63      wait for 2 * clock_period;
64      i <= '1';
65      wait for clock_period;
66      i <= '1';
67      wait for clock_period;
68      i <= '0';
69      wait for clock_period;
70      i <= '1';
71      wait for clock_period;
72      i <= '0';
73      wait for clock_period;
74      i <= '0';
75      wait for clock_period;
76      i <= '1';
77      wait for clock_period;
78      i <= '0';
79      wait for clock_period;
80      i <= '1';
81      wait for clock_period;
82
83      RST <= '1';
84      wait for 2 * clock_period;
85      RST <= '0';
86      wait for clock_period;
87
88      M <= '0';
89      wait for 2 * clock_period;
90      i <= '1';
91      wait for clock_period;
92      i <= '1';
93      wait for clock_period;
94      i <= '0';
95      wait for clock_period;
96      i <= '1';
97      wait for clock_period;
```

```

98      i <= '0';
99      wait for clock_period;
100     i <= '1';
101     wait for clock_period;
102     i <= '1';
103     wait for clock_period;
104     i <= '0';
105     wait for clock_period;
106     i <= '1';
107     wait for clock_period;
108
109
110     -- End simulation
111     stop_the_clock <= true;
112     wait;
113   end process;
114
115 end bench;

```

Code 3.2: riconoscitore\_tb.vhdl

Gli ingressi sono i seguenti:

- $M = 1$ :

– 1, 1, 0, 1, 0, 0, 1, 0, 1

- $M = 0$ :

– 1, 1, 0, 1, 0, 1, 1, 0, 1

Il risultato è il seguente:

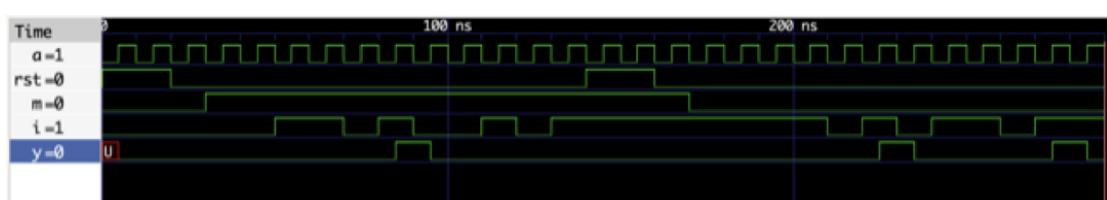


Figura 3.2: Simulazione Riconosciore

## 3.2 Implementazione su board del punto precedente

### 3.2.1 Traccia

Sintetizzare e implementare su board la rete sviluppata al punto precedente, utilizzando uno switch S1 per codificare l'input i e uno switch S2 per codificare il modo M, in combinazione con due bottoni B1 e B2 utilizzati rispettivamente per acquisire l'input da S1 e S2 in sincronismo con il segnale di temporizzazione A, che deve essere ottenuto a partire dal clock della board. Infine, l'uscita Y può essere codificata utilizzando un led.

### 3.2.2 Implementazione

Il Riconoscitore di sequenza viene ripreso dal punto precedente, di conseguenza il suo codice viene importato nel progetto senza variazioni. Per gestire il funzionamento di tale sistema su board, prima di tutto si rende necessario l'utilizzo di un divisore di frequenze che ha la funzione di generare un segnale di clock con una frequenza più bassa rispetto al clock di ingresso. In particolare il processo implementato genera un clock di uscita con una frequenza pari a quella di ingresso divisa per il valore di *DIVISOR*. Si mostra il codice:

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
```

```

3  use IEEE.NUMERIC_STD.ALL;
4
5  entity frequency_divider is
6      Port (
7          clock_in : in STD_LOGIC;    -- Clock di ingresso
8          reset : in STD_LOGIC;      -- Segnale di reset
9          clock_out : out STD_LOGIC -- Clock di uscita, divisore di
10             → frequenza
11      );
12 end frequency_divider;
13
14
15 architecture Behavioral of frequency_divider is
16
17     -- Definire il valore massimo del contatore in base alla
18     -- divisione desiderata
19     -- clock di ingresso di 100 MHz, si vuole ottenere una frequenza
20     -- di uscita di 1 Hz:
21     constant CLOCK_FREQ : integer := 100_000_000; -- Frequenza del
22             → clock di ingresso (100 MHz)
23     constant DIVISOR : integer := 100_000_000;      -- Divisione
24             → desiderata (1 Hz: 100 MHz / 100_000_000)
25     constant COUNT_MAX : integer := DIVISOR / 2 - 1; -- Calcola il
26             → massimo valore del contatore (per ottenere un periodo
27             → completo)
28
29     signal counter : integer range 0 to COUNT_MAX := 0; -- Contatore
30             → per dividere la frequenza
31     signal clock_signal : STD_LOGIC := '0'; -- Segnale di clock di
32             → uscita
33
34 begin
35
36     -- Processo che divide la frequenza
37     process(clock_in)
38     begin
39         if rising_edge(clock_in) then
40             if reset = '1' then
41                 counter <= 0; -- Reset del contatore
42                 clock_signal <= '0'; -- Reset del segnale di clock di
43                     → uscita
44             else
45                 if counter = COUNT_MAX then
46                     counter <= 0; -- Reset del contatore al
47                         → raggiungimento del massimo
48                     clock_signal <= not clock_signal; -- Si inverte
49                         → il segnale di clock di uscita
50             end if;
51         end if;
52     end process;
53 end;

```

```
37         else
38             counter <= counter + 1; -- Incrementa il
39             --> contatore
40         end if;
41     end if;
42 end process;
43
44 -- Collega il segnale di uscita al segnale di clock
45 clock_out <= clock_signal;
46
47 end Behavioral;
```

---

Code 3.3: frequency\_divider.vhdl

Per la gestione degli input tramite i bottoni della board, è stata realizzata una unità di controllo; si usa lo *switch[0]* unito al bottone *BTNL* per l'ingresso "i", e lo *switch[1]* unito al bottone *BTNR* per l'ingresso "M". Si è scelto inoltre di mostrare le variazioni del clock sul *led[0]*, in modo da poter inserire correttamente gli input in corrispondenza del fronte di salita. L'uscita viene invece visualizzata sul *led[1]*.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity control_unit is
5   Port (
6     A, reset: in STD_LOGIC;
7     load_in, load_sel: in STD_LOGIC;
8     value_in: in STD_LOGIC;
9     sel_in: in STD_LOGIC;
10    value_out: out STD_LOGIC;
11    sel_out: out STD_LOGIC
12  );
13 end control_unit;
14
15 architecture Behavioral of control_unit is
16   signal reg_value: STD_LOGIC;
```

```
17 signal reg_sel: STD_LOGIC;
18
19 begin
20 value_out <= reg_value;
21 sel_out <= reg_sel;
22
23 main: process (A)
24 begin
25 if (A'event AND A = '1') then
26 if (reset = '1') then
27 reg_value <= '0';
28 reg_sel <= '0';
29 elsif (load_sel = '1') then
30 reg_sel <= sel_in;
31 elsif (load_in = '1') then
32 reg_value <= value_in;
33 end if;
34 end if;
35 end process;
36 end Behavioral;
```

Code 3.4: control\_unit.vhdl

Il codice del sistema su board, nel suo complesso, è stato realizzato seguendo un approccio di tipo strutturale:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity seqRecOnBoard is
5 Port (
6     A, reset: in STD_LOGIC;
7     load_i, load_M: in STD_LOGIC;
8     i, M: in STD_LOGIC;
9     Y: out STD_LOGIC;
10    led_out: out STD_LOGIC
11
12 );
13 end seqRecOnBoard;
14
15 architecture structural of seqRecOnBoard is
16 component control_unit
17 Port (
18     A, reset: in STD_LOGIC;
```

```

19      load_in, load_sel: in STD_LOGIC;
20      value_in: in STD_LOGIC;
21      sel_in: in STD_LOGIC;
22      value_out: out STD_LOGIC;
23      sel_out: out STD_LOGIC
24  );
25 end component;
26
27 component riconoscitore is
28     port
29     (
30         RST:    in std_logic;
31         i:      in std_logic;
32         A:      in std_logic;
33         M:      in std_logic;
34         Y:      out std_logic
35     );
36 end component;
37
38 component frequency_divider is
39     Port (
40         clock_in : in STD_LOGIC; -- Clock di ingresso
41         reset : in STD_LOGIC; -- Segnale di reset
42         clock_out : out STD_LOGIC -- Clock di uscita, divisore di
43             -- frequenza
44     );
45     end component;
46
47 signal M_cu: STD_LOGIC;
48 signal i_cu: STD_LOGIC;
49 signal clock_temp: STD_LOGIC;
50
51 begin
52     CU: control_unit
53     port map(
54         A => A,
55         reset => reset,
56         load_in => load_i,
57         load_sel => load_M,
58         value_in => i,
59         sel_in => M,
60         value_out => i_cu,
61         sel_out => M_cu
62     );
63     SR: riconoscitore

```

```
64  port map(
65      RST => reset,
66      i => i_cu,
67      A => clock_temp,
68      M => M_cu,
69      Y => Y
70  );
71
72 fd: frequency_divider
73  port map(
74      clock_in => A,
75      reset => reset,
76      clock_out => clock_temp
77  );
78 led_out <= clock_temp;
79 end structural;
```

---

Code 3.5: Riconoscitore su board in vhdl

# Capitolo 4

## Esercizio 4

### 4.1 Shift Register - Approccio comportamentale

Si vuole implementare uno Shift Register con approccio comportamentale, la cui dimensione è  $N$ .

Tale macchina ha come ingresso un valore  $y$  con il quale si può scegliere di fare shift verso destra o sinistra e di fare shift di uno o due bit.

#### 4.1.1 Progetto e architettura

La macchina da implementare è la seguente:

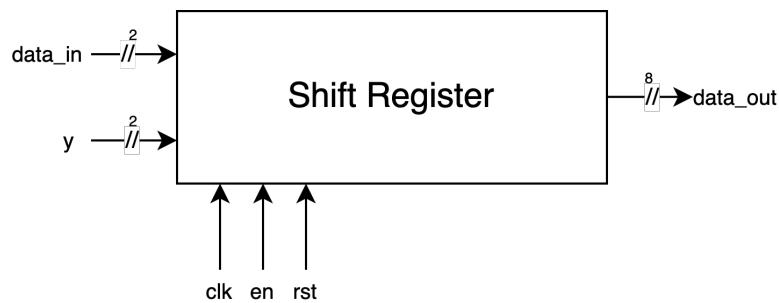


Figura 4.1: Shift Register

Dall'immagine si può notare che è stato scelto  $N = 8$ .

Gli ingressi sono i seguenti:

- `clk`: il clock, necessario per la temporizzazione. La macchina lavorerà sul fronte di salita;
- `en`: l'abilitazione, la quale permette di abilitare o disabilitare la macchina;
- `rst`: reset sincrono della macchina;
- `y`: vettore di 2 elementi che sceglie la modalità di funzionamento della macchina; in particolare:
  - $y = 00$ : shift a sinistra di 1;
  - $y = 01$ : shift a sinistra di 2;
  - $y = 10$ : shift a destra di 1;
  - $y = 11$ : shift a destra di 2;

- `data_in`: rappresenta i dati in ingresso; esso è un vettore di due elementi poiché quando vi è la necessità di fare uno shift di 2, si ha bisogno di due bit

L'uscita della macchina è `data_out`, vettore di 8 bit.

### 4.1.2 Implementazione

L'implementazione è la seguente

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity shift_register_beh is
6     generic ( N: integer := 8 );      --lunghezza registro
7
8     port
9     (
10         clk      : in  std_logic;
11         rst      : in  std_logic;
12         en       : in  std_logic;
13         y        : in  std_logic_vector(1 downto 0);
14         data_in : in  std_logic_vector(1 downto 0);
15         data_out: out std_logic_vector(N-1 downto 0)
16     );
17 end entity;
18
19 architecture behavioral of shift_register_beh is
20     signal reg  : std_logic_vector(N-1 downto 0);
21
22 begin
23     process(clk)
24     begin
25         if (rising_edge(clk)) then
26             if (rst = '1') then
27                 reg <= (others => '0');
28             end if;
29
30             if (en = '1') then
```

```

31      case y is
32        when "00" =>          --shift a sinistra di 1
33          reg(7 downto 1) <= reg(6 downto 0);
34          reg(0)           <= data_in(0);
35        when "01" =>          --shift a sinistra di 2
36          reg(7 downto 2) <= reg(5 downto 0);
37          reg(1)           <= data_in(1);
38          reg(0)           <= data_in(0);
39        when "10" =>          --shift a destra di 1
40          reg(6 downto 0) <= reg(7 downto 1);
41          reg(7)           <= data_in(0);
42        when "11" =>          --shift a destra di 2
43          reg(5 downto 0) <= reg(6 downto 1);
44          reg(6)           <= data_in(0);
45          reg(7)           <= data_in(1);
46        when others =>
47          null;
48      end case;
49    end if;
50  end if;
51 end process;
52
53 data_out <= reg;
54 end behavioral;

```

Code 4.1: shift\_register\_beh.vhdl

### 4.1.3 Simulazione

Per effettuare la simulazione, si utilizza il seguente testbench:

```

1 library IEEE;
2 use IEEE.Std_logic_1164.all;
3 use IEEE.Numeric_Std.all;
4
5 entity shift_register_beh_tb is
6 end;
7
8 architecture bench of shift_register_beh_tb is
9
10 component shift_register_beh
11   generic ( N: integer := 8 );

```

```

12      port
13      (
14          clk      : in  std_logic;
15          rst      : in  std_logic;
16          en       : in  std_logic;
17          Y        : in  std_logic_vector(1 downto 0);
18          data_in : in  std_logic_vector(1 downto 0);
19          data_out: out std_logic_vector(N-1 downto 0)
20      );
21  end component;
22
23  -- Signal declarations
24  signal clk      : std_logic := '0';
25  signal rst      : std_logic := '0';
26  signal en       : std_logic := '0';
27  signal Y        : std_logic_vector(1 downto 0) := "00";
28  signal data_in : std_logic_vector(1 downto 0) := "00";
29  signal data_out: std_logic_vector(7 downto 0);
30
31  constant clock_period: time := 10 ns;
32  signal stop_the_clock: boolean := false;
33
34 begin
35
36  -- Instanziazione del componente sotto test
37  uut: shift_register_beh
38  generic map ( N => 8 )  -- Dimensione del registro
39  port map (
40      clk      => clk,
41      rst      => rst,
42      en       => en,
43      Y        => Y,
44      data_in  => data_in,
45      data_out => data_out
46  );
47
48  -- Stimoli
49  stimulus: process
50  begin
51      -- Reset iniziale
52      rst <= '1';
53      wait for clock_period;
54      rst <= '0';
55      wait for clock_period;
56
57      -- Abilitazione e shift a sinistra di 1

```

```
58      en <= '1';
59      y <= "00";
60      data_in <= "01"; -- Input dati
61      wait for clock_period;
62
63      -- Shift a sinistra di 2
64      y <= "01";
65      data_in <= "10";
66      wait for clock_period;
67
68      -- Shift a destra di 1
69      y <= "10";
70      data_in <= "11";
71      wait for clock_period;
72
73      -- Shift a destra di 2
74      y <= "11";
75      data_in <= "01";
76      wait for clock_period;
77
78      -- Disabilitazione
79      en <= '0';
80      data_in <= "00";
81      wait for clock_period;
82
83      -- Concludi la simulazione
84      stop_the_clock <= true;
85      wait;
86  end process;
87
88  -- Processo di generazione del clock
89  clocking: process
90  begin
91      while not stop_the_clock loop
92          clk <= '0', '1' after clock_period / 2;
93          wait for clock_period;
94      end loop;
95      wait;
96  end process;
97
98 end;
```

Code 4.2: shift\_register\_beh\_tb.vhdl

Il risultato della simulazione è il seguente:

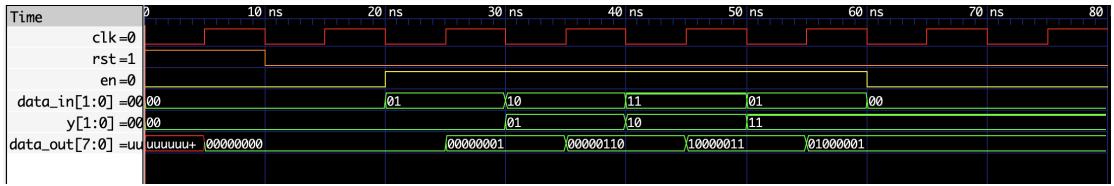


Figura 4.2: Simulazione Shift Register con approccio comportamentale

Si può facilmente notare dall'immagine che la macchina lavora come desiderato: ad ogni fronte di salita del clock e quando l'abilitazione è alta, in base alla modalità di lavoro, shifta a destra o a sinistra, di uno o due bit.

## 4.2 Shift Register - Approccio strutturale

Si vuole riprogettare la macchina precedente, figura 4.1, utilizzando un approccio strutturale.

Le componenti della macchina sono 8 registri da 1 bit e 8 mux 4:1.

Si sono scelti 8 registri poiché in tale esempio si realizza un registro da 8 bit ( $N = 8$ ).

### 4.2.1 Progetto e architettura

#### Registro da un bit

Il primo componente necessario è il registro da un bit.

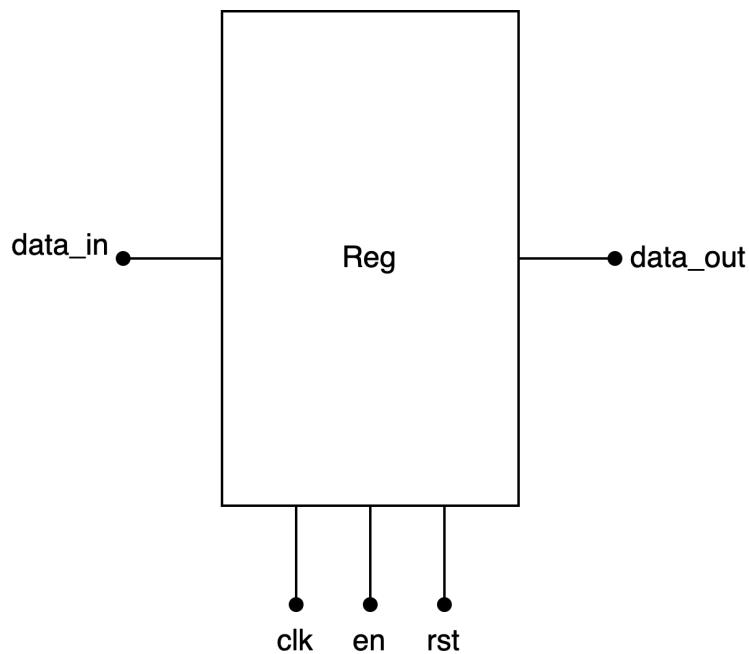


Figura 4.3: Registro da 1 bit

Gli ingressi di tale componente sono i seguenti:

- **data\_in**: bit in ingresso, che verrà memorizzato nel registro;
- **clk**: il clock per la temporizzazione; il registro lavora sul fronte di salita di quest'ultimo;
- **en**: segnale di abilitazione; il registro memorizza il bit in ingresso solo quando tale segnale è alto;
- **rst**: segnale che quando è alto resetta il registro, portando il valore al suo interno a 0; il reset è sincrono.

La sua uscita è **data\_out**, che altro non rappresenta il bit memorizzato nel registro.

**Mux 4:1**

Il secondo componente è il mux 4:1.

Tramite quest'ultimo si decide qual è l'ingresso di un registro, attraverso la selezione.

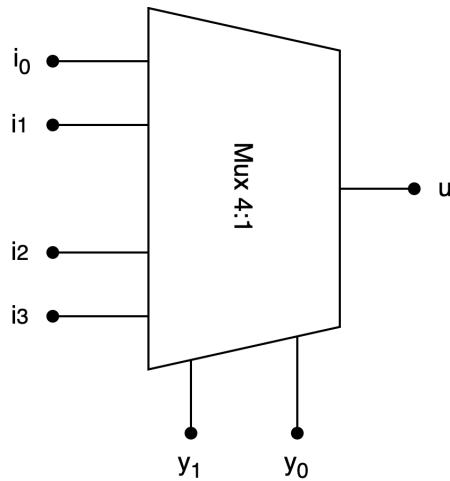


Figura 4.4: Mux 4:1

Tale multiplexer lavora seguendo la seguente tabella: Si può banalmente

<b>y<sub>1</sub></b>	<b>y<sub>0</sub></b>	<b>u</b>
0	0	<b>i<sub>0</sub></b>
0	1	<b>i<sub>1</sub></b>
1	0	<b>i<sub>2</sub></b>
1	1	<b>i<sub>3</sub></b>

Tabella 4.1: Tabella di verità del multiplexer 4:1 per lo Shift Register

te notare che l'uscita altro non è che uno dei 4 ingressi del multiplexer, scelto tramite la selezione.

Si può ora comporre lo Shift Register.

Facendo gli opportuni collegamenti, si ottiene il seguente schema

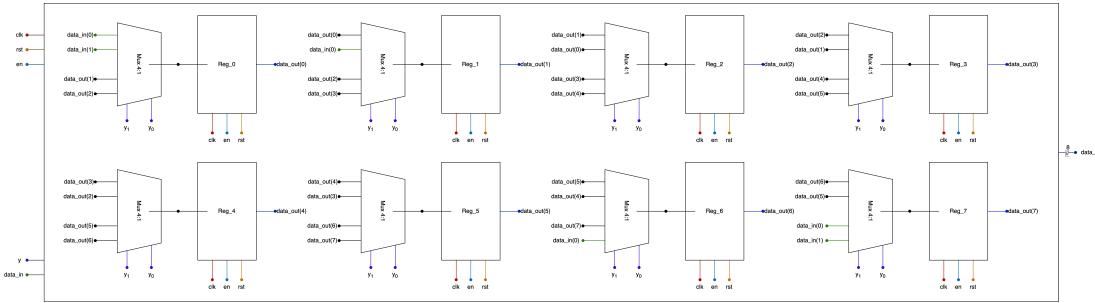


Figura 4.5: Shift Register con approccio strutturale

Gli ingressi e l'uscita dello Shift Register sono identici a quelli visti nell'esercizio precedente.

Quello che si vuole mettere in evidenza in questo caso è come tali ingressi siano collegati con le strutture interni: in particolare si vede che  $y$ , che sceglie la modalità di lavoro della macchina, è collegato alla selezione dei multiplexer, mentre  $\text{data\_in}$ , è collegato solo ai primi due e agli ultimi due multiplexer.

I restanti sono collegati direttamente ai registri.

#### 4.2.2 Implementazione

Si vuole ora procedere con l'implementazione in VHDL.

Partendo dal registro, si ha:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity reg is
6     port
7     (
8         data_in:      in  std_logic;
9         en:          in  std_logic;
10        clk:         in  std_logic;

```

```
11      rst:      in  std_logic;
12      data_out:   out std_logic
13  );
14 end entity;
15
16 architecture Behavioral of reg is
17 begin
18     process(clk)
19     begin
20         if(rising_edge(clk))then
21             if (rst = '1') then
22                 data_out <= '0';
23             end if;
24
25             if (en = '1') then
26                 data_out <= data_in;
27             end if;
28         end if;
29     end process;
30 end architecture;
```

Code 4.3: register.vhdl

Si prosegue con il multiplexer 4:1

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity mux_41 is
5     port
6     (
7         i: in std_logic_vector(0 to 3);
8         y: in std_logic_vector(1 downto 0);
9         output: out std_logic
10    );
11 end entity;
12
13 architecture Behavioral of mux_41 is
14 begin
15     process(y, i)
16     begin
17         case y is
18             when "00" =>
19                 output <= i(0);
20             when "01" =>
```

```

21          output <= i(1);
22      when "10" =>
23          output <= i(2);
24      when "11" =>
25          output <= i(3);
26      when others =>
27          output <= '---';
28  end case;
29 end process;
30 end architecture;

```

Code 4.4: mux\_4\_1.vhdl

Implementate le componenti base per il progetto, si prosegue con lo Shift Register:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity shift_register is
5 port
6 (
7     clk:      in  std_logic;
8     rst:      in  std_logic;
9     en:       in  std_logic;
10    y:        in  std_logic_vector(1 downto 0);
11    data_in:   in  std_logic_vector(1 downto 0);
12    data_out:  out std_logic_vector(7 downto 0)
13 );
14 end entity;
15
16 architecture structural of shift_register is
17 signal in_mux:      std_logic_vector(0 to 31);
18 signal out_mux:     std_logic_vector(0 to 7);
19 signal temp_out:    std_logic_vector(0 to 7);
20
21 component reg is
22 port
23 (
24     data_in:   in  std_logic;
25     en:       in  std_logic;
26     clk:      in  std_logic;
27     rst:      in  std_logic;
28     data_out:  out std_logic

```

```

29      );
30  end component;
31
32 component mux_41 is
33  port
34  (
35    i:      in std_logic_vector(0 to 3);
36    y:      in std_logic_vector(1 downto 0);
37    output: out std_logic
38  );
39 end component;
40
41 begin
42   data_out    <=  temp_out;
43
44   mux0to7: for k in 0 to 7 generate
45     m: mux_41
46     port map
47     (
48       in_mux(4*k to (k*4 + 3)),
49       y,
50       out_mux(k)
51     );
52   end generate;
53
54   reg0_to7: for k in 0 to 7 generate
55     r: reg
56     port map
57     (
58       out_mux(k),
59       en,
60       clk,
61       rst,
62       temp_out(k)
63     );
64   end generate;
65
66   --assignment input signals for mux
67   --mux_0
68   in_mux(0)    <=  data_in(0);
69   in_mux(1)    <=  data_in(1);
70   in_mux(2)    <=  temp_out(1);
71   in_mux(3)    <=  temp_out(2);
72
73   --mux_1
74   in_mux(4)    <=  temp_out(0);

```

```
75      in_mux(5)    <=  data_in(0);
76      in_mux(6)    <=  temp_out(2);
77      in_mux(7)    <=  temp_out(3);
78
79      --mux_2 to mux_5
80
81      conn_mux2to5: for k in 2 to 5 generate
82          in_mux(4*k)      <=  temp_out(k-1);
83          in_mux(4*k + 1)  <=  temp_out(k-2);
84          in_mux(4*k + 2)  <=  temp_out(k+1);
85          in_mux(4*k + 3)  <=  temp_out(k+2);
86      end generate;
87
88      --mux_6
89      in_mux(24)    <=  temp_out(5);
90      in_mux(25)    <=  temp_out(4);
91      in_mux(26)    <=  temp_out(7);
92      in_mux(27)    <=  data_in(0);
93
94      --mux_7
95      in_mux(28)    <=  temp_out(6);
96      in_mux(29)    <=  temp_out(5);
97      in_mux(30)    <=  data_in(0);
98      in_mux(31)    <=  data_in(1);
99
100 end structural;
```

---

Code 4.5: shift\_register.vhdl

Si può notare come nella architettura, sono state prima generate le componenti e solo dopo sono stati effettuati i vari collegamenti, utilizzando variabili ausiliarie.

### 4.2.3 Simulazione

Per effettuare la simulazione, si implementa il seguente testbench

```
1  library IEEE;
2  use IEEE.Std_logic_1164.all;
3  use IEEE.Numeric_Signed.all;
```

```
4
5 entity shift_register_tb is
6 end;
7
8 architecture bench of shift_register_tb is
9
10 component shift_register
11 port
12 (
13     clk:      in std_logic;
14     rst:      in std_logic;
15     en:       in std_logic;
16     y:        in std_logic_vector(1 downto 0);
17     data_in:   in std_logic_vector(1 downto 0);
18     data_out:  out std_logic_vector(7 downto 0)
19 );
20 end component;
21
22 signal clk: std_logic := '0';
23 signal rst: std_logic := '0';
24 signal en: std_logic := '0';
25 signal y: std_logic_vector(1 downto 0) := "00";
26 signal data_in: std_logic_vector(1 downto 0) := "00";
27 signal data_out: std_logic_vector(7 downto 0);
28
29 begin
30
31     uut: shift_register port map ( clk      => clk,
32                                     rst      => rst,
33                                     en       => en,
34                                     y        => y,
35                                     data_in  => data_in,
36                                     data_out => data_out );
37
38     clk_process: process
39 begin
40     -- Clock generation
41     clk <= not clk after 10 ns;
42     wait for 10 ns;
43 end process;
44
45     stimulus: process
46 begin
47     -- Test Case 1: Apply reset
48     rst <= '1';           -- Assert reset
49     wait for 20 ns;       -- Wait for reset to propagate
```

```

50      rst <= '0';           -- Deassert reset
51      wait for 20 ns;
52
53      -- Test Case 2: Enable shift register with y = "00"
54      en <= '1';           -- Enable shift register
55      y <= "00";           -- Set y to "00"
56      data_in <= "01";     -- Apply data "01"
57      wait for 40 ns;
58
59      -- Test Case 3: Apply data_in while changing y to "01"
60      y <= "01";           -- Change y to "01"
61      data_in <= "10";     -- Apply data "10"
62      wait for 40 ns;
63
64      -- Test Case 4: Apply data_in while changing y to "10"
65      y <= "10";           -- Change y to "10"
66      data_in <= "11";     -- Apply data "11"
67      wait for 40 ns;
68
69      -- Test Case 5: Disable shift register with y = "11"
70      en <= '0';           -- Disable shift register
71      y <= "11";           -- Set y to "11"
72      data_in <= "00";     -- Change data input
73      wait for 40 ns;
74
75      -- Test Case 6: Apply data_in with y = "00" and reset active
76      rst <= '1';           -- Assert reset
77      y <= "00";           -- Set y to "00"
78      data_in <= "10";     -- Apply data "10" while reset is active
79      wait for 20 ns;
80      rst <= '0';           -- Deassert reset
81      wait for 40 ns;
82
83      -- Test Case 7: Apply data_in while y = "01" and enable shifting
84      y <= "01";           -- Set y to "01"
85      en <= '1';           -- Enable shift register
86      data_in <= "11";     -- Apply data "11"
87      wait for 40 ns;
88      data_in <= "00";     -- Change data to "00"
89      wait for 40 ns;
90
91      -- Test Case 8: Apply reset while shifting and changing y
92      rst <= '1';           -- Assert reset
93      y <= "10";           -- Change y to "10"
94      data_in <= "11";     -- Apply data "11"
95      wait for 20 ns;

```

```

96      rst <= '0';           -- Deassert reset
97      wait for 40 ns;
98
99      -- Test Case 9: Apply y = "11" and shift continuously with
100     --> data_in changes
101    y <= "11";           -- Set y to "11"
102    en <= '1';           -- Enable shift register
103    data_in <= "00";     -- Apply data "00"
104    wait for 40 ns;
105    data_in <= "01";     -- Apply data "01"
106    wait for 40 ns;
107    data_in <= "10";     -- Apply data "10"
108    wait for 40 ns;
109    data_in <= "11";     -- Apply data "11"
110    wait for 40 ns;
111
112    -- Test Case 10: Reset during shifting with y = "00"
113    rst <= '1';           -- Assert reset during shift
114    y <= "00";           -- Set y to "00"
115    data_in <= "01";     -- Apply data "01"
116    wait for 20 ns;
117    rst <= '0';           -- Deassert reset
118    wait for 40 ns;
119
120    -- Test Case 11: Changing y while shifting with enabled register
121    y <= "01";           -- Set y to "01"
122    en <= '1';           -- Enable shift register
123    data_in <= "01";     -- Apply data "01"
124    wait for 40 ns;
125    y <= "10";           -- Change y to "10"
126    data_in <= "11";     -- Change data to "11"
127    wait for 40 ns;
128
129    -- Test Case 12: Apply y = "11" and shift with reset active
130    rst <= '1';           -- Assert reset
131    y <= "11";           -- Set y to "11"
132    data_in <= "10";     -- Apply data "10"
133    wait for 20 ns;
134    rst <= '0';           -- Deassert reset
135    wait for 40 ns;
136
137    -- Test Case 13: Disable shifting during changes in y
138    en <= '0';           -- Disable shift register
139    y <= "00";           -- Set y to "00"
140    data_in <= "11";     -- Change data to "11"
141    wait for 40 ns;

```

```

141      y <= "01";           -- Change y to "01"
142      data_in <= "00";     -- Change data to "00"
143      wait for 40 ns;
144
145      -- End
146  end process;
147
148 end;

```

Code 4.6: tb\_shift\_register.vhdl

Lanciando la simulazione, il risultato è il seguente:

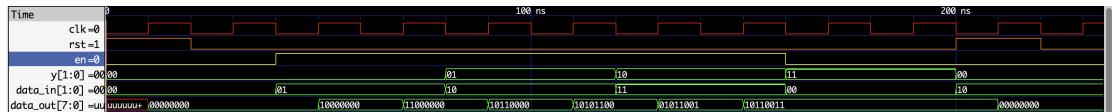


Figura 4.6: Simulazione dello Shift Register con approccio strutturale

Si nota chiaramente un corretto funzionamento della macchina.

# Capitolo 5

## Esercizio 5

### 5.1 Cronometro

Si vuole progettare, implementare e testare un cronometro, in grado di scandire secondi, minuti e ore, a partire da una base dei tempi prefissata (clock).

Si vuole inoltre che l'inizializzazione del cronometro possa essere fatta anche con un valore iniziale, espresso in ore, minuti e secondi, mediante un ingresso di `set`, e deve prevedere un ingresso di `reset` per azzerare il tempo.

#### 5.1.1 Progettazione

Per la progettazione di tale macchina, si utilizza un approccio strutturale.

In linea generale, si parte da un flip-flop D per la composizione di con-

tatori di modulo 64 e modulo 32. Questi ultimi saranno necessari per la progettazione di due contatori modulo 60 e un contatore modulo 24, rispettivamente.

Questi ultimi tre, collegati opportunamente, andranno a comporre il cronometro.

### Flip-Flop D

Il Flip-Flop D è progettato come segue:

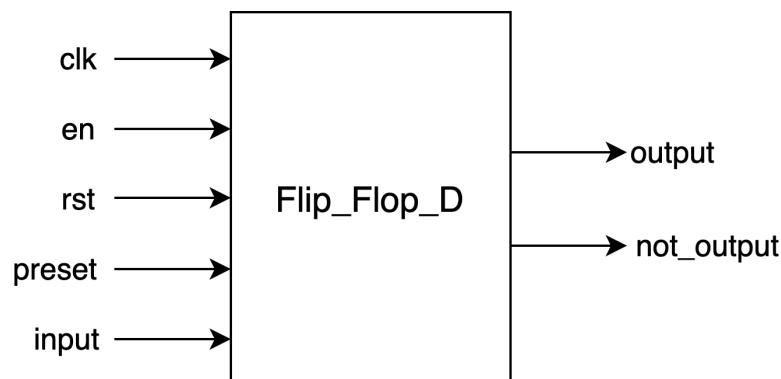


Figura 5.1: Flip-Flop D

La macchina lavora sul fronte di salita del clock e quando il valore en è alto.

Ha due uscite, la prima output presenta in uscita ciò che è memorizzato nel Flip-Flop, mentre not\_output, presenta il negato.

### Contatore modulo 64

Il contatore modulo 64 che si vuole progettare è il seguente:

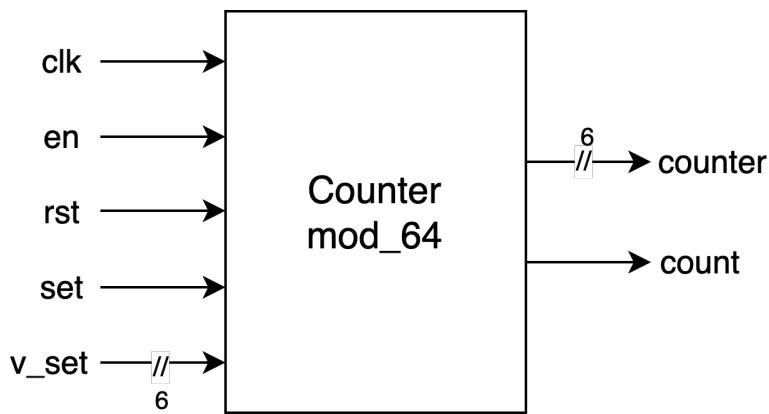


Figura 5.2: Contatore modulo 64

Gli ingessi sono:

- `clk`: clock per la temporizzazione;
- `en`: segnale di abilitazione;
- `rst`: segnale di reset;
- `set` e `v_set`: segnali necessari per il setting del valore di partenza

Le uscite sono `counter`, che altro non è il conteggio, e `count`, uscita necessaria per permettere il collegamento con altri contatori.

Si vuole progettare tale macchina strutturalmente, utilizzando come componente base il Flip-Flop D.

Per far ciò si collegano parallelamente cinque Flip-Flop D, nel modo seguente:

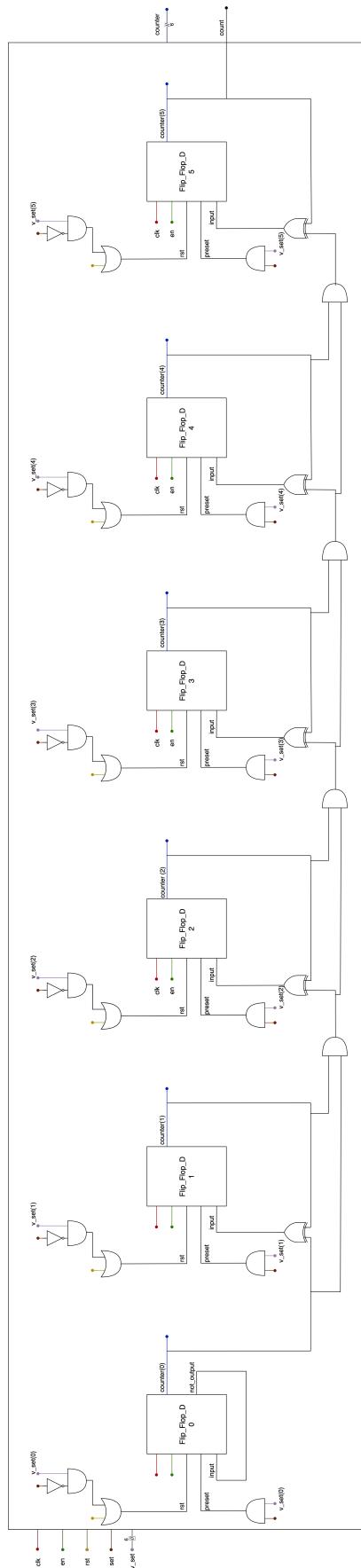


Figura 5.3: Contatore modulo 64 - approccio strutturale

## Contatore modulo 32

Il contatore modulo 32 è esteriormente identico a quello modulo 64:

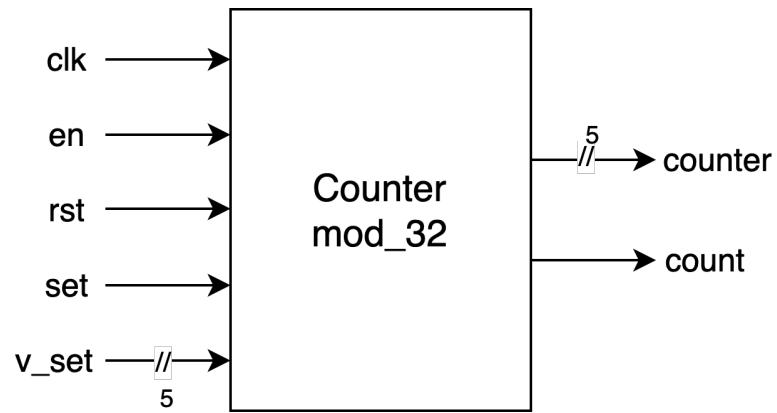


Figura 5.4: Contatore modulo 32

Analogamente al caso precedente, si compone strutturalmente con i Flip-Flop D:

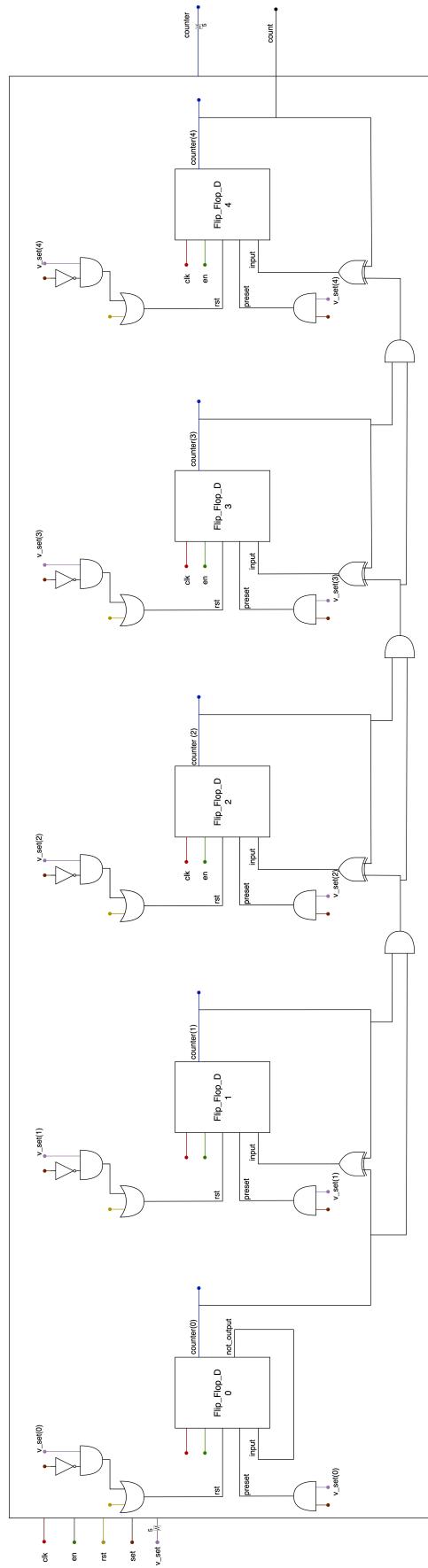


Figura 5.5: Contatore modulo 32 - approccio strutturale

## Contatore modulo 60

Per progettare un contatore modulo 60, si parte da un contatore modulo 64 facendo in modo che ogni volta raggiunga 59 si resetti.

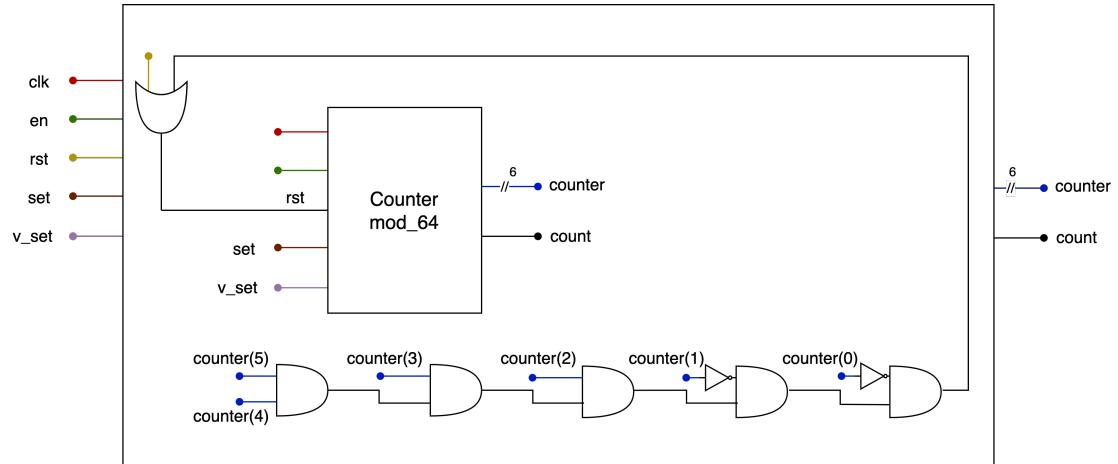


Figura 5.6: Contatore modulo 60

## Contatore modulo 24

Analogamente al contatore precedente, si realizza il contatore modulo 24 utilizzando il contatore modulo 32 in modo che si resetti ogni qualvolta raggiunga 23

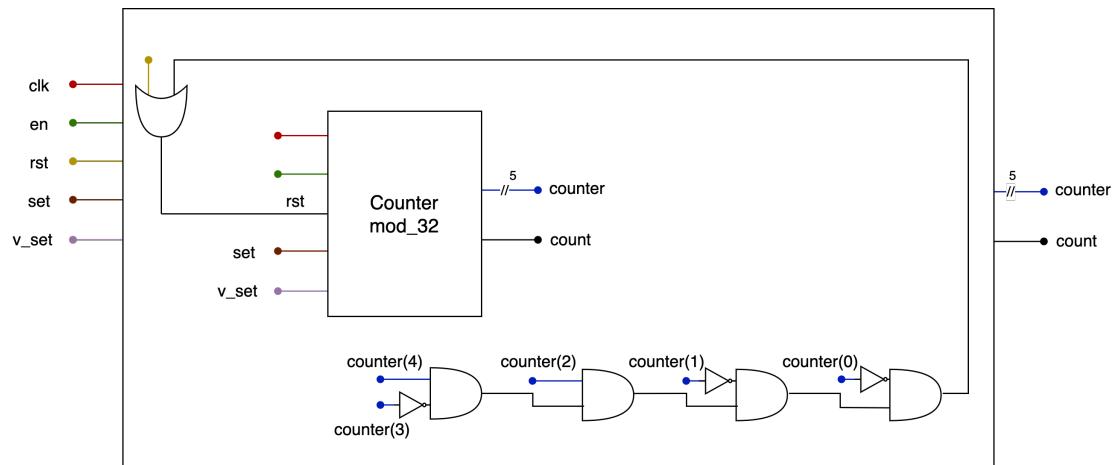


Figura 5.7: Contatore modulo 24

## Cronometro

Si possono ora effettuare i collegamenti per creare un cronometro: ovviamente servono due contatori modulo 60 per secondi e minuti, e un contatore modulo 24:

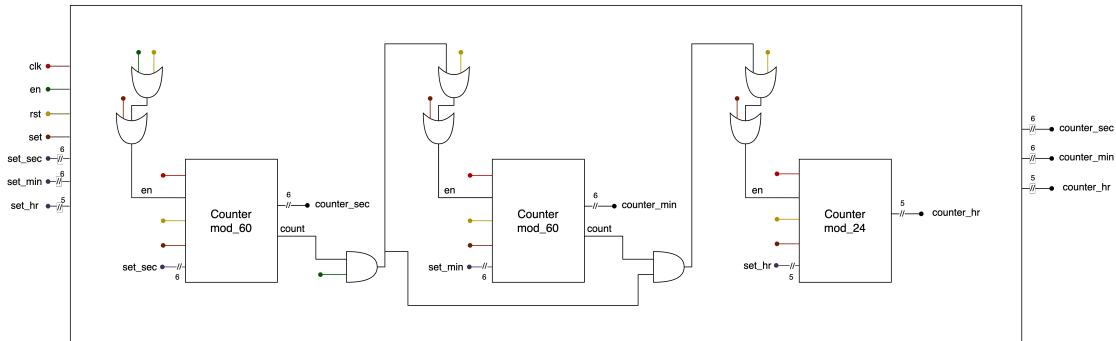


Figura 5.8: Cronometro

### 5.1.2 Implementazione

Si può a questo punto procedere con l'implementazione in VHDL del nostro progetto, partendo dalle sue componenti.

## Flip-Flop D

```

1 library ieee;
2 use      ieee.std_logic_1164.all;
3
4 entity ffD is
5   port
6   (
7     clk          :  in  std_logic;
8     en           :  in  std_logic;
9     rst          :  in  std_logic;
10    preset       :  in  std_logic;
11    input         :  in  std_logic;
12    output        :  out std_logic;
13    not_output   :  out std_logic

```

```
14      );
15 end entity;
16
17 architecture behavioral of ffD is
18     signal temp      : std_logic;
19
20 begin
21     process(clk)
22     begin
23         if(clk'event and clk='1') then
24             if rst = '1' then
25                 temp      <=  '0';
26             elsif en = '1' then
27                 if preset = '1' then
28                     temp      <=  '1';
29                 else
30                     temp      <=  input;
31                 end if;
32             end if;
33         end if;
34     end process;
35     output      <=  temp;
36     not_output <=  not temp;
37 end behavioral;
```

Code 5.1: ffD.vhdl

Il Flip-Flop D è stato implementato con un approccio comportamentale e con reset sincrono.

## Contatore modulo 64

Utilizzando il Flip-Flop D, si implementa ora il contatore modulo 64.

```
1 library ieee;
2 use      ieee.std_logic_1164.all;
3
4 entity counter_mod_64 is
5     port
6     (
7         clk      : in std_logic;
```

```

8      en      :  in  std_logic;
9      rst      :  in  std_logic;
10     set      :  in  std_logic;
11     v_set    :  in  std_logic_vector(5 downto 0);
12     counter :  out std_logic_vector(5 downto 0);
13     count    :  out std_logic
14   );
15 end entity;
16
17 architecture structural of counter_mod_64 is
18   signal temp_counter    :  std_logic_vector(5 downto 0)      := 
19   ← (others => '0');
20   signal back            :  std_logic;
21
22   signal resets          :  std_logic_vector(5 downto 0);
23   signal sets            :  std_logic_vector(5 downto 0);
24   signal counters         :  std_logic_vector(5 downto 0);
25
26
27 component ffD is
28   port
29   (
30     clk      :  in  std_logic;
31     en       :  in  std_logic;
32     rst      :  in  std_logic;
33     preset   :  in  std_logic;
34     input    :  in  std_logic;
35     output   :  out std_logic;
36     not_output :  out std_logic
37   );
38 end component;
39
40 begin
41   resets(0)    <=  (set and not (v_set(0))) or (rst);
42   sets(0)      <=  set and v_set(0);
43
44   counter_0 : ffD
45     port map (
46       clk,
47       en,
48       resets(0),
49       sets(0),
50       back,
51       temp_counter(0),
52       back

```

```
53      );
54
55      resets(1)    <=  (set and not v_set(1)) or rst;
56      sets(1)      <=  set and v_set(1);
57      counters(1)  <=  temp_counter(0) xor temp_counter(1);
58
59      counter_1 : ffD
60      port map (
61          clk,
62          en,
63          resets(1),
64          sets(1),
65          counters(1),
66          temp_counter(1)
67      );
68
69      resets(2)    <=  (set and not v_set(2)) or rst;
70      sets(2)      <=  set and v_set(2);
71      counters(2)  <=  (temp_counter(0) and temp_counter(1)) xor
72          ↳ temp_counter(2);
73
74      counter_2 : ffD
75      port map (
76          clk,
77          en,
78          resets(2),
79          sets(2),
80          counters(2),
81          temp_counter(2)
82      );
83
83      resets(3)    <=  (set and not v_set(3)) or rst;
84      sets(3)      <=  set and v_set(3);
85      counters(3)  <=  (temp_counter(0) and temp_counter(1) and
86          ↳ temp_counter(2)) xor temp_counter(3);
87
87      counter_3 : ffD
88      port map (
89          clk,
90          en,
91          resets(3),
92          sets(3),
93          counters(3),
94          temp_counter(3)
95      );
96
```

```
97      resets(4)    <=  (set and not v_set(4)) or rst;
98      sets(4)      <=  set and v_set(4);
99      counters(4)  <=  (temp_counter(0) and temp_counter(1) and
100        ↳ temp_counter(2) and temp_counter(3)) xor temp_counter(4);
101
102      counter_4 : ffD
103      port map (
104          clk,
105          en,
106          resets(4),
107          sets(4),
108          counters(4),
109          temp_counter(4)
110      );
111
112      resets(5)    <=  (set and not v_set(5)) or rst;
113      sets(5)      <=  set and v_set(5);
114      counters(5)  <=  (temp_counter(0) and temp_counter(1) and
115        ↳ temp_counter(2) and temp_counter(3) and temp_counter(4)) xor
116        ↳ temp_counter(5);
117
118      counter_5 : ffD
119      port map (
120          clk,
121          en,
122          resets(5),
123          sets(5),
124          counters(5),
125          temp_counter(5)
126      );
127
128      counter <= temp_counter;
129      count   <= temp_counter(5);
130
131  end structural;
```

Code 5.2: counter\_mod\_64.vhdl

## Contatore modulo 32

In modo analogo, si procede per il contatore modulo 32.

---

```
1  library ieee;
2  use      ieee.std_logic_1164.all;
```

```

3
4 entity counter_mod_32 is
5   port
6   (
7     clk      : in std_logic;
8     en       : in std_logic;
9     rst      : in std_logic;
10    set      : in std_logic;
11    v_set    : in std_logic_vector(4 downto 0);
12    counter : out std_logic_vector(4 downto 0);
13    count    : out std_logic
14  );
15 end entity;
16
17 architecture structural of counter_mod_32 is
18   signal temp_counter    : std_logic_vector(4 downto 0)      := 
19   ← (others => '0');
20   signal back            : std_logic;
21   signal resets          : std_logic_vector(4 downto 0);
22   signal sets            : std_logic_vector(4 downto 0);
23   signal counters        : std_logic_vector(4 downto 0);
24
25
26 component ffD is
27   port
28   (
29     clk      : in std_logic;
30     en       : in std_logic;
31     rst      : in std_logic;
32     preset   : in std_logic;
33     input    : in std_logic;
34     output   : out std_logic;
35     not_output : out std_logic
36   );
37 end component;
38
39 begin
40
41   resets(0)    <= (set and not (v_set(0))) or (rst);
42   sets(0)      <= set and v_set(0);
43
44   counter_0 : ffD
45     port map (
46       clk,
47       en,

```

```
48         resets(0),
49         sets(0),
50         back,
51         temp_counter(0),
52         back
53     );
54
55     resets(1)    <=  (set and not v_set(1)) or rst;
56     sets(1)      <=  set and v_set(1);
57     counters(1)  <=  temp_counter(0) xor temp_counter(1);
58
59     counter_1 : ffD
60     port map (
61         clk,
62         en,
63         resets(1),
64         sets(1),
65         counters(1),
66         temp_counter(1)
67     );
68
69     resets(2)    <=  (set and not v_set(2)) or rst;
70     sets(2)      <=  set and v_set(2);
71     counters(2)  <=  (temp_counter(0) and temp_counter(1)) xor
72       ↳  temp_counter(2);
73
74     counter_2 : ffD
75     port map (
76         clk,
77         en,
78         resets(2),
79         sets(2),
80         counters(2),
81         temp_counter(2)
82     );
83
83     resets(3)    <=  (set and not v_set(3)) or rst;
84     sets(3)      <=  set and v_set(3);
85     counters(3)  <=  (temp_counter(0) and temp_counter(1) and
86       ↳  temp_counter(2)) xor temp_counter(3);
87
88     counter_3 : ffD
89     port map (
90         clk,
91         en,
92         resets(3),
```

```
92      sets(3),
93      counters(3),
94      temp_counter(3)
95  );
96
97  resets(4)    <=  (set and not v_set(4)) or rst;
98  sets(4)      <=  set and v_set(4);
99  counters(4)  <=  (temp_counter(0) and temp_counter(1) and
100   ↳ temp_counter(2) and temp_counter(3)) xor temp_counter(4);
101
102  counter_4 : ffD
103  port map (
104    clk,
105    en,
106    resets(4),
107    sets(4),
108    counters(4),
109    temp_counter(4)
110  );
111
112  counter <= temp_counter;
113  count   <= temp_counter(4);
end structural;
```

Code 5.3: counter\_mod\_32.vhdl

## Contatore modulo 60

Si implemeta ora il contatore modulo 60, partendo da quello modulo 64.

---

```
1  library ieee;
2  use     ieee.std_logic_1164.all;
3
4  entity counter_mod_60 is
5    port
6    (
7      clk      : in std_logic;
8      en       : in std_logic;
9      rst      : in std_logic;
10     set      : in std_logic;
11     v_set    : in std_logic_vector(5 downto 0);
```

```
12      counter :  out std_logic_vector(5 downto 0);
13      count   :  out std_logic
14  );
15 end counter_mod_60;
16
17 architecture structural of counter_mod_60 is
18     signal temp_counter    :  std_logic_vector(5 downto 0);
19     signal last_number_rs :  std_logic;
20
21     signal reset           :  std_logic;
22
23 component counter_mod_64 is
24     port
25     (
26         clk      :  in  std_logic;
27         en       :  in  std_logic;
28         rst      :  in  std_logic;
29         set      :  in  std_logic;
30         v_set    :  in  std_logic_vector(5 downto 0);
31         counter :  out std_logic_vector(5 downto 0);
32         count   :  out std_logic
33     );
34 end component;
35
36 begin
37     last_number_rs  <= temp_counter(5) and temp_counter(4) and
38             ← temp_counter(3) and temp_counter(2) and (not temp_counter(1))
39             ← and (not temp_counter(0));
40     reset          <= last_number_rs or rst;
41
42     counter_64: counter_mod_64
43     port map (clk, en, reset, set, v_set, temp_counter);
44
45     counter        <= temp_counter;
46     count          <= reset;
47 end structural;
```

Code 5.4: counter\_mod\_60.vhdl

## Contatore modulo 24

Analogamente al contatore modulo 60, si implementa il contatore modulo 24 a partire da quello di modulo 32.

```

1 library ieee;
2 use      ieee.std_logic_1164.all;
3
4 entity counter_mod_24 is
5   port
6   (
7     clk      : in std_logic;
8     en       : in std_logic;
9     rst      : in std_logic;
10    set      : in std_logic;
11    v_set    : in std_logic_vector(4 downto 0);
12    counter : out std_logic_vector(4 downto 0);
13    count    : out std_logic
14  );
15 end counter_mod_24;
16
17 architecture structural of counter_mod_24 is
18   signal temp_counter      : std_logic_vector(4 downto 0);
19   signal last_number_rs    : std_logic;
20
21   signal reset             : std_logic;
22
23 component counter_mod_32 is
24   port
25   (
26     clk      : in std_logic;
27     en       : in std_logic;
28     rst      : in std_logic;
29     set      : in std_logic;
30     v_set    : in std_logic_vector(4 downto 0);
31     counter : out std_logic_vector(4 downto 0);
32     count    : out std_logic
33   );
34 end component;
35
36 begin
37   last_number_rs  <= temp_counter(4) and (not temp_counter(3)) and
38   ↳ temp_counter(2) and temp_counter(1) and temp_counter(0);
39

```

```
39      reset          <= last_number_rs or rst;
40
41  counter_32: counter_mod_32
42    port map (clk, en, reset, set, v_set, temp_counter);
43
44  counter          <= temp_counter;
45  count            <= reset;
46 end structural;
```

Code 5.5: counter\_mod\_24.vhdl

## Cronometro

Si può finalmente implementare il cronometro, utilizzando gli ultimi due contatori implementati.

```
1  library ieee;
2  use     ieee.std_logic_1164.all;
3
4  entity stopwatch is
5    port
6    (
7      clk      : in std_logic;
8      rst      : in std_logic;
9      set      : in std_logic;
10     en       : in std_logic;
11     set_sec  : in std_logic_vector(5 downto 0);
12     set_min  : in std_logic_vector(5 downto 0);
13     set_hr   : in std_logic_vector(4 downto 0);
14     seconds  : out std_logic_vector(5 downto 0);
15     minutes  : out std_logic_vector(5 downto 0);
16     hours    : out std_logic_vector(4 downto 0)
17   );
18 end stopwatch;
19
20 architecture structural of stopwatch is
21   signal en_count   : std_logic_vector(1 downto 0);
22
23   signal temp_enable : std_logic_vector(2 downto 0);
24
25   component counter_mod_60
26     port
```

```

27      (
28          clk      :  in  std_logic;
29          en       :  in  std_logic;
30          rst      :  in  std_logic;
31          set      :  in  std_logic;
32          v_set    :  in  std_logic_vector(5 downto 0);
33          counter :  out std_logic_vector(5 downto 0);
34          count    :  out std_logic
35      );
36  end component;

37
38 component counter_mod_24
39     port
40     (
41         clk      :  in  std_logic;
42         en       :  in  std_logic;
43         rst      :  in  std_logic;
44         set      :  in  std_logic;
45         v_set    :  in  std_logic_vector(4 downto 0);
46         counter :  out std_logic_vector(4 downto 0);
47         count    :  out std_logic
48     );
49 end component;
50 begin
51     --seconds
52     temp_enable(0) <= en or rst or set;
53     counter_seconds:   counter_mod_60
54     port map (clk, temp_enable(0), rst, set, set_sec, seconds,
55               en_count(0));
56
57     --minutes
58     temp_enable(1) <= (en and en_count(0)) or rst or set;
59     counter_minutes:   counter_mod_60
60     port map (clk, temp_enable(1), rst, set, set_min, minutes,
61               en_count(1));
62
63     --hours
64     temp_enable(2) <= (en and en_count(0) and en_count(1)) or rst or
65               set;
66     counter_hours:   counter_mod_24
67     port map (clk, temp_enable(2), rst, set, set_hr, hours);
68 end structural;

```

Code 5.6: stopwatch.vhdl

### 5.1.3 Simulazione

Per la simulazione, si implementa il seguente testbench:

```

1  -- Testbench created online at:
2  --
3  → https://www.doulos.com/knowhow/perl/vhdl-testbench-creation-using-perl/
4  -- Copyright Doulos Ltd
5
6  library IEEE;
7  use IEEE.Std_logic_1164.all;
8  use IEEE.Numeric_Std.all;
9
10 entity stopwatch_tb is
11 end;
12
13 architecture bench of stopwatch_tb is
14
15 component stopwatch
16     port
17     (
18         clk      : in  std_logic;
19         rst      : in  std_logic;
20         set      : in  std_logic;
21         en       : in  std_logic;
22         set_sec : in  std_logic_vector(5 downto 0);
23         set_min : in  std_logic_vector(5 downto 0);
24         set_hr  : in  std_logic_vector(4 downto 0);
25         seconds : out std_logic_vector(5 downto 0);
26         minutes : out std_logic_vector(5 downto 0);
27         hours   : out std_logic_vector(4 downto 0)
28     );
29 end component;
30
31 signal clk: std_logic;
32 signal rst: std_logic;
33 signal set: std_logic;
34 signal en: std_logic;
35 signal set_sec: std_logic_vector(5 downto 0);
36 signal set_min: std_logic_vector(5 downto 0);
37 signal set_hr: std_logic_vector(4 downto 0);
38 signal seconds: std_logic_vector(5 downto 0);
39 signal minutes: std_logic_vector(5 downto 0);
40 signal hours: std_logic_vector(4 downto 0) ;

```

```

41 begin
42
43     uut: stopwatch port map ( clk      => clk,
44                               rst      => rst,
45                               set      => set,
46                               en       => en,
47                               set_sec  => set_sec,
48                               set_min  => set_min,
49                               set_hr   => set_hr,
50                               seconds  => seconds,
51                               minutes  => minutes,
52                               hours    => hours );
53
54     clock_process: process
55         begin
56             clk <= '0';
57             while True loop
58                 wait for 5 ns; -- Periodo del clock (adattare a seconda
59                             -- delle esigenze)
60                 clk <= not clk;
61             end loop;
62         end process;
63
64         -- Stimuli per il testbench
65         stimulus: process
66         begin
67             -- Inizializzazione
68             rst <= '1'; -- Reset iniziale
69             wait for 10 ns;
70             rst <= '0';
71
72             -- Test case 1: Avvio del cronometro da zero
73             en <= '1';
74             wait for 1 ms;
75             en <= '0';
76
77             -- Verifica dei risultati (es. utilizzando asserzioni)
78
79             -- Test case 2: Impostazione di un tempo iniziale e avvio
80             set_sec <= "000111";
81             set_min <= "000011";
82             set_hr  <= "00001";
83             set <= '1';
84             wait for 10 ns;
85             set <= '0';
86             en <= '1';

```

```

86      wait for 500 ms;
87      en <= '0';
88
89      -- Verifica dei risultati
90
91      -- ... Altri test case ...
92
93      wait for 5 ns;
94  end process;
95
96
97 end;

```

Code 5.7: stopwatch\_tb.vhdl

Lanciando la simulazione si ha

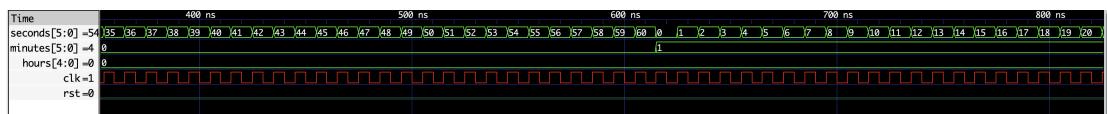


Figura 5.9: Simulazione Cronometro

Si vuole far notare che per materiali, è impossibile riportare nella documentazione l'intera simulazione.

Per cui qualore si volesse testare il cronometro, il codice VHDL si trova nella repository GitHub associata al progetto, nella cartella Esercizio 5 . 1.

## 5.2 Implementazione su board del punto precedente

### 5.2.1 Traccia

Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando i display a 7 segmenti per la visualizzazione dell'orario (o una combinazione di display e led nel caso in cui i display a disposizione siano in numero inferiore a quello necessario), gli switch per l'immissione dell'orario iniziale e due bottoni, uno per il set dell'orario e uno per il reset. Si utilizzi una codifica a scelta dello studente per la visualizzazione dell'orario sui display (esadecimale o decimale).

### 5.2.2 Implementazione

Per l'implementazione su board è stato necessario utilizzare il visore presente sulla board, per fare ciò sono stati usati i seguenti codici:

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity anodes_manager is
5     Port ( counter : in STD_LOGIC_VECTOR (2 downto 0);
6             enable_digit : in STD_LOGIC_VECTOR (7 downto 0);
7             anodes : out STD_LOGIC_VECTOR (7 downto 0)
8             );
9 end anodes_manager;
10
11 architecture Behavioral of anodes_manager is
12
```

```
13 signal anodes_switching : std_logic_vector(7 downto 0) := (others =>
14   => '0');
15
16 begin
17
18   anodes <= not anodes_switching OR not enable_digit;
19
20   anodes_process: process(counter)
21   begin
22     --a seconda del valore di counter le cifre si illuminano una
23     --> alla volta da destra a sinistra
24     case counter is
25       when "000" =>
26         anodes_switching <= "00000001";
27       when "001" =>
28         anodes_switching <= "00000010";
29       when "010" =>
30         anodes_switching <= "00000100";
31       when "011" =>
32         anodes_switching <= "00001000";
33       when "100" =>
34         anodes_switching <= "00010000";
35       when "101" =>
36         anodes_switching <= "00100000";
37       when "110" =>
38         anodes_switching <= "01000000";
39       when "111" =>
40         anodes_switching <= "10000000";
41       when others =>
42         anodes_switching <= (others => '0');
43     end case;
44   end process;
45
46
47 end Behavioral;
```

---

Code 5.8: anodes\_manager.vhdl

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

```

4  entity anodes_manager is
5      Port ( counter : in STD_LOGIC_VECTOR (2 downto 0);
6              enable_digit : in STD_LOGIC_VECTOR (7 downto 0);
7              anodes : out STD_LOGIC_VECTOR (7 downto 0)
8                  );
9  end anodes_manager;
10
11 architecture Behavioral of anodes_manager is
12
13 signal anodes_switching : std_logic_vector(7 downto 0) := (others =>
14     '0');
15
16 begin
17
18 anodes <= not anodes_switching OR not enable_digit;
19
20 anodes_process: process(counter)
21 begin
22     --a seconda del valore di caunter le cifre si illuminano una
23     --→ alla volta da destra a sinistra
24     case counter is
25         when "000" =>
26             anodes_switching <= "00000001";
27         when "001" =>
28             anodes_switching <= "00000010";
29         when "010" =>
30             anodes_switching <= "00000100";
31         when "011" =>
32             anodes_switching <= "00001000";
33         when "100" =>
34             anodes_switching <= "00010000";
35         when "101" =>
36             anodes_switching <= "00100000";
37         when "110" =>
38             anodes_switching <= "01000000";
39         when "111" =>
40             anodes_switching <= "10000000";
41         when others =>
42             anodes_switching <= (others => '0');
43     end case;
44
45
46 end process;
47
48 end Behavioral;

```

Code 5.9: catode\_manager.vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity display_seven_segments is
5     Generic(
6         CLKIN_freq : integer := 100000000;
7         CLKOUT_freq : integer := 500
8     );
9     Port ( CLK : in STD_LOGIC;
10            RST : in STD_LOGIC;
11            VALUE : in STD_LOGIC_VECTOR (31 downto 0);
12            ENABLE : in STD_LOGIC_VECTOR (7 downto 0); -- decide
13            -- quali cifre abilitare
14            DOTS : in STD_LOGIC_VECTOR (7 downto 0); -- decide quali
15            -- punti visualizzare
16            ANODES : out STD_LOGIC_VECTOR (7 downto 0);
17            CATHODES : out STD_LOGIC_VECTOR (7 downto 0));
18 end display_seven_segments;
19
20 architecture Structural of display_seven_segments is
21
22 signal counter : std_logic_vector(2 downto 0);
23 signal clock_filter_out : std_logic := '0';
24
25 COMPONENT counter_mod8
26     PORT(
27         clock : in STD_LOGIC;
28         reset : in STD_LOGIC;
29         enable : in STD_LOGIC;
30         counter : out STD_LOGIC_VECTOR (2 downto 0)
31     );
32 END COMPONENT;
33
34 COMPONENT cathodes_manager
35     PORT(
36         counter : IN std_logic_vector(2 downto 0);
37         value : IN std_logic_vector(31 downto 0);
38         dots : IN std_logic_vector(7 downto 0);
39         cathodes : OUT std_logic_vector(7 downto 0)
40     );

```

```

39 END COMPONENT;
40
41 COMPONENT anodes_manager
42     PORT (
43
44         counter : IN std_logic_vector(2 downto 0);
45         enable_digit : IN std_logic_vector(7 downto 0);
46         →
47         anodes : OUT std_logic_vector(7 downto 0)
48     ) ;
49 END COMPONENT;
50
51 COMPONENT clock_filter
52     GENERIC (
53         CLKIN_freq : integer := 1000000000;
54         CLKOUT_freq : integer := 500
55     ) ;
56     PORT (
57         clock_in : IN std_logic;
58         reset : in STD_LOGIC;
59         clock_out : OUT std_logic
60     ) ;
61 END COMPONENT;
62 begin
63     --il clock filter genera un segnale di abilitazione per il contatore
64     --→ mod8 che viene usato
65     --come segnale di conteggio e quindi di fatto fornisce la frequenza
66     --→ con cui viene modificata
67     --la cifra da mostrare
68
69     clk_filter: clock_filter GENERIC MAP (
70         CLKIN_freq => CLKIN_freq,
71         CLKOUT_freq => CLKOUT_freq
72     )
73     PORT MAP (
74         clock_in => CLK,
75         reset => RST,
76         clock_out => clock_filter_out
77     ) ;
78
79     counter_instance: counter_mod8 port map (
80         clock => CLK,
81         enable => clock_filter_out,
82         reset => RST,
83         counter => counter
84     ) ;

```

```

82 --il valore di conteggio viene usato dal gestore dei catodi e degli
83 --→ anodi per
84 --selezionare l'anodo da accendere e il suo rispettivo valore
85 cathodes_instance: cathodes_manager port map(
86     counter => counter,
87     value => value,
88     dots => dots,
89     cathodes => cathodes
90 );
91
92 anodes_instance: anodes_manager port map(
93     counter => counter,
94     enable_digit => enable,
95     anodes => anodes
96 );
97
98 end Structural;
99

```

Code 5.10: display\_seven\_segments.vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity clock_filter is
5     generic(
6         CLKIN_freq : integer := 100000000; --clock
7             -- board 100MHz
8         CLKOUT_freq : integer := 500
9             -- frequenza desiderata 500Hz
10    );
11
12 Port (
13     clock_in : in STD_LOGIC;
14     reset : in STD_LOGIC;
15     clock_out : out STD_LOGIC -- attenzione: non e' un vero
16             -- clock ma un impulso che sare' usato come enable
17 );
18 end clock_filter;
19
20
21 architecture Behavioral of clock_filter is
22
23 signal clockfx : std_logic := '0';
24

```

```

20 constant count_max_value : integer := CLKIN_freq/ (CLKOUT_freq)-1;
21
22 begin
23
24   clock_out <= clockfx;
25
26   count_for_division: process(clock_in)
27
28   variable counter : integer range 0 to count_max_value := 0;
29   begin
30
31     if rising_edge(clock_in) then
32       if( reset = '1') then
33         counter := 0;
34         clockfx <= '0';
35       else
36         if counter = count_max_value then
37           clockfx <= '1';
38           counter := 0;
39         else
40           clockfx <= '0';
41           counter := counter + 1;
42         end if;
43       end if;
44     end if;
45   end process;
46
47
48 end Behavioral;
49

```

Code 5.11: clock\_filter.vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 -- contatore utilizzato scorrere le cifre da visualizzare
5 entity counter_mod8 is
6   Port ( clock : in STD_LOGIC;
7          reset : in STD_LOGIC;
8          enable : in STD_LOGIC; --enable viene dal divisore
9          ↳ di frequenza
10         counter : out STD_LOGIC_VECTOR (2 downto 0));
11 end counter_mod8;

```

```
11
12 architecture Behavioral of counter_mod8 is
13
14 signal c : std_logic_vector (2 downto 0) := (others => '0');
15 begin
16 counter <= c;
17
18 counter_process: process(clock)
19 begin
20
21     if(rising_edge(clock)) then
22         if reset = '1' then
23             c <= (others => '0');
24         elsif enable = '1' then
25             c <= std_logic_vector(unsigned(c) + 1);
26         end if;
27         end if;
28     end process;
29
30 end Behavioral;
```

---

Code 5.12: count\_mod8.vhdl

Inoltre per avere una rappresentazione coerente sui visori, sono stati implementati i seguenti codici per distinguere decine e unità dalle uscite del cronometro:

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity separator is
6 port(
7     num_in : in std_logic_vector (5 downto 0);
8     clk : in std_logic;
9     unit : out std_logic_vector (3 downto 0);
10    dec : out std_logic_vector (3 downto 0)
11    );
12 end separator;
13
14 architecture Behavioral of separator is
15    signal temp_dec, temp_un : integer;
```

---

```

16 begin
17
18 calc : process(clk)
19 begin
20
21 temp_dec <= (to_integer(unsigned(num_in))) / 10;
22 temp_un <= (to_integer(unsigned(num_in))) mod 10;
23
24 end process;
25
26 unit <= std_logic_vector(to_unsigned(temp_un, 4));
27 dec <= '0' & std_logic_vector(to_unsigned(temp_dec, 3));
28
29 end Behavioral;

```

Code 5.13: separator.vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity sep_h_m_s is
5   Port (
6     CLK: in STD_LOGIC;
7     sec, min: in STD_LOGIC_VECTOR(5 downto 0);
8     hours: in STD_LOGIC_VECTOR(4 downto 0);
9     sec_uni, min_uni, h_uni, sec_dec, min_dec, h_dec: out
10    STD_LOGIC_VECTOR(3 downto 0)
11  );
12 end sep_h_m_s;
13
14 architecture Behavioral of sep_h_m_s is
15
16 component separator
17 port(
18   num_in : in std_logic_vector (5 downto 0);
19   clk : in std_logic;
20   unit : out std_logic_vector (3 downto 0);
21   dec : out std_logic_vector (3 downto 0)
22 );
23 end component;
24
25
26 begin

```

```
27
28 sec_separator: separator
29     port map(
30         num_in => sec,
31         clk => CLK,
32         unit => sec_uni,
33         dec => sec_dec
34     );
35
36 min_separator: separator
37     port map(
38         num_in => min,
39         clk => CLK,
40         unit => min_uni,
41         dec => min_dec
42     );
43
44 h_temp <= '0' & hours;
45
46 h_separator: separator
47     port map(
48         num_in => h_temp,
49         clk => CLK,
50         unit => h_uni,
51         dec => h_dec
52     );
53
54 end Behavioral;
```

Code 5.14: anodes\_manager.vhdl

Si è inoltre scelto di usare un divisore di frequenza per ottenere un clock più adatto al funzionamento del progetto:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity frequency_divider is
6     Port (
7         clock_in : in STD_LOGIC;    -- Clock di ingresso
8         reset : in STD_LOGIC;      -- Segnale di reset
9         clock_out : out STD_LOGIC -- Clock di uscita, divisore di
                           -- frequenza
```

```

10      );
11 end frequency_divider;
12
13 architecture Behavioral of frequency_divider is
14
15   -- Definire il valore massimo del contatore in base alla
16   -- divisione desiderata
17   -- Supponiamo un clock di ingresso di 100 MHz, per ottenere una
18   -- frequenza di uscita di 1 Hz:
19   constant CLOCK_FREQ : integer := 100_000_000; -- Frequenza del
20   -- clock di ingresso (100 MHz)
21   constant DIVISOR : integer := 100_000_000;      -- Divisione
22   -- desiderata (1 Hz: 100 MHz / 100_000_000)
23   constant COUNT_MAX : integer := DIVISOR / 2 - 1; -- Calcola il
24   -- massimo valore del contatore (per ottenere un periodo
25   -- completo)
26
27   signal counter : integer range 0 to COUNT_MAX := 0; -- Contatore
28   -- per dividere la frequenza
29   signal clock_signal : STD_LOGIC := '0'; -- Segnale di clock di
30   -- uscita
31
32 begin
33
34   -- Processo che divide la frequenza
35   process(clock_in)
36     begin
37       if rising_edge(clock_in) then
38         if reset = '1' then
39           counter <= 0; -- Reset del contatore
40           clock_signal <= '0'; -- Reset del segnale di clock di
41           -- uscita
42         else
43             if counter = COUNT_MAX then
44               counter <= 0; -- Reset del contatore al
45               -- raggiungimento del massimo
46               clock_signal <= not clock_signal; -- Inverti il
47               -- segnale di clock di uscita
48             else
49               counter <= counter + 1; -- Incrementa il
50               -- contatore
51             end if;
52           end if;
53         end if;
54       end process;
55 
```

```

44      -- Collega il segnale di uscita al segnale di clock
45      clock_out <= clock_signal;
46
47 end Behavioral;

```

Code 5.15: freq\_divider.vhdl

Per la gestione degli input, soprattutto relativi al *SET* di secondi, minuti e ore, si è implementata una unità di controllo, che sulla base del bottone premuto caricherà rispettivamente secondi (BTNU), minuti (BTNL) e ore (BTNR), e con un ulteriore bottone viene abilitato il *SET* (BTND). Inoltre con il bottone centrale si abilita il *RESET*.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity control_unit is
5    Port (
6      CLK, RST, SET: in STD_LOGIC;
7      load_s, load_m, load_h: in STD_LOGIC;
8      value_in: in STD_LOGIC_VECTOR(5 downto 0);
9      sec_out, min_out: out STD_LOGIC_VECTOR(5 downto 0);
10     h_out: out STD_LOGIC_VECTOR(4 downto 0)
11   );
12 end control_unit;
13
14 architecture Behavioral of control_unit is
15
16 signal sec_val, min_val: STD_LOGIC_VECTOR(5 downto 0);
17 signal h_val: STD_LOGIC_VECTOR(4 downto 0);
18 signal sec_set, min_set: STD_LOGIC_VECTOR(5 downto 0);
19 signal h_set: STD_LOGIC_VECTOR(4 downto 0);
20
21 begin
22
23 sec_out <= sec_val;
24 min_out <= min_val;
25 h_out <= h_val;
26
27 main: process(CLK)
28 begin

```

```

29      if (CLK'event AND CLK = '0') then
30          if (RST = '1') then
31              sec_val <= (others => '0');
32              min_val <= (others => '0');
33              h_val <= (others => '0');
34      else
35
36          if (load_s = '1') then
37              sec_set <= value_in;
38          elsif (load_m = '1') then
39              min_set <= value_in;
40          elsif (load_h = '1') then
41              h_set <= value_in(4 downto 0);
42          end if;
43          if SET = '1' then
44              sec_val <= sec_set;
45              min_val <= min_set;
46              h_val <= h_set;
47          end if;
48      end if;
49  end if;
50
51
52 end Behavioral;

```

Code 5.16: control\_unit.vhdl

La gestione del sistema nel suo complesso è implementata con il seguente codice:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity cron_onBOARD is
5     Port (
6         CLK, RST, SET: in STD_LOGIC;
7         load_s, load_m, load_h: in STD_LOGIC;
8         value_in: in STD_LOGIC_VECTOR(5 downto 0);
9         anodes_out : out STD_LOGIC_VECTOR (7 downto 0); --anodi e
10        → catodi delle cifre, sono un output del topmodule
11        cathodes_out : out STD_LOGIC_VECTOR (7 downto 0)
12    );
13 end cron_onBOARD;

```

```

13
14 architecture Behavioral of cron_onBOARD is
15 component control_unit
16     Port (
17         CLK, RST, SET: in STD_LOGIC;
18         load_s, load_m, load_h: in STD_LOGIC;
19         value_in: in STD_LOGIC_VECTOR(5 downto 0);
20         sec_out, min_out: out STD_LOGIC_VECTOR(5 downto 0);
21         h_out: out STD_LOGIC_VECTOR(4 downto 0)
22     );
23 end component;
24
25 component stopwatch
26     port
27     (
28         clk      : in std_logic;
29         rst      : in std_logic;
30         set      : in std_logic;
31         en       : in std_logic;
32         set_sec : in std_logic_vector(5 downto 0);
33         set_min : in std_logic_vector(5 downto 0);
34         set_hr  : in std_logic_vector(4 downto 0);
35         seconds : out std_logic_vector(5 downto 0);
36         minutes : out std_logic_vector(5 downto 0);
37         hours   : out std_logic_vector(4 downto 0)
38     );
39 end component;
40
41 component frequency_divider is
42     Port (
43         clock_in : in STD_LOGIC; -- Clock di ingresso
44         reset : in STD_LOGIC; -- Segnale di reset
45         clock_out : out STD_LOGIC -- Clock di uscita, divisore di
46             -- frequenza
47     );
48 end component;
49
50 component display_seven_segments is
51     Generic(
52         CLKIN_freq : integer := 100000000;
53         CLKOUT_freq : integer := 500
54             );
55     Port ( CLK : in STD_LOGIC;
56             RST : in STD_LOGIC;
57             VALUE : in STD_LOGIC_VECTOR (31 downto 0);

```

```

57      ENABLE : in STD_LOGIC_VECTOR (7 downto 0); -- decide
      ↳ quali cifre abilitare
58      DOTS : in STD_LOGIC_VECTOR (7 downto 0); -- decide quali
      ↳ punti visualizzare
59      ANODES : out STD_LOGIC_VECTOR (7 downto 0);
60      CATHODES : out STD_LOGIC_VECTOR (7 downto 0));
61 end component;
62
63 component sep_h_m_s
64   Port (
65     CLK: in STD_LOGIC;
66     sec, min: in STD_LOGIC_VECTOR(5 downto 0);
67     hours: in STD_LOGIC_VECTOR(4 downto 0);
68     sec_uni, min_uni, h_uni, sec_dec, min_dec, h_dec: out
     ↳ STD_LOGIC_VECTOR(3 downto 0)
69   );
70 end component;
71
72
73 signal s_cu, m_cu: STD_LOGIC_VECTOR(5 downto 0);
74 signal h_cu: STD_LOGIC_VECTOR(4 downto 0);
75 signal sec_temp, min_temp: STD_LOGIC_VECTOR(5 downto 0);
76 signal h_temp: STD_LOGIC_VECTOR(4 downto 0);
77 signal fd_out: STD_LOGIC;
78 signal temp_sec_uni, temp_min_uni, temp_h_uni, temp_sec_dec,
     ↳ temp_min_dec, temp_h_dec: STD_LOGIC_VECTOR(3 downto 0);
79 begin
80 cu: control_unit
81   port map(
82     CLK => CLK,
83     RST => RST,
84     SET => SET,
85     load_s => load_s,
86     load_m => load_m,
87     load_h => load_h,
88     value_in => value_in,
89     sec_out => s_cu,
90     min_out => m_cu,
91     h_out => h_cu
92   );
93
94 cron: stopwatch
95   port map(
96     clk => fd_out,
97     rst => RST,
98     set => SET,

```

```

99      en => '1',
100     set_sec => s_cu,
101     set_min => m_cu,
102     set_hr => h_cu,
103     seconds => sec_temp,
104     minutes => min_temp,
105     hours => h_temp
106   );
107
108 fd: frequency_divider
109   port map(
110     clock_in => CLK,
111     reset => '0',
112     clock_out => fd_out
113   );
114
115 sep: sep_h_m_s
116   port map(
117     CLK => CLK,
118     sec => sec_temp,
119     min => min_temp,
120     hours => h_temp,
121     sec_uni => temp_sec_uni,
122     min_uni => temp_min_uni,
123     h_uni => temp_h_uni,
124     sec_dec => temp_sec_dec,
125     min_dec => temp_min_dec,
126     h_dec => temp_h_dec
127   );
128
129 dss: display_seven_segments
130   port map(
131     CLK => CLK,
132     RST => RST,
133     VALUE(3 downto 0) => temp_sec_uni,
134     VALUE(7 downto 4) => temp_sec_dec,
135     VALUE(11 downto 8) => temp_min_uni,
136     VALUE(15 downto 12) => temp_min_dec,
137     VALUE(19 downto 16) => temp_h_uni,
138     VALUE(23 downto 20) => temp_h_dec,
139     VALUE(31 downto 24) => (others => '0'),
140     ENABLE => "11111111",
141     DOTS => "01010100",
142     anodes => anodes_out,
143           cathodes => cathodes_out
144   );

```

---

145  
146   **end Behavioral;**

---

Code 5.17: cron\_onBoard.vhd

Si mostra anche il codice di Nexys-A7-50T-Master.xdc, fondamentale per la generazione del bitstream e di conseguenza per la programmazione della board.

```
## Clock signal

set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33}
[get_ports {CLK}]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports {CLK}];

##Switches

set_property -dict {PACKAGE_PIN J15    IOSTANDARD LVCMOS33}
[get_ports {value_in[0]}]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 }
[get_ports { value_in[1] }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13    IOSTANDARD LVCMOS33 }
[get_ports { value_in[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15    IOSTANDARD LVCMOS33 }
[get_ports { value_in[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17    IOSTANDARD LVCMOS33 }
[get_ports { value_in[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18    IOSTANDARD LVCMOS33 }
[get_ports { value_in[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]

##7 segment display

set_property -dict { PACKAGE_PIN T10    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16    IOSTANDARD LVCMOS33 }
```

```

[get_ports { cathodes_out[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13      IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15      IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11      IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18      IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[6] }]; #IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15      IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17      IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18      IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9       IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14      IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14      IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14      IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2       IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13      IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

##Buttons
#set_property -dict { PACKAGE_PIN C12      IOSTANDARD LVCMOS33 }
[get_ports { reset_in }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17      IOSTANDARD LVCMOS33 }
[get_ports { RST }]; #IO_L9P_T1_DQS_14 Sch=btnc

```

## CAPITOLO 5. ESERCIZIO 5

---

```
set_property -dict { PACKAGE_PIN M18      IOSTANDARD LVCMOS33 }
[get_ports { load_s }];
#IO_L4N_T0_D05_14 Sch=btぬ
set_property -dict { PACKAGE_PIN P17      IOSTANDARD LVCMOS33 }
[get_ports { load_m }];
#IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17      IOSTANDARD LVCMOS33 }
[get_ports { load_h }];
#IO_L10N_T1_D15_14 Sch=btぬ
set_property -dict { PACKAGE_PIN P18      IOSTANDARD LVCMOS33 }
[get_ports { SET }];
#IO_L9N_T1_DQS_D13_14 Sch=btぬ
```

# Capitolo 6

## Esercizio 6

### 6.1 Sistema di lettura - elaborazione - scrittura PO\\_PC

#### 6.1.1 Traccia

Progettare, implementare in VHDL e verificare mediante simulazione un sistema dotato di una memoria ROM di  $N$  locazioni da 8 bit ciascuna, una macchina combinatoria  $M$  in grado di trasformare (secondo una funzione a scelta dello studente) la stringa di 8 bit letta dalla ROM in una stringa di 4 bit, e una memoria MEM di  $N$  locazioni che memorizza la stringa in output da  $M$ . Il sistema si avvia in corrispondenza di un segnale di START che viene fornito esternamente. Una volta avviato, tramite un'apposita unità di controllo che gestisce la temporizzazione del sistema, viene scandita una locazione alla volta della ROM

e viene scritta la corrispondente locazione di MEM. Gli indirizzi di memoria sono forniti da un contatore. Le memorie ROM e MEM hanno rispettivamente un read e un write sincrono.

### 6.1.2 Progettazione

Per il progetto di questo sistema si riprende l'esercizio nel capitolo 2: **Sistema ROM + M**. Viene usato anche un contatore, per scandire una alla volta tutte le locazioni della ROM da cui prelevare la stringa contenente 8 bit. Come nell'esercizio precedente, l'uscita della ROM viene posta in ingresso alla macchina M, che somma i 4 bit più significativi dell'ingresso con i 4 bit meno significativi. L'uscita della macchina M viene posto in ingresso a una memoria MEM e poi caricato nella locazione corrispondente all'uscita del contatore; il funzionamento del sistema viene gestito da un'unità di controllo. La struttura del sistema sarà fatta in questo modo:

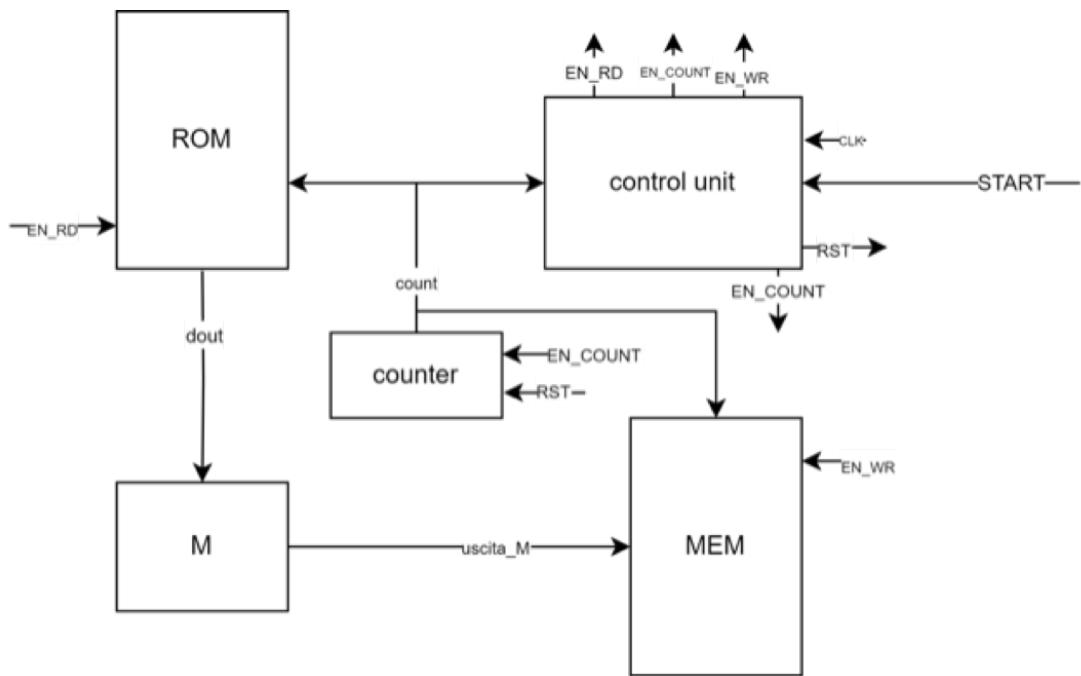


Figura 6.1: Schema a blocchi del sistema ROM + M + MEM

L'unità di controllo (CU) può essere efficacemente modellata come una macchina a stati finiti (FSM). Nel caso particolare avremo i seguenti stati:

- **idle**
- **read**
- **m\_work**
- **write**

L'unità di controllo può essere quindi rappresentata da un automa come si mostra in figura:

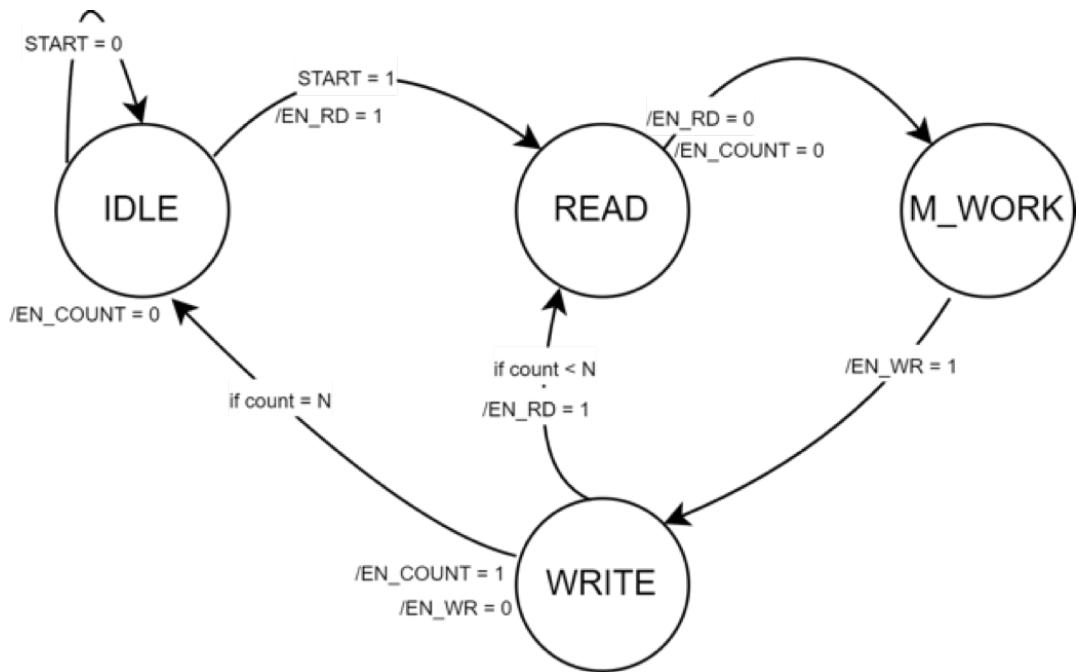


Figura 6.2: Macchina a stati della control unit di ROM + M + MEM

### 6.1.3 Implementazione

Il codice implementativo di M resta invariato, in quanto macchina puramente combinatoria. Si nota che viene richiesto che le operazioni di Read dalla ROM e di Write sulla memoria MEM siano svolte in modo sincrono. Quindi, a differenza della ROM usata nell'esercizio precedente, che era puramente combinatoria, le operazioni di lettura di questa ROM avvengono in sincronia con un segnale di clock. Questo segnale fornisce un riferimento temporale preciso per tutte le operazioni interne della ROM, garantendo così un funzionamento coerente e affidabile. Inoltre, sono stati utilizzati dei segnali di abilitazione alla lettura e alla scrittura, in modo da evitare conflitti e da permettere che i dati vengano letti al momento giusto. Si mostra innanzitutto il

nuovo codice di ROM:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity ROM is
6     generic(N: integer range 0 to 32:= 16);
7     port(
8         CLK: in STD_LOGIC; --read sincrona
9         address: in STD_LOGIC_VECTOR(3 downto 0); --l'indirizzo in
10        ↳ ingresso viene dal contatore
11        EN_RD: in STD_LOGIC;
12        dout: out STD_LOGIC_VECTOR(7 downto 0)
13    );
14 end entity ROM;
15
16
17 architecture Behavioral of ROM is
18
19 type MEMORY is array(N-1 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
20    ↳ --memoria da N locazioni che contengono 8 bit
21 constant ROM_N: MEMORY := (
22     "01000000", -- in locazione 15
23     "01000001",
24     "01000010",
25     "01000011",
26     "00010100",
27     "01000101",
28     "00000110",
29     "01000111",
30     "00001000",
31     "00001001",
32     "01001010",
33     "00001011",
34     "00001100",
35     "00001101",
36     "10001010",
37     "00001001" --in locazione 0
38 );
39
40 begin
41     lettura: process(EN_RD, address, CLK)
42     begin

```

```

42      if (CLK'event AND CLK = '1') then
43          if (EN_RD = '1') then
44              dout<= ROM_N(TO_INTEGER(unsigned(address))); --lettura
45                  ↳ dalla rom
46          end if;
47      end if;
48  end process;
49
50
51 end architecture Behavioral;

```

Code 6.1: ROM.vhdl

Si è scelto di utilizzare le stesse stringhe dell'esercizio 2 per "popolare" la ROM, in modo da poter confrontare i risultati. Si mostra ora l'implementazione dell'unità di controllo del sistema; tale codice gestisce i cambiamenti di stato, e si può considerare il "cervello" del sistema in esame.

Si noti che si è scelto di porre in uscita gli stati, in modo da visualizzare in simulazione anche le variazioni di stato, è una scelta ovviamente facoltativa, ma per ragione di debugging è stato scelto di visualizzare anche la variazione di stato, come sarà visibile dalla waveform nella prossima sezione.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity control_unit is
5     Port (
6         START, RST: in STD_LOGIC;
7         CLK: in STD_LOGIC;
8         count: in STD_LOGIC_VECTOR(3 downto 0);
9         stato: out STD_LOGIC_VECTOR(1 downto 0);
10        EN_RD, EN_WR, EN_COUNT: out STD_LOGIC
11    );

```

```

12 end control_unit;
13
14 architecture Behavioral of control_unit is
15 type stati is (idle, read, m_work, write);
16 signal current_state: stati;
17 signal next_state: stati;
18
19 begin
20
21 reg_stato: process(CLK, RST)
22 begin
23     if (CLK'event AND CLK = '1') then
24         if RST = '1' then
25             current_state <= idle;
26         else
27             current_state <= next_state;
28         end if;
29     end if;
30 end process;
31
32 change: process(CLK, START, count)
33 begin
34     CASE current_state is
35         WHEN idle =>
36             EN_COUNT <= '0';
37             if (START = '0') then
38                 next_state <= idle;
39             else
40                 EN_RD <= '1';
41                 next_state <= read;
42             end if;
43         WHEN read =>
44             EN_RD <= '0';
45             EN_COUNT <= '0';
46             next_state <= M_work;
47         WHEN M_work =>
48             EN_WR <= '1';
49             next_state <= write;
50         WHEN write =>
51             EN_WR <= '0';
52             EN_COUNT <= '1';
53             if (count = "1111") then
54                 next_state <= idle;
55             else
56                 EN_RD <= '1';
57                 next_state <= read;

```

```

58         end if;
59     end CASE;
60
61 end process;
62
63 stato <= "00" when current_state = idle else
64     "01" when current_state = read else
65     "10" when current_state = m_work else
66     "11" when current_state = write; -- Associa a ogni stato un
67     → codice binario
68
end Behavioral;
```

Code 6.2: control unit.vhdl

Si mostra infine il codice sistema nel suo complesso, composto dall’unità di controllo e da tutte le altre componenti utilizzate; è stato utilizzato un approccio di tipo strutturale:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Rom_M_MEM is
5 generic(N: integer range 0 to 32:=16);
6 Port (
7     START, RST: in std_logic;
8     CLK: in std_logic;
9     stato: out STD_LOGIC_VECTOR(1 downto 0);
10    count: out STD_LOGIC_VECTOR(3 downto 0);
11    Y: out std_logic_vector(3 downto 0)
12 );
13 end Rom_M_MEM;
14
15 architecture structural of Rom_M_MEM is
16 --segnali intermedi:
17 signal Yc: std_logic_vector(3 downto 0);
18 signal yROM : std_logic_vector(7 downto 0); --uscita della ROM di 8
19     → bit
20 signal YM: std_logic_vector(3 downto 0); --uscita dalla macchina di
21     → trasform di 4 bit
22 signal ENABLE, WRITE, READ: std_logic;
23
24 component ROM is
```

```

23      port (
24        CLK: in STD_LOGIC;
25        address: in STD_LOGIC_VECTOR(3 downto 0);
26        EN_RD: in STD_LOGIC;
27        dout: out STD_LOGIC_VECTOR(7 downto 0)
28      );
29    end component;
30
31 component M is
32   port(
33     ingresso: in std_logic_vector(7 downto 0);
34     uscita: out std_logic_vector(3 downto 0)
35   );
36 end component;
37
38 component MEM
39   port(
40     CLK: in std_logic;
41     EN_WR: in std_logic;
42     ADD: in std_logic_vector(3 downto 0);
43     DATA_IN: in std_logic_vector(3 downto 0)
44   );
45 end component;
46
47 component cont_mod16 is
48   Port (
49     CLK: in std_logic;
50     RST: in std_logic;
51     EN_COUNT: in std_logic;
52     count: out std_logic_vector(3 downto 0)
53   );
54 end component;
55
56 component control_unit
57   Port (
58     START, RST: in STD_LOGIC;
59     CLK: in STD_LOGIC;
60     count: in STD_LOGIC_VECTOR(3 downto 0);
61     stato: out STD_LOGIC_VECTOR(1 downto 0);
62     EN_RD, EN_WR, EN_COUNT: out STD_LOGIC
63   );
64 end component;
65
66 begin
67
68 --collegamenti tra le componenti

```

```

69 ROM0: ROM
70     Port map(
71         CLK => CLK,
72         address => Yc,
73         EN_RD => READ,
74         dout => yROM
75     );
76 M0: M
77     Port map(
78         ingresso => yROM,
79         uscita => yM
80     );
81
82 MEM0: MEM
83     Port map(
84         CLK => CLK,
85         EN_WR => WRITE,
86         ADD => Yc,
87         DATA_IN => yM
88     );
89
90 cont: cont_mod16
91     port map(
92         CLK => CLK,
93         RST => RST,
94         EN_COUNT => ENABLE,
95         count => Yc --l'uscita del contatore deve andare in ingresso ad
96             ← address della ROM e della mem
97         );
98
99 cu: control_unit
100    port map(
101        START => START,
102        RST => RST,
103        CLK => CLK,
104        count => Yc,
105        stato => stato,
106        EN_RD => READ,
107        EN_WR => WRITE,
108        EN_COUNT => ENABLE
109    );
110
111 Y <= yM;
112 count <= Yc;
113

```

---

```
113  end structural;
```

---

Code 6.3: ROM + M + MEM.vhdl

Si vuole porre l'attenzione allo Schematic generato dall'ambiente Vivado, che mostra chiaramente i collegamenti e le dipendenze tra tutte le componenti del sistema.

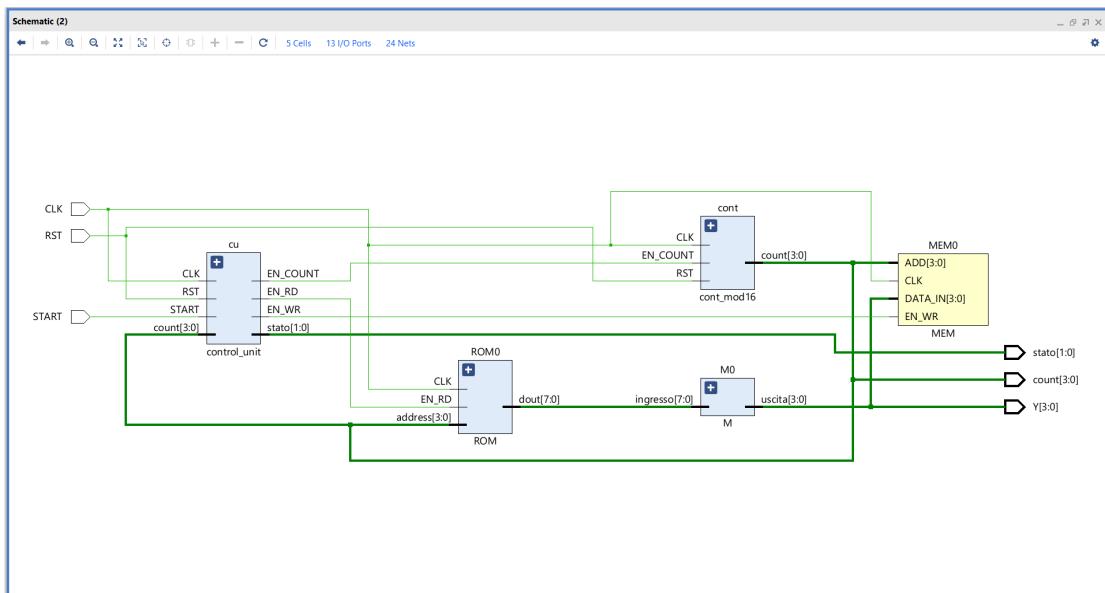


Figura 6.3: Schematic di ROM + M + MEM

#### 6.1.4 Simulazione

Per procedere con la simulazione è stato necessario generare un testbench:

---

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 -- Testbench for Rom_M_MEM entity
6
7 entity tb_Rom_M_MEM is
8 end entity;

```

---

```

9
10 architecture Behavioral of tb_Rom_M_MEM is
11
12   -- Signals for the testbench
13   signal CLK: std_logic := '0';
14   signal RST: std_logic := '1';
15   signal START: std_logic := '0';
16   signal stat0: std_logic_vector(1 downto 0);
17   signal count: STD_LOGIC_VECTOR(3 downto 0);
18   signal Y: std_logic_vector(3 downto 0);
19
20 begin
21
22   -- Clock generation
23   process
24   begin
25     wait for 5 ns;
26     CLK <= not CLK;
27   end process;
28
29   -- Test stimulus
30   process
31   begin
32     wait for 10 ns; -- Wait
33
34     -- Reset the system
35     RST <= '0';
36     wait for 10 ns;
37     RST <= '1';
38     wait for 10 ns;
39     RST <= '0';
40     wait for 10 ns;
41     -- Start the operation
42     START <= '1';
43     wait for 10 ns;
44     START <= '0';
45     wait for 10 ns;
46
47     wait for 100 ns;
48
49     -- End of simulation
50     wait;
51   end process;
52
53   uut: entity work.Rom_M_MEM
54   generic map (N => 16)

```

```

55      port map (
56          CLK => CLK,
57          RST => RST,
58          START => START,
59          stato => stato,
60          count => count,
61          Y => Y
62      );
63
64  end Behavioral;

```

Code 6.4: Testbench di ROM + M + MEM.vhd

Eseguendo la simulazione si avrà la seguente forma d'onda:

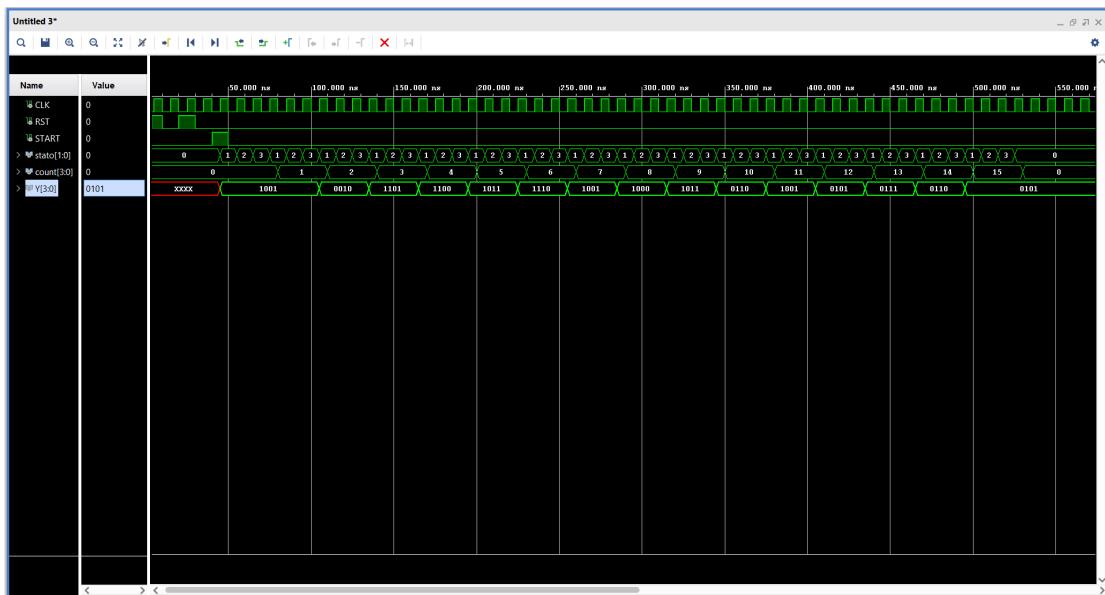


Figura 6.4: Waveform di ROM + M + MEM

Si analizzano alcuni casi per dimostrare la correttezza del sistema.

- **istante 0:** si accede alla locazione di memoria 0, in cui si trova la stringa "00001001", sommando i 4 bit più significativi con i 4 meno significativi si ottiene 1001;
- **istante 1:** si accede alla locazione di memoria 1, in cui si trova la stringa "10001010", all'uscita della macchina M si ottiene 0010;

- **istante 2:** si accede alla locazione di memoria 2, in cui si trova la stringa "00001101", procedendo come in precedenza si ottiene 1101.

Si può procedere in questo modo per tutte le locazioni di memoria scandite dal contatore confermando così il risultato della simulazione. Come detto, si è scelto di mostrare anche le variazioni dello stato e del contatore, in modo da avere la possibilità di osservare in ogni istante il comportamento del sistema, tale scelta è del tutto opzionale.

## 6.2 Implementazione su board del punto precedente

### 6.2.1 Traccia

Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando due bottoni per i segnali di read e reset rispettivamente e i led per la visualizzazione delle uscite della macchina istante per istante.

### 6.2.2 Implementazione

Nel caso in esame, per procedere all'implementazione su board, è stato necessario apportare alcune modifiche: viene infatti richiesto che la lettura sia abilitata da un segnale esterno proveniente da un bottone

della board, mentre nell'implementazione precedente, l'abilitazione alla lettura era un'uscita dell'unità di controllo, che veniva posta alta o bassa in base allo stato corrente. In ogni caso, i codici di ROM, M, counter e MEM restano invariati rispetto al punto precedente, e vengono quindi semplicemente importati.

Si dimostra inoltre necessario l'utilizzo di un divisore di frequenza, componente che si importa dagli esercizi precedenti, per cui non si riporta nuovamente il codice.

Si procede quindi a mostrare le modifiche apportate all'implementazione dell'unità di controllo e di *ROM\_M\_M.vhdl* per la gestione dell'*EN\_RD* come segnale di ingresso dalla board.

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity control_unit is
5   Port (
6     START, RST: in STD_LOGIC;
7     CLK: in STD_LOGIC;
8     EN_RD: in STD_LOGIC;
9     count: in STD_LOGIC_VECTOR(3 downto 0);
10    --  stato: out STD_LOGIC_VECTOR(1 downto 0);
11    EN_WR, EN_COUNT: out STD_LOGIC
12  );
13 end control_unit;
14
15 architecture Behavioral of control_unit is
16 type stati is (idle, read, m_work, write);
17 signal current_state: stati;
18 signal next_state: stati;
19
20 begin
21
22 reg_stato: process(CLK, RST)
23 begin
24   if (CLK'event AND CLK = '1') then
```

```

25      if RST = '1' then
26          current_state <= idle;
27      else
28          current_state <= next_state;
29      end if;
30      end if;
31  end process;

32
33 change: process(CLK, START, count)
34 begin
35     CASE current_state IS
36         WHEN idle =>
37             EN_COUNT <= '0';
38             if (START = '0') then
39                 next_state <= idle;
40             else
41                 if(EN_RD = '1') then
42                     next_state <= read;
43                 else
44                     next_state <= idle;
45                 end if;
46             end if;
47         WHEN read =>
48             EN_COUNT <= '0';
49             next_state <= M_work;
50         WHEN M_work =>
51             EN_WR <= '1';
52             next_state <= write;
53         WHEN write =>
54             EN_WR <= '0';
55             EN_COUNT <= '1';
56             if (count = "1111") then
57                 next_state <= idle;
58             else
59                 if EN_RD = '1' then
60                     next_state <= read;
61                 else
62                     EN_COUNT <= '0';
63                     next_state <= current_state;
64                 end if;
65             end if;
66
67     end CASE;
68
69 end process;
70

```

```

71 --stato <= "00" when current_state = idle else
72 --          "01" when current_state = read else
73 --          "10" when current_state = m_work else
74 --          "11" when current_state = write; -- Associare a ogni
75 --      stato un codice binario
76 end Behavioral;

```

Code 6.5: Control Unit con EN\_RD come ingresso

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Rom_M_MEM is
5 generic(N: integer range 0 to 32:=16);
6 Port (
7     START, RST, READ: in std_logic;
8     CLK: in std_logic;
9     --count: out std_logic_vector(3 downto 0);
10    --stato: out STD_LOGIC_VECTOR(1 downto 0);
11    Y: out std_logic_vector(3 downto 0);
12    led_out: out std_logic
13 );
14 end Rom_M_MEM;
15
16 architecture structural of Rom_M_MEM is
17 --segnali intermedi:
18 signal Yc: std_logic_vector(3 downto 0):= "0000";
19 signal yROM : std_logic_vector(7 downto 0):= "00000000"; --uscita
20 -- della ROM di 8 bit
21 signal yM: std_logic_vector(3 downto 0):="0000"; --uscita dalla
22 -- macchina di trasform di 4 bit
23 signal ENABLE, WRITE: std_logic;
24 signal temp_clock: std_logic;
25
26 component ROM is
27     port (
28         CLK: in STD_LOGIC; --La read e' sincrona
29         address: in STD_LOGIC_VECTOR(3 downto 0); --l'indirizzo in
30         -- ingresso me lo da il contatore
31         EN_RD: in STD_LOGIC;
32         dout: out STD_LOGIC_VECTOR(7 downto 0)
33     );
34 end component;

```

```

32
33 component M is
34     port(
35         ingresso: in std_logic_vector(7 downto 0);
36         uscita: out std_logic_vector(3 downto 0)
37     );
38 end component;
39
40 component MEM
41     port(
42         CLK: in std_logic;
43         EN_WR: in std_logic; --abilitazione alla scrittura nella
        ↳ locazione di memoria indicata da loc
44         ADD: in std_logic_vector(3 downto 0); --l'indirizzo
        ↳ corrispondente all'uscita del contatore cont (mod16)
45         DATA_IN: in std_logic_vector(3 downto 0) --una riga di 4 bit
46     );
47 end component;
48
49 component cont_mod16 is
50     Port (
51         CLK: in std_logic;
52         RST: in std_logic;
53         EN_COUNT: in std_logic;
54         count: out std_logic_vector(3 downto 0)
55     );
56 end component;
57
58 component control_unit
59     Port (
60         START, RST: in STD_LOGIC;
61         CLK: in STD_LOGIC;
62         EN_RD: in STD_LOGIC;
63         count: in STD_LOGIC_VECTOR(3 downto 0);
64         -- stato: out STD_LOGIC_VECTOR(1 downto 0);
65         EN_WR, EN_COUNT: out STD_LOGIC
66     );
67 end component;
68
69 component frequency_divider is
70     Port (
71         clock_in : in STD_LOGIC; -- Clock di ingresso
72         reset : in STD_LOGIC; -- Segnale di reset
73         clock_out : out STD_LOGIC -- Clock di uscita, divisore di
        ↳ frequenza
74     );

```

```
75 end component;
76
77 begin
78
79 --collegamenti tra le componenti
80 ROM0: ROM
81     Port map(
82         CLK => temp_clock,
83         address => Yc,
84         EN_RD => READ,
85         dout => yROM
86     );
87 M0: M
88     Port map(
89         ingresso => yROM,
90         uscita => yM
91     );
92
93 MEM0: MEM
94     Port map(
95         CLK => temp_clock,
96         EN_WR => WRITE,
97         ADD => Yc,
98         DATA_IN => yM
99     );
100
101 cont: cont_mod16
102     port map(
103         CLK => temp_clock,
104         RST => RST,
105         EN_COUNT => ENABLE,
106         count => Yc
107     );
108
109 cu: control_unit
110     port map(
111         START => START,
112         RST => RST,
113         CLK => temp_clock,
114         count => Yc,
115         -- stato => stato,
116         EN_RD => READ,
117         EN_WR => WRITE,
118         EN_COUNT => ENABLE
119     );
120
```

```
121 fd: frequency_divider
122     Port map (
123         clock_in => CLK,    -- Clock di ingresso
124         reset => RST,      -- Segnale di reset
125         clock_out => temp_clock
126     );
127 Y <= yM;
128 --Y <= Yc;
129 led_out <= temp_clock;
130
131 end structural;
```

Code 6.6: ROM + M + MEM su board

Si nota che le uscite commentate nel codice (stato e count) sono state utilizzate ai fini del debugging, per monitorare che le uscite corrispondessero alla giusta locazione di memoria e che i cambiamenti di stato avvenissero in maniera efficiente.

Si è scelto inoltre di mostrare attraverso *led15* della board anche le variazioni del clock, per un’ulteriore conferma visiva della correttezza del sistema.

Come si deduce dal codice mostrato, è stato scelto di usare 3 bottoni per gestire il funzionamento del sistema, *BTNU* come ingresso per *START*, *BTNL* come ingresso per *EN\_RD* e *BNTC* come ingresso per *reset*; per consentire ciò i collegamenti sono stati aggiunti nel file *Nexys - A7 - 50T - Master.xdc*.

Si mostrano ora i riultati ottenuti sulla board, avendo dato l’abilitazione alla lettura in 3 istanti diversi.

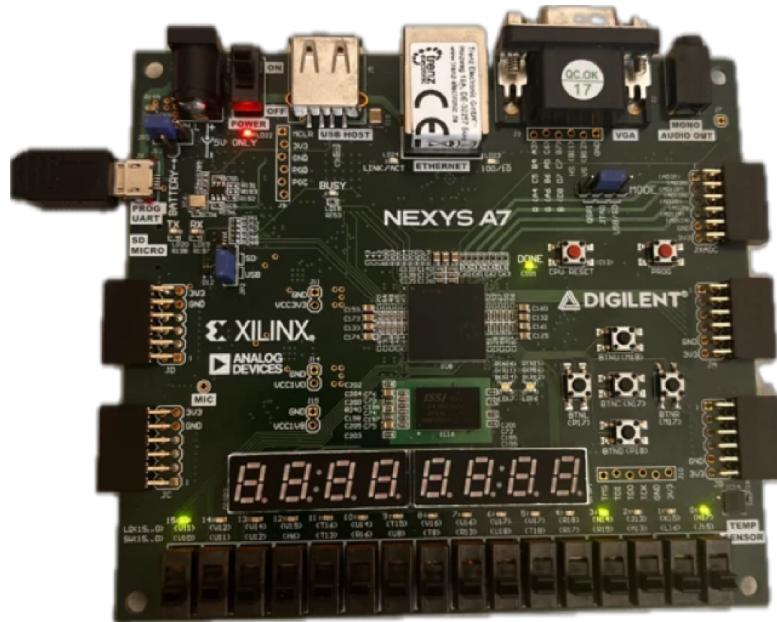


Figura 6.5: Istante 0: uscita = 1001

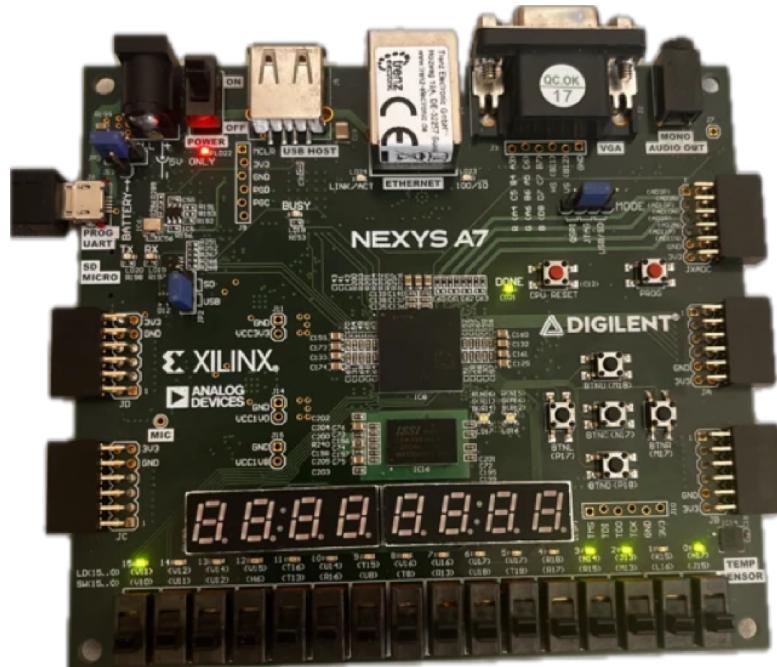


Figura 6.6: Istante 2: uscita = 1101

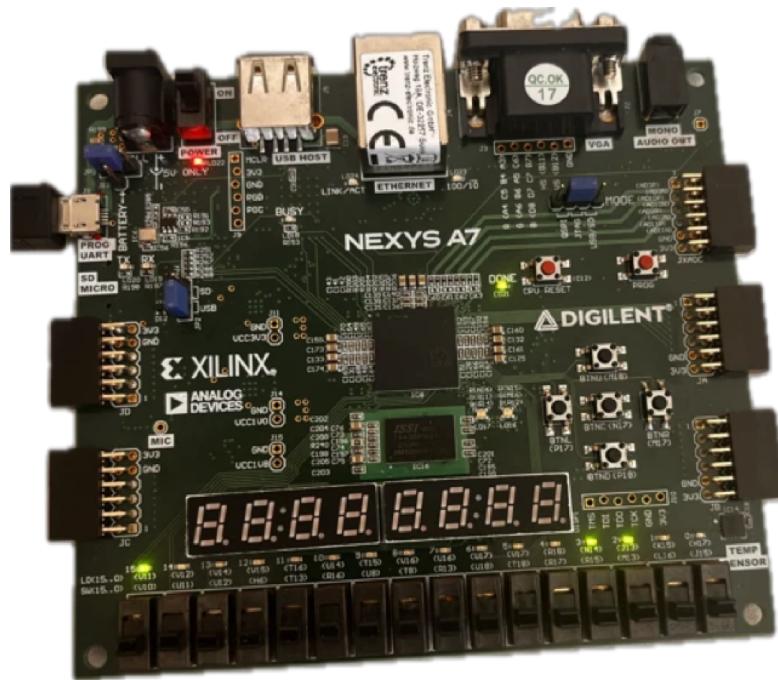


Figura 6.7: Istante 3: uscita = 1100

# Capitolo 7

## Esercizio 7

### 7.1 Moltiplicatore di Booth

Si vuole progettare, implementare in VHDL e simulare il moltiplicatore di Booth, in grado di effettuare il prodotto tra due stringhe di 8 bits ciascuna.

#### 7.1.1 Progettazione

Prima di progettare le componenti necessari, è necessario progettare per interezza il moltiplicatore:

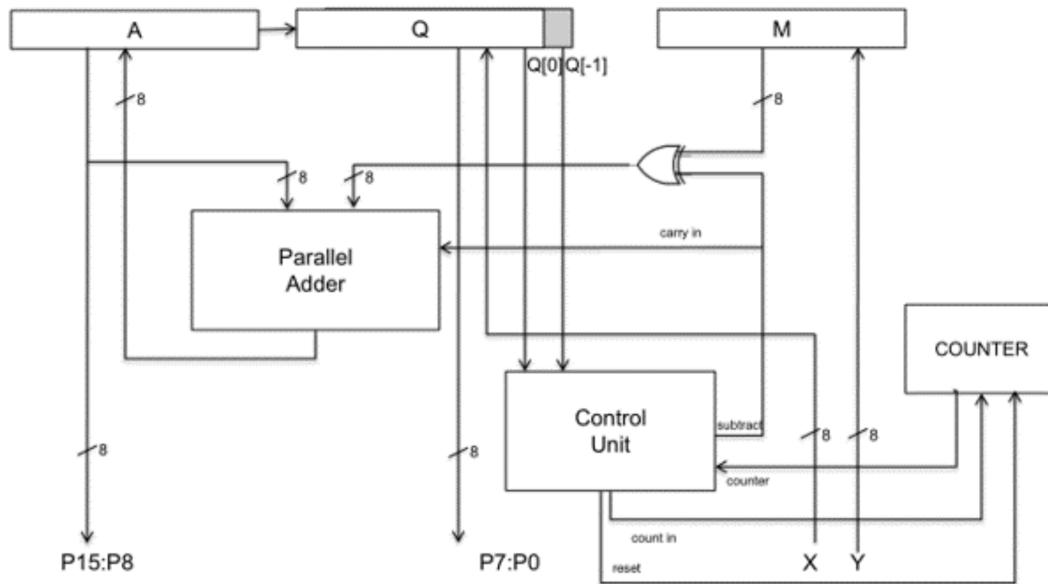


Figura 7.1: Moltiplicatore di Booth

Le componenti necessarie sono quindi:

- **Shift\_register**: uno per il valore  $A$  (Accumulator), uno per il valore  $Q$  e un terzo, il quale memorizzerà il valore  $M$  (quest'ultimo non avrà necessità di fare shift);
- **Flip-Flop D**: necessario per la memorizzazione del valore  $Q_1$  e componente strutturale il contatore;
- **Counter**: contatore che porta il conteggio dei passi effettuati;
- **Parallel Adder**: un moltiplicatore parallelo per effettuare la somma: nell'esempio si utilizza un sommatore Carry-Look-Ahead;
- **Control unit**: unità di controllo per la gestione dell'unità operativa.

## Shift\_Register

Lo Shift\_Register utilizzato differisce da quello visto in precedenza per la presenza di due ingressi aggiuntivi che consentono l'inizializzazione del registro e l'inserimento di un'intera stringa nel registro:

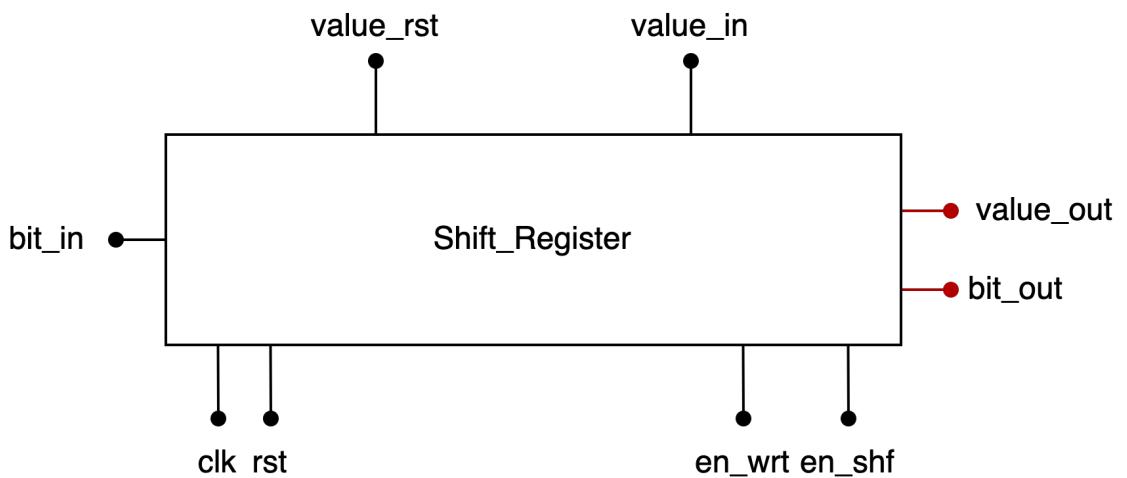


Figura 7.2: Shift\_Register

## Flip-Flop D

Il Flip-Flop D è lo stesso progettato nel capitolo (Citare capitolo). Si deve notare, che alcune delle porte di ingresso del Flip-Flop D rimarranno inutilizzate, poiché in tale progetto non vi è la necessità di effettuare un settaggio iniziale.

## Counter

La macchina counter è un contatore modulo 3.

Analogamente ai contatori presentati nell'esercizio del cronometro, es-

so si ottiene utilizzando 3 Flip-Flop D in parallelo.

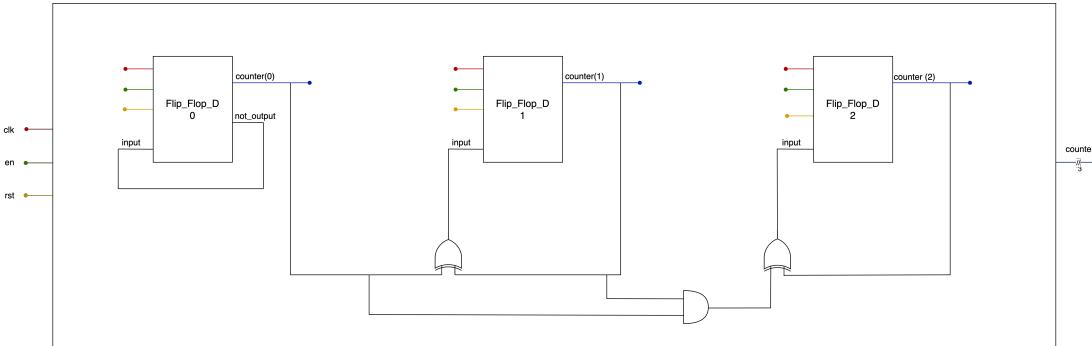


Figura 7.3: Counter mod 8

### Carry-Look-Ahead

Per effettuare la somma, si necessita un sommatore parallelo.

Si vuole utilizzare il Carry-Look-Ahead (è possibile scegliere qualsiasi altro sommatore parallelo) con operandi ad 8 bits.

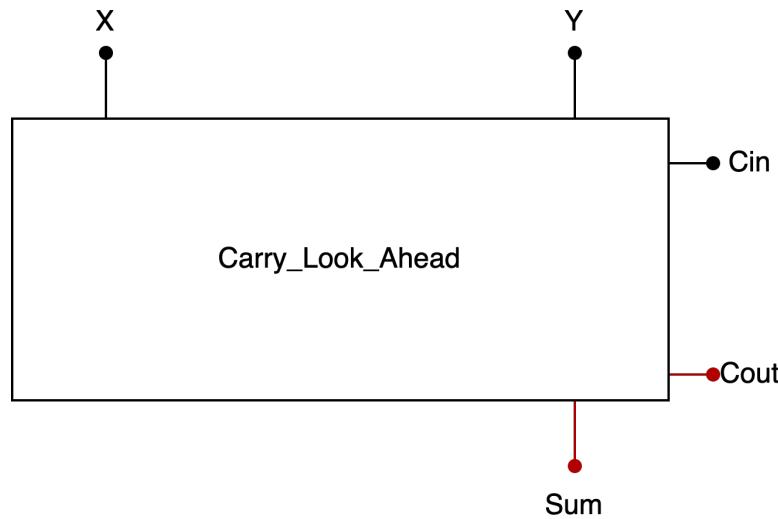


Figura 7.4: Carry Look Ahead

Si noti che all'interno di tale sommatore, viene gestita la scelta di operare un'addizione o una sottrazione.

## Control Unit

Per la gestione delle operazioni, si utilizza una Control Unit.

Essa altro non è che una macchina sequenziale e quindi va progettato l'automa a stati finiti corrispondente:

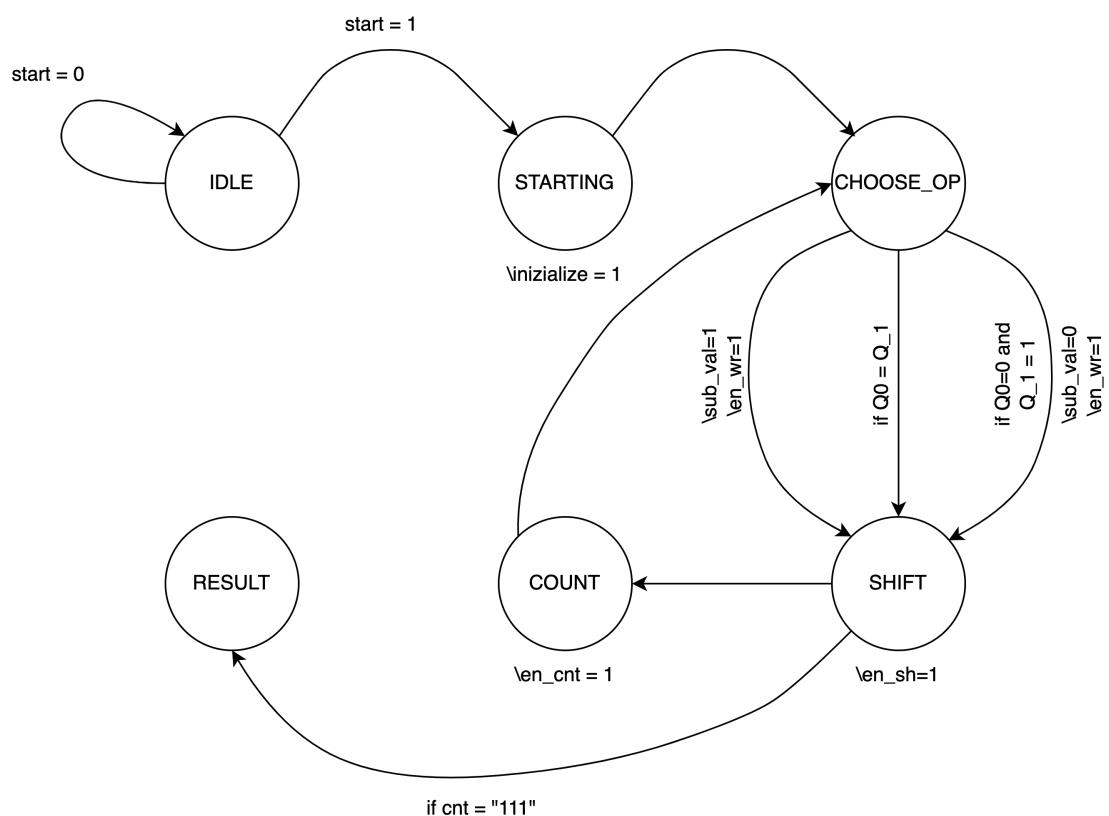


Figura 7.5: Control Unit

### 7.1.2 Implementazione

#### Shift\_Register

L'implementazione dello Shift Register è la seguente:

```

1 library ieee;
2 use      ieee.std_logic_1164.all;
3
  
```

```
4  entity shift_register is
5      port
6      (
7          clk          :  in  std_logic;
8          rst          :  in  std_logic;
9
10         en_wrt       :  in  std_logic;
11         en_shf       :  in  std_logic;
12
13         value_RST    :  in  std_logic_vector(7 downto 0);
14         value_in     :  in  std_logic_vector(7 downto 0);
15
16         bit_in       :  in  std_logic;
17
18         value_out    :  out std_logic_vector(7 downto 0);
19         bit_out      :  out std_logic
20     );
21 end shift_register;
22
23 architecture behavioral of shift_register is
24     signal temp_out      :  std_logic_vector(7 downto 0);
25
26 begin
27     process(clk)
28     begin
29         if (clk'event and clk='1') then
30             if rst = '1' then
31                 temp_out      <=  value_RST;
32
33             else
34                 if en_wrt = '1' then
35                     temp_out      <=  value_in;
36                 elsif en_shf = '1' then
37                     temp_out(6 downto 0)   <= temp_out(7 downto 1);
38                     temp_out(7)           <=  bit_in;
39                 end if;
40             end if;
41         end if;
42         value_out      <=  temp_out;
43         bit_out       <=  temp_out(0);
44     end process;
45
46 end behavioral;
```

Code 7.1: shift\_register.vhdl

## Counter

L'implementazione dello Counter è la seguente:

```
1 library ieee;
2 use      ieee.std_logic_1164.all;
3
4 entity counter_mod_8 is
5   port
6   (
7     clk      : in std_logic;
8     en       : in std_logic;
9     rst      : in std_logic;
10    counter : out std_logic_vector(2 downto 0)
11  );
12 end entity;
13
14 architecture structural of counter_mod_8 is
15   signal temp_counter      : std_logic_vector(2 downto 0)      := 
16   <others => '0';
17   signal back              : std_logic;
18   signal resets            : std_logic_vector(4 downto 0);
19   signal counters          : std_logic_vector(2 downto 0);
20
21
22 component ffD is
23   port
24   (
25     clk      : in std_logic;
26     en       : in std_logic;
27     rst      : in std_logic;
28     preset   : in std_logic;
29     input    : in std_logic;
30     output   : out std_logic;
31     not_output : out std_logic
32   );
33 end component;
34
35 begin
36
37   resets(0)    <= rst;
38
39   counter_0 : ffD
40     port map (
```

```
41         clk,
42         en,
43         resets(0),
44         '0',
45         back,
46         temp_counter(0),
47         back
48     );
49
50     resets(1)    <=  rst;
51     counters(1)  <=  temp_counter(0) xor temp_counter(1);
52
53     counter_1 : ffD
54     port map (
55         clk,
56         en,
57         resets(1),
58         '0',
59         counters(1),
60         temp_counter(1)
61     );
62
63     resets(2)    <=  rst;
64     counters(2)  <=  (temp_counter(0) and temp_counter(1)) xor
65                         ~ temp_counter(2);
66
67     counter_2 : ffD
68     port map (
69         clk,
70         en,
71         resets(2),
72         '0',
73         counters(2),
74         temp_counter(2)
75     );
76
77     counter <= temp_counter;
78 end structural;
```

Code 7.2: counter\_mod\_8.vhdl

L'implementazione del Flip-Flop D è la stessa fatta in precedenza.

## Carry-Look-Ahead

L'implementazione del sommatore, con approccio comportamentale, è la seguente:

```

1 library ieee;
2 use      ieee.std_logic_1164.all;
3
4 entity carry_look_ahead is
5   port
6   (
7     X    :  in  std_logic_vector(7 downto 0);
8     Y    :  in  std_logic_vector(7 downto 0);
9     Cin  :  in  std_logic;
10
11    Sum  :  out std_logic_vector(7 downto 0);
12    Cout:  out std_logic
13  );
14 end entity;
15
16 architecture behavioral of carry_look_ahead is
17   signal G           :  std_logic_vector(7 downto 0);
18   signal P           :  std_logic_vector(7 downto 0);
19
20   signal C           :  std_logic_vector(8 downto 0);
21
22   signal multiplicand :  std_logic_vector(7 downto 0);
23
24 begin
25   process(Cin)
26   begin
27     Y_value: for k in 0 to 7 loop
28       multiplicand(k) <= Y(k) xor Cin;
29     end loop;
30   end process;
31
32   G_P: for k in 0 to 7 generate
33     G(k)      <= X(k) and multiplicand(k);
34     P(k)      <= X(k) xor multiplicand(k);
35   end generate;
36
37   C(0)      <= Cin;
38
39   C_value: for i in 1 to 8 generate

```

```
40      C(i)    <=  G(i-1) or (P(i-1) and C(i-1));
41  end generate;
42
43  sum_value: for i in 0 to 7 generate
44      Sum(i) <= P(i) xor C(i);
45  end generate;
46
47      Cout     <= C(8);
48 end behavioral;
```

---

Code 7.3: carry\_look\_ahead.vhdl

## Control Unit

Lo sviluppo della Control Unit è esposto di seguito:

```
1  library ieee;
2  use      ieee.std_logic_1164.all;
3
4  entity cu is
5      port
6      (
7          clk           : in std_logic;
8          rst           : in std_logic;
9
10         start         : in std_logic;
11
12         Q0            : in std_logic;
13         Q_1           : in std_logic;
14         cnt           : in std_logic_vector(2 downto 0);
15
16         initialize    : out std_logic;
17
18         sub_val       : out std_logic;
19
20         en_wr         : out std_logic;
21         en_sh         : out std_logic;
22         en_cnt        : out std_logic
23     );
24 end entity;
25
26 architecture behavioral of cu is
```

```

27      type      type_state  is  (IDLE, STARTING, CHOOSE_OP, SHIFT, COUNT,
28          →  RESULT);
29
30  begin
31
32      change_state_process: process(clk, rst)
33  begin
34          if (clk'event and clk='1') then
35              if rst = '1' then
36                  current_state    <=  IDLE;
37              else
38                  current_state    <=  next_state;
39              end if;
40          end if;
41      end process;
42
43      sf_machine: process(current_state)
44  begin
45          initialize    <=  '0';
46          en_wr        <=  '0';
47          en_sh        <=  '0';
48          en_cnt       <=  '0';
49
50          case current_state is
51              when IDLE =>
52                  if start = '0' then
53                      next_state    <=  current_state;
54                  else
55                      next_state    <=  STARTING;
56                  end if;
57
58              when STARTING    =>
59                  initialize    <=  '1';
60                  next_state    <=  CHOOSE_OP;
61
62              when CHOOSE_OP =>
63                  if Q0 = Q_1 then
64                      next_state    <=  SHIFT;
65                  else
66                      if Q0 = '1' and Q_1 = '0' then
67                          sub_val      <=  '1';
68                      elsif Q0 = '0' and Q_1 = '1' then
69                          sub_val      <=  '0';
70                      end if;
71                  next_state    <=  SHIFT;
72
73          end case;
74      end process;
75
76      sub_val    <=  sub_val;
77
78      end architecture;

```

```
72         en_wr    <=  '1';
73     end if;
74
75     when SHIFT =>
76         en_sh    <=  '1';
77         if cnt = "111" then
78             next_state  <=  RESULT;
79         else
80             next_state  <=  COUNT;
81         end if;
82
83     when COUNT =>
84         en_cnt   <=  '1';
85         next_state  <=  CHOOSE_OP;
86
87     when RESULT =>
88
89         end case;
90     end process;
91 end behavioral;
```

Code 7.4: cu.vhdl

## Booth

Quindi a questo punto, si può implementare strutturalmente la macchina Booth:

```
1 library ieee;
2 use      ieee.std_logic_1164.all;
3
4 entity booth is
5     port
6     (
7         clk      :  in  std_logic;
8         rst      :  in  std_logic;
9
10        start    :  in  std_logic;
11
12        X       :  in  std_logic_vector(7 downto 0);
13        Y       :  in  std_logic_vector(7 downto 0);
14
```

```

15         res      :  out std_logic_vector(15 downto 0)
16     );
17 end booth;
18
19 architecture structural of booth is
20     component shift_register is
21         port
22         (
23             clk      :  in  std_logic;
24             rst      :  in  std_logic;
25             en_wrt   :  in  std_logic;
26             en_shf   :  in  std_logic;
27             value_rst :  in  std_logic_vector(7 downto 0);
28             value_in  :  in  std_logic_vector(7 downto 0);
29             bit_in    :  in  std_logic;
30             value_out :  out std_logic_vector(7 downto 0);
31             bit_out   :  out std_logic
32         );
33     end component;
34
35     component counter_mod_8 is
36         port
37         (
38             clk      :  in  std_logic;
39             en       :  in  std_logic;
40             rst      :  in  std_logic;
41             counter :  out std_logic_vector(2 downto 0)
42         );
43     end component;
44
45     component ffD is
46         port
47         (
48             clk      :  in  std_logic;
49             en       :  in  std_logic;
50             rst      :  in  std_logic;
51             preset   :  in  std_logic;
52             input    :  in  std_logic;
53             output   :  out std_logic;
54             not_output :  out std_logic
55         );
56     end component;
57
58     component carry_look_ahead is
59         port
60         (

```

```

61      X    :  in  std_logic_vector(7 downto 0);
62      Y    :  in  std_logic_vector(7 downto 0);
63      Cin :  in  std_logic;
64      Sum :  out std_logic_vector(7 downto 0);
65      Cout:  out std_logic
66  );
67 end component;
68
69 component cu is
70   port
71   (
72     clk          :  in  std_logic;
73     rst          :  in  std_logic;
74     start        :  in  std_logic;
75     Q0           :  in  std_logic;
76     Q_1          :  in  std_logic;
77     cnt          :  in  std_logic_vector(2 downto 0);
78     initialize   :  out std_logic;
79     sub_val      :  out std_logic;
80     en_wr        :  out std_logic;
81     en_sh        :  out std_logic;
82     en_cnt       :  out std_logic
83   );
84 end component;
85
86 signal rst_iniz   :  std_logic;
87
88 signal en_wr_temp :  std_logic;
89 signal en_sh_temp :  std_logic;
90
91 signal A_value_in :  std_logic_vector(7 downto 0);
92 signal A_bit_in    :  std_logic;
93 signal A_value_out :  std_logic_vector(7 downto 0);
94 signal A_bit_out   :  std_logic;
95
96 signal Q_bit_in   :  std_logic;
97 signal Q_value_out :  std_logic_vector(7 downto 0);
98 signal Q_bit_out   :  std_logic;
99
100 signal Q_1_bit_in :  std_logic;
101 signal Q_1_bit_out :  std_logic;
102
103 signal M_value_out :  std_logic_vector(7 downto 0);
104
105 signal subst       :  std_logic;
106

```

```

107     signal en_cnt_temp : std_logic;
108     signal count_value : std_logic_vector(2 downto 0);
109
110     signal initialize : std_logic;
111
112 begin
113     rst_iniz      <= initialize;
114
115     A_reg: shift_register
116     port map (clk, rst_iniz, en_wr_temp, en_sh_temp, (others => '0'),
117               ↳ A_value_in, A_value_out(7), A_value_out, A_bit_out);
118
119     Q_bit_in      <= A_bit_out;
120     A_bit_in      <= A_value_out(7);
121
122     Q_reg: shift_register
123     port map (clk, rst_iniz, '0', en_sh_temp, X, (others=>'0'),
124               ↳ A_bit_out, Q_value_out, Q_bit_out);
125
126     Q_minus_one: ffd
127     port map (clk, en_sh_temp, rst_iniz, '0', Q_bit_out, Q_1_bit_out);
128
129     M: shift_register
130     port map (clk, rst_iniz, '0', '0', Y, (others => '0'), '0',
131               ↳ M_value_out);
132
133     adder: carry_look_ahead
134     port map (A_value_out, M_value_out, subst, A_value_in);
135
136     count: counter_mod_8
137     port map (clk, en_cnt_temp, rst_iniz, count_value);
138
139     control_unit: cu
140     port map (clk, rst, start, Q_bit_out, Q_1_bit_out, count_value,
141               ↳ initialize, subst, en_wr_temp, en_sh_temp, en_cnt_temp);
142
143     res <= A_value_out & Q_value_out;
144
145 end structural;

```

Code 7.5: booth.vhdl

### 7.1.3 Simulazione

Per effettuare la simulazione della macchina, è stato implementato il seguente test\_bench:

```
1 -- Testbench created online at:
2 --
3   ↳ https://www.doulos.com/knowhow/perl/vhdl-testbench-creation-using-perl/
4 -- Copyright Doulos Ltd
5
6 library IEEE;
7 use IEEE.Std_logic_1164.all;
8 use IEEE.Numeric_Signed.all;
9
10 entity booth_tb is
11 end;
12
13 architecture bench of booth_tb is
14
15   component booth
16     port
17       (
18         clk      : in std_logic;
19         rst      : in std_logic;
20         start    : in std_logic;
21         X        : in std_logic_vector(7 downto 0);
22         Y        : in std_logic_vector(7 downto 0);
23         res      : out std_logic_vector(15 downto 0)
24       );
25   end component;
26
27   signal clk: std_logic;
28   signal rst: std_logic;
29   signal start: std_logic;
30   signal X: std_logic_vector(7 downto 0);
31   signal Y: std_logic_vector(7 downto 0);
32   signal prod: std_logic_vector(15 downto 0) ;
33
34   constant clk_period : time := 10 ns;
35
36 begin
37   uut: booth port map ( clk      => clk,
38                         rst      => rst,
```

```

39                     start => start,
40                     X      => X,
41                     Y      => Y,
42                     res   => prod);
43
44 clk_process: process
45 begin
46     while true loop
47         clk <= '0';
48         wait for clk_period / 2;
49         clk <= '1';
50         wait for clk_period / 2;
51     end loop;
52 end process;
53
54 stimulus: process
55 begin
56
57     X      <= "00000010";
58     Y      <= "11110000";
59     start  <= '1';
60     -- Put test bench stimulus code here
61
62     wait;
63 end process;
64
65
66 end;

```

Code 7.6: booth\_tb.vhdl

Il risultato è il seguente:

Figura 7.6: Simulazione 1:  $3 \times -1$ Figura 7.7: Simulazione 1:  $-13 \times -16$

## 7.2 Implementazione su board del punto precedente

### 7.2.1 Traccia

Sintetizzare il moltiplicatore implementato al punto 7.1 su FPGA e testarlo mediante l'utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di sviluppo in dotazione. La modalità di utilizzo degli stessi è a completa discrezione degli studenti.

### 7.2.2 Implementazione

Per l'implementazione su board si è scelto di utilizzare gli switches da 0 a 7 per il primo operando e gli switches da 8 a 15 per il secondo operando. Per lo start si è utilizzato il bottone BTNU, mentre per il reset si è utilizzato il bottone centrale BTNC. Per realizzare ciò è stato utilizzato il seguente file *Nexys-A7-50T-Master.xdc* come mostrato:

```
# Clock signal
set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 100000 -waveform {0 5}
[get_ports {clk}];

##Switches
set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 }
[get_ports { X[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 }
[get_ports { X[1] }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 }
[get_ports { X[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
```

```
set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 }
[get_ports { X[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 }
[get_ports { X[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 }
[get_ports { X[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 }
[get_ports { X[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 }
[get_ports { X[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8       IOSTANDARD LVCMOS18 }
[get_ports { Y[0] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8       IOSTANDARD LVCMOS18 }
[get_ports { Y[1] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16      IOSTANDARD LVCMOS33 }
[get_ports { Y[2] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13      IOSTANDARD LVCMOS33 }
[get_ports { Y[3] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6       IOSTANDARD LVCMOS33 }
[get_ports { Y[4] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12      IOSTANDARD LVCMOS33 }
[get_ports { Y[5] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11      IOSTANDARD LVCMOS33 }
[get_ports { Y[6] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10      IOSTANDARD LVCMOS33 }
[get_ports { Y[7] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 }
[get_ports { res[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 }
[get_ports { res[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMOS33 }
```

```

[get_ports { res[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 }
[get_ports { res[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18      IOSTANDARD LVCMOS33 }
[get_ports { res[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17      IOSTANDARD LVCMOS33 }
[get_ports { res[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17      IOSTANDARD LVCMOS33 }
[get_ports { res[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16      IOSTANDARD LVCMOS33 }
[get_ports { res[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16      IOSTANDARD LVCMOS33 }
[get_ports { res[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15      IOSTANDARD LVCMOS33 }
[get_ports { res[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14      IOSTANDARD LVCMOS33 }
[get_ports { res[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16      IOSTANDARD LVCMOS33 }
[get_ports { res[11] }]; #IO_L15N_T2_DQS_DOUT_CS0_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15      IOSTANDARD LVCMOS33 }
[get_ports { res[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14      IOSTANDARD LVCMOS33 }
[get_ports { res[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12      IOSTANDARD LVCMOS33 }
[get_ports { res[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11      IOSTANDARD LVCMOS33 }
[get_ports { res[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##Buttons
#set_property -dict { PACKAGE_PIN C12      IOSTANDARD LVCMOS33 }
[get_ports { reset }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17      IOSTANDARD LVCMOS33 }
[get_ports { rst }]; #IO_L9P_T1_DQS_14 Sch=btnc

```

```

set_property -dict { PACKAGE_PIN M18      IOSTANDARD LVCMOS33 }
[get_ports { start }]; #IO_L4N_T0_D05_14 Sch=btnu
#set_property -dict { PACKAGE_PIN P17      IOSTANDARD LVCMOS33 }
[get_ports { load_i }]; #IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17      IOSTANDARD LVCMOS33 }
[get_ports { load_M }]; #IO_L10N_T1_D15_14 Sch=btnr
#set_property -dict { PACKAGE_PIN P18      IOSTANDARD LVCMOS33 }
[get_ports { BTND }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd

```

Da cui si osserva come sono state collegate le componenti utilizzate.

In seguito si mostrano alcune immagini del funzionamento del moltiplicatore sulla board.

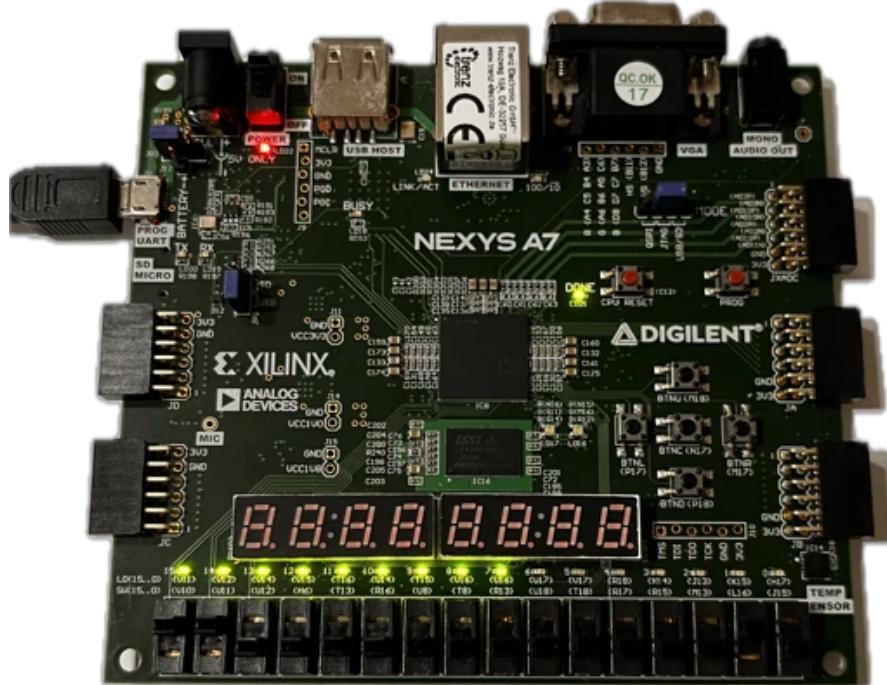


Figura 7.8: Test 1

Nell'immagine rappresentante il Test 1 si è svolta la seguente moltiplicazione:

$$11000000 * 00000010 = 1111111100000000$$

che convertita in decimale risulta

$$-64 * 2 = -128$$

avendo utilizzato la notazione in complemento a 2.

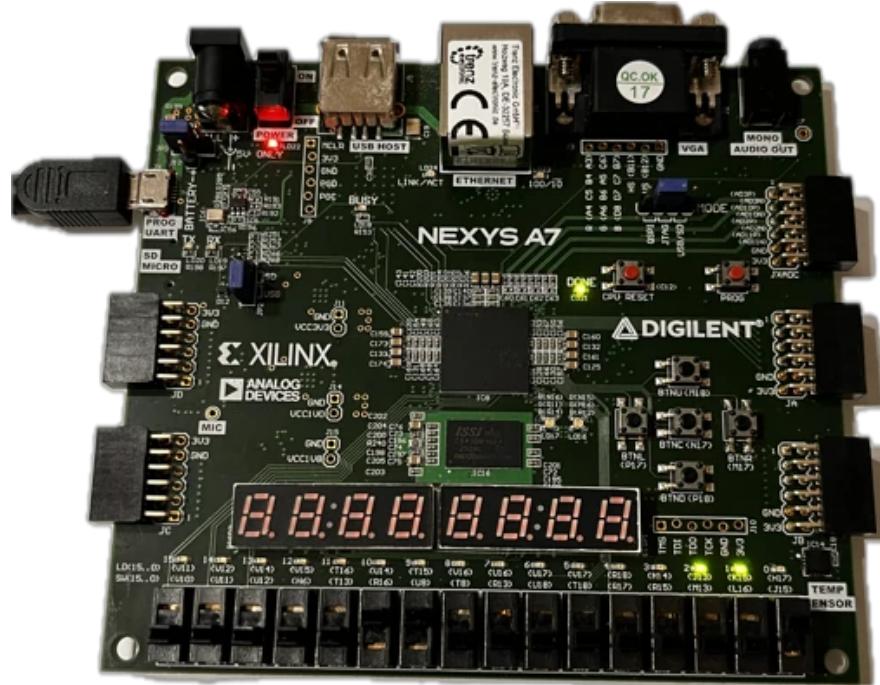


Figura 7.9: Test 2

In Test 2 invece si è svolta la seguente moltiplicazione

$$00000110 * 00000001 = 0000000000000110$$

che convertita in decimale risulta

$$3 * 1 = 3$$

# Capitolo 8

## Esercizio 8.1

### 8.1 Comunicazione con handshaking

#### 8.1.1 Traccia

Progettare, implementare in VHDL e testare mediante simulazione un sistema composto da 2 nodi, A e B, che comunicano mediante un protocollo di handshaking. Il nodo A e il nodo B possiedono entrambi una memoria interna in cui sono memorizzate N stringhe di M bit, denominate  $X(i)$  e  $Y(i)$  rispettivamente ( $i=0,..,N-1$ ). Il nodo A trasmette a B ciascuna stringa  $X(i)$  utilizzando un protocollo di handshaking; B, ricevuta la stringa  $X(i)$ , calcola  $S(i)=X(i)+Y(i)$  e immagazzina la somma in opportune locazioni della propria memoria interna. Per il progetto è possibile considerare una implementazione di tipo comportamentale per effettuare la somma, mentre è necessario prevedere esplicitamente un componente contatore sia nel sistema A sia nel sistema

B per scandire la trasmissione/ricezione delle stringhe e per terminare la comunicazione.

### 8.1.2 Progettazione

Il sistema in esame è composto da due nodi (A e B), che comunicano mediante un protocollo di handshaking per lo scambio di dati e il coordinamento delle operazioni. Entrambi i nodi dispongono di una memoria interna in cui sono memorizzate N stringhe binarie di lunghezza M bit.

Il Nodo A è responsabile della trasmissione delle stringhe  $X(i)$  verso il Nodo B tramite handshaking, che garantisce che i dati siano trasmessi in modo sicuro e sincronizzato. Una volta ricevuta una stringa, il Nodo B esegue una somma binaria con la corrispondente stringa  $Y(i)$  memorizzata nella propria memoria interna, calcolando  $S(i) = X(i) + Y(i)$ . Il risultato della somma  $S(i)$  viene quindi caricato nella memoria del Nodo B. Per la progettazione si inizia studiando gli schemi a blocchi dei due nodi, che vengono mostrati nelle successive figure:

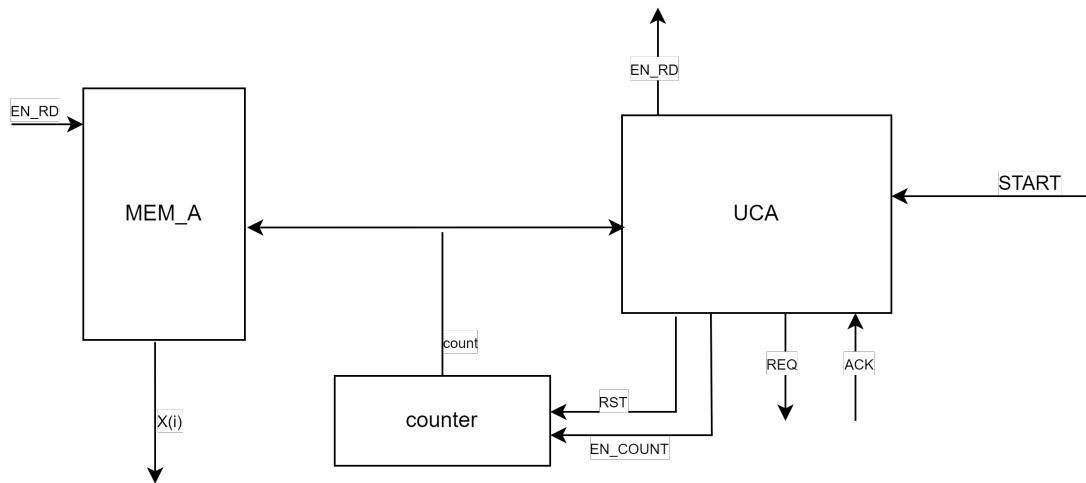


Figura 8.1: Schema a blocchi nodo A

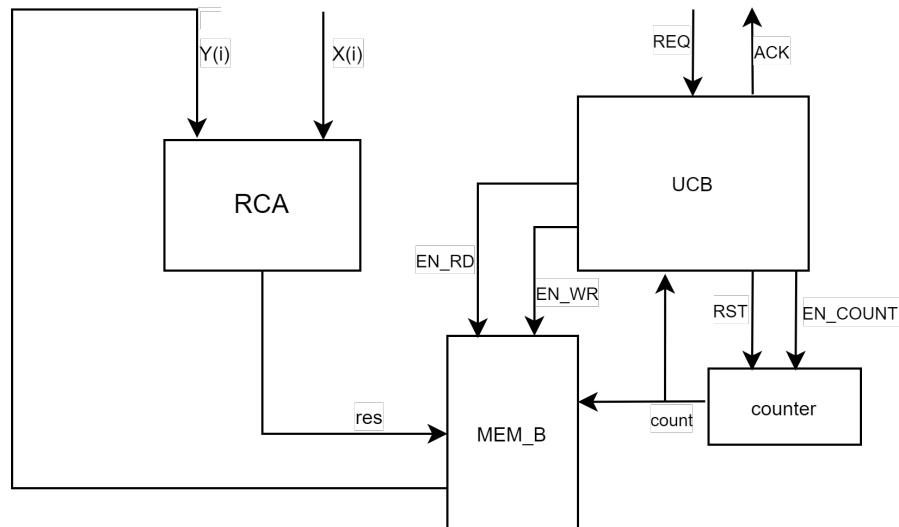


Figura 8.2: Schema a blocchi nodo B

I due sistemi sono dotati ciascuno della propria unità di controllo il cui funzionamento è rappresentato dagli automi mostrati:

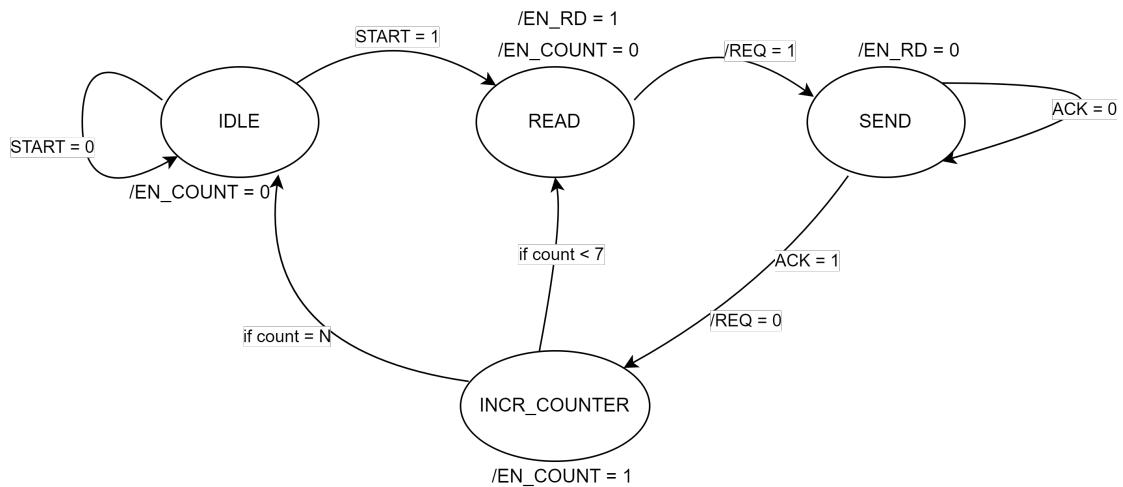


Figura 8.3: Automa del nodo A

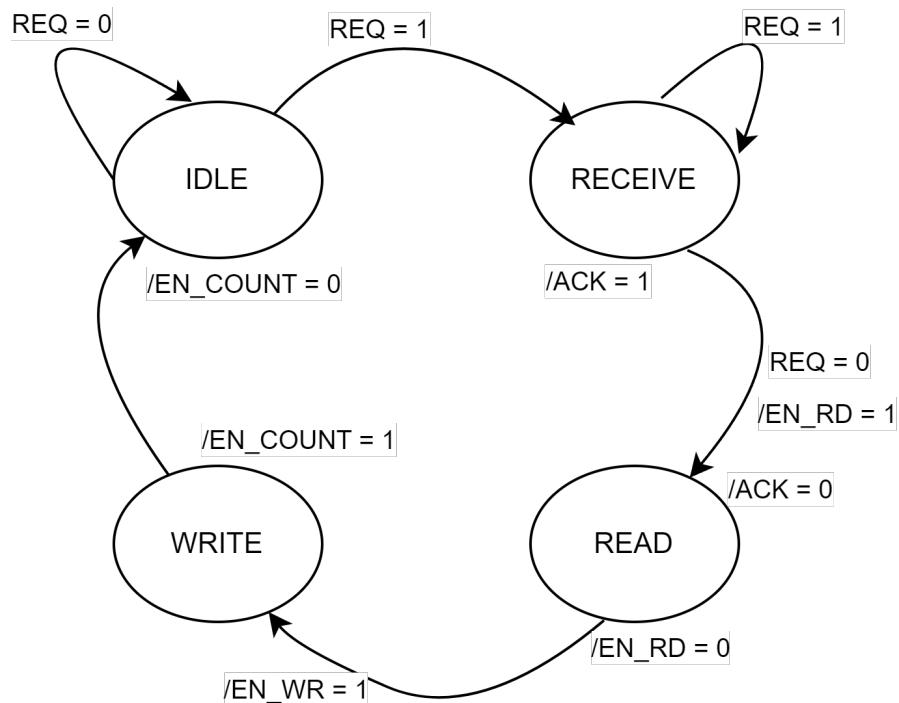


Figura 8.4: Automa del nodo B

### 8.1.3 Implementazione

Per l'implementazione di tale sistema si inizia definendo il nodo A. Come detto, esso si compone di una ROM e un contatore, che insieme

costituiscono l'unità operativa, e una unità di controllo per la gestione.

Si mostrano i codici relativi all'implementazione del nodo A:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity MEM_A is
6     generic(N: integer range 0 to 32:= 16);
7     port(
8         CLK: in STD_LOGIC; --La read incrona
9         address: in STD_LOGIC_VECTOR(3 downto 0); --l'indirizzo in
10        ↳ ingresso me lo da il contatore
11        EN_RD: in STD_LOGIC;
12        dout: out STD_LOGIC_VECTOR(7 downto 0)
13    );
14 end entity MEM_A;
15
16
17 architecture Behavioral of MEM_A is
18
19 type MEMORY is array(N-1 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
20        ↳ --memoria da N locazioni che contengono 8 bit
21 constant ROM_N: MEMORY := (
22     "01000001", -- in locazione 15
23     "01000001",
24     "01000010",
25     "01000011",
26     "00010100",
27     "01000101",
28     "00000110",
29     "01000111",
30     "00001000",
31     "00001001",
32     "01001010",
33     "00001000",
34     "00001100",
35     "00001101",
36     "10001010",
37     "00001001" --in locazione 0
38 );
39
40
41 begin

```

```

40 lettura: process(EN_RD, address, CLK)
41 begin
42     if (CLK'event AND CLK = '1') then
43         if (EN_RD = '1') then
44             dout<= ROM_N(TO_INTEGER(unsigned(address))); --lettura dalla
45             --rom
46         end if;
47     end if;
48 end process;
49
50
51 end architecture Behavioral;

```

Code 8.1: MEM\_A.vhdl

```

1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 use IEEE.NUMERIC_STD.ALL;
6
7 entity counter_modN is
8 generic (N: integer range 0 to 32:= 16);
9     Port ( clock : in STD_LOGIC;
10            reset : in STD_LOGIC;
11            enable : in STD_LOGIC;
12            counter : out STD_LOGIC_VECTOR (3 downto 0));
13 end counter_modN;
14
15 architecture Behavioral of counter_modN is
16
17 signal c : std_logic_vector (3 downto 0) := (others => '0');
18 begin
19 counter <= c;
20
21 counter_process: process(clock)
22 begin
23
24     if(rising_edge(clock)) then
25         if reset = '1' then
26             c <= (others => '0');
27         elsif enable = '1' then
28             c <= std_logic_vector(unsigned(c) + 1);

```

```
29      end if;
30      end if;
31 end process;
32
33 end Behavioral;
```

---

Code 8.2: counter.vhdl

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity unita_operativa_A is
5     Port (
6         CLK, RST: in STD_LOGIC;
7         EN_RD, EN_COUNT: in STD_LOGIC;
8         count: out STD_LOGIC_VECTOR(3 downto 0);
9         X: out STD_LOGIC_VECTOR(7 downto 0)
10    );
11 end unita_operativa_A;
12
13 architecture structural of unita_operativa_A is
14
15 component counter_modN
16     generic (N: integer range 0 to 32:= 16);
17     Port ( clock : in STD_LOGIC;
18             reset : in STD_LOGIC;
19                     enable : in STD_LOGIC;
20                     counter : out STD_LOGIC_VECTOR (3 downto 0));
21 end component;
22
23 component MEM_A
24     generic(N: integer range 0 to 32:= 16);
25     port(
26         CLK: in STD_LOGIC; --La read incrona
27         address: in STD_LOGIC_VECTOR(3 downto 0); --l'indirizzo in
28             ↳ ingresso me lo da il contatore
29         EN_RD: in STD_LOGIC;
30         dout: out STD_LOGIC_VECTOR(7 downto 0)
31     );
32 end component;
33
34 signal temp_c: STD_LOGIC_VECTOR(3 downto 0);
35 begin
```

```

36 counter: counter_modN
37     port map(
38         clock => CLK,
39         reset => RST,
40         enable => EN_COUNT,
41         counter => temp_c
42     );
43
44 mem: MEM_A
45     port map(
46         CLK => CLK,
47         address => temp_c,
48         EN_RD => EN_RD,
49         dout => X
50     );
51 count <= temp_c;
52 end structural;

```

Code 8.3: unità operativa di A in vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity UCA is
5     Port (
6         START, CLK, RST: in STD_LOGIC;
7         ACK: in STD_LOGIC;
8         count: in STD_LOGIC_VECTOR(3 downto 0);
9         REQ: out STD_LOGIC;
10        EN_COUNT, EN_RD: out STD_LOGIC
11        --stato: out STD_LOGIC_VECTOR(1 downto 0)
12    );
13 end UCA;
14
15 architecture Behavioral of UCA is
16
17 type stati is (IDLE, READ, SEND, INCR_COUNTER);
18 signal current_state: stati;
19 signal next_state: stati;
20
21 begin
22 reg_stato: process(CLK, RST)
23 begin
24 if (CLK'event AND CLK = '1') then

```

```

25      if (RST = '1') then
26          current_state <= IDLE;
27      else
28          current_state <= next_state;
29          end if;
30      end if;
31  end process;
32
33 change_stat0: process(START, current_state, ACK, count)
34 begin
35     EN_RD <= '0';
36     EN_COUNT <= '0';
37     REQ <= '0';
38     CASE current_state is
39         when IDLE =>
40             if (START = '1') then
41                 EN_RD <= '1';
42                 next_state <= READ;
43             else
44                 next_state <= current_state;
45             end if;
46         when READ =>
47             EN_RD <= '0';
48             EN_COUNT <= '0';
49             next_state <= SEND;
50         when SEND =>
51             REQ <= '1';
52             if (ACK = '0') then
53                 next_state <= current_state;
54             else
55                 REQ <= '0';
56                 next_state <= INCR_COUNTER;
57             end if;
58         when INCR_COUNTER =>
59             EN_COUNT <= '1';
60             if (count = "1111") then
61                 next_state <= IDLE;
62             else
63                 EN_RD <= '1';
64                 next_state <= READ;
65             end if;
66
67         end CASE;
68     end process;
69
70 --stat0 <= "00" when current_state = IDLE else

```

```

71      --      "01" when current_state = READ else
72      --      "10" when current_state = SEND else
73      --      "11" when current_state = INCR_COUNTER;
74
75
76 end Behavioral;

```

Code 8.4: unità di controllo di A in vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity nodo_A is
5   Port (
6     START: in STD_LOGIC;
7     CLK, RST: in STD_LOGIC;
8     X: out STD_LOGIC_VECTOR(7 downto 0);
9     ACK: in STD_LOGIC;
10    REQ: out STD_LOGIC
11    --      count: out std_logic_vector(3 downto 0);
12    --      stato: out STD_LOGIC_VECTOR(1 downto 0)
13    );
14 end nodo_A;
15
16 architecture structural of nodo_A is
17
18 component unita_operativa_A
19   Port (
20     CLK, RST: in STD_LOGIC;
21     EN_RD, EN_COUNT: in STD_LOGIC;
22     count: out STD_LOGIC_VECTOR(3 downto 0);
23     X: out STD_LOGIC_VECTOR(7 downto 0)
24   );
25 end component;
26
27 component UCA
28   Port (
29     START, CLK, RST: in STD_LOGIC;
30     ACK: in STD_LOGIC;
31     count: in STD_LOGIC_VECTOR(3 downto 0);
32     REQ: out STD_LOGIC;
33     EN_COUNT, EN_RD: out STD_LOGIC
34     --      stato: out STD_LOGIC_VECTOR(1 downto 0)

```

```

35      );
36 end component;
37
38 signal en_rd_temp, en_count_temp: STD_LOGIC;
39 signal temp_count : STD_LOGIC_VECTOR(3 downto 0);
40
41 begin
42 uo: unita_operativa_A
43     port map(
44         CLK => CLK,
45         RST => RST,
46         EN_RD => en_rd_temp,
47         EN_COUNT => en_count_temp,
48         count => temp_count,
49         X => X
50     );
51
52 uc: UCA
53     port map(
54         START => START,
55         CLK => CLK,
56         RST => RST,
57         ACK => ACK,
58         count => temp_count,
59         REQ => REQ,
60         EN_COUNT => en_count_temp,
61         EN_RD => en_rd_temp
62         -- stato => stato
63     );
64 --count <= temp_count;
65 end structural;

```

Code 8.5: nodo A in vhdl

Per quanto riguarda il nodo B, come si vede dallo schema a blocchi, esso è composto da una memoria a cui si accede sia per la lettura che per la scrittura, un contatore e un addizionatore (implementato come Ripple Carry Adder in maniera strutturale a partire da full adder). Il componente *counter* è comune sia al nodo A che al nodo B, quindi non se ne riporterà nuovamente il codice.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity MEM_B is
6     generic(N: integer range 0 to 32:= 16);
7     port(
8         CLK: in STD_LOGIC; --La read incrona
9         address: in STD_LOGIC_VECTOR(3 downto 0);
10        value_in: in STD_LOGIC_VECTOR(7 downto 0);
11        EN_RD: in STD_LOGIC;
12        EN_WR: in STD_LOGIC;
13        dout: out STD_LOGIC_VECTOR(7 downto 0)
14    );
15 end entity MEM_B;
16
17 architecture Behavioral of MEM_B is
18
19 type MEMORY is array(N-1 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
--memoria da N locazioni che contengono 8 bit
20 signal MEM_N: MEMORY := (
21     15 => "00001000", -- in locazione 15
22     14 => "01000001",
23     13 => "01000010",
24     12 => "01000011",
25     11 => "00010100",
26     10 => "01000101",
27     9  => "00000110",
28     8  => "00000000",
29     7  => "00001000",
30     6  => "00000100",
31     5  => "01001010",
32     4  => "00001000",
33     3  => "00001100",
34     2  => "00000001",
35     1  => "00000000",
36     0  => "00000001" -- in locazione 0
37 );
38
39 begin
40
41 lettura: process(EN_RD, address, CLK, value_in)
42 begin
43     if (CLK'event AND CLK = '1') then

```

```

45      if (EN_RD = '1') then
46          dout<= MEM_N(TO_INTEGER(unsigned(address))); --lettura dalla
47          ↗ rom
48      end if;
49      if (EN_WR = '1') then
50          MEM_N(TO_INTEGER(unsigned(address))) <= value_in;
51      end if;
52  end if;
53
54
55
56 end architecture Behavioral;

```

---

Code 8.6: MEM\_B.vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity full_adder is
5     Port (
6         x, y, cin: in std_logic;
7         cout: out std_logic;
8         s: out std_logic
9
10    );
11 end full_adder;
12
13 architecture dataflow of full_adder is
14
15 begin
16     cout <= (x AND y) OR (cin AND (x XOR y));
17     s <= (x XOR y XOR cin);
18
19 end dataflow;

```

---

Code 8.7: full\_adder.vhdl

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity RCA is

```

```

5 generic (N: integer range 1 to 32:= 8);
6   Port (
7     X, Y: in std_logic_vector(N-1 downto 0);
8     cin: in std_logic;
9     cout: out std_logic;
10    sum: out std_logic_vector(N-1 downto 0)
11  );
12 end RCA;
13
14 architecture structural of RCA is
15 signal temp_s: std_logic_vector(N-1 downto 0);
16 signal temp_c: std_logic_vector(N-1 downto 0);
17
18 component full_adder is
19   port(
20     x, y, cin: in std_logic;
21     cout: out std_logic;
22     s: out std_logic
23   );
24 end component;
25
26 begin
27 fa0: full_adder
28   port map(
29     x => X(0),
30     y => Y(0),
31     cin => cin,
32     cout => temp_c(0),
33     s => temp_s(0)
34   );
35
36 faltoN_1: for i in 1 to N-1 generate
37   fa: full_adder
38     port map(
39       x => X(i),
40       y => Y(i),
41       cin => temp_c(i-1),
42       cout => temp_c(i),
43       s => temp_s(i)
44     );
45 end generate;
46
47 sum <= temp_s;
48 cout <= temp_c(N-1);
49

```

---

50   **end structural;**

---

Code 8.8: Ripple Carry Adder (approccio strutturale) in vhdl

Tali componenti insieme vanno a costituire l'unità operativa.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity unita_operativa_B is
5     Port (
6         CLK, RST: in STD_LOGIC;
7         EN_RD, EN_WR, EN_COUNT: in STD_LOGIC;
8         X: in STD_LOGIC_VECTOR(7 downto 0);
9         Y: out std_logic_vector(7 downto 0);
10        count: out STD_LOGIC_VECTOR(3 downto 0);
11        result: out STD_LOGIC_VECTOR(7 downto 0)
12    );
13 end unita_operativa_B;
14
15 architecture structural of unita_operativa_B is
16
17 component RCA
18     generic (N: integer range 1 to 32:= 8);
19     Port (
20         X, Y: in std_logic_vector(N-1 downto 0);
21         cin: in std_logic;
22         cout: out std_logic;
23         sum: out std_logic_vector(N-1 downto 0)
24     );
25 end component;
26
27 component MEM_B
28     generic(N: integer range 0 to 32:= 16);
29     port(
30         CLK: in STD_LOGIC; --La read incrona
31         address: in STD_LOGIC_VECTOR(3 downto 0);
32         value_in: in STD_LOGIC_VECTOR(7 downto 0);
33         EN_RD: in STD_LOGIC;
34         EN_WR: in STD_LOGIC;
35         dout: out STD_LOGIC_VECTOR(7 downto 0)
36     );
37 end component;
38

```

```

39 component counter_modN
40 generic (N: integer range 0 to 32:= 16);
41     Port ( clock : in STD_LOGIC;
42             reset : in STD_LOGIC;
43             enable : in STD_LOGIC;
44             counter : out STD_LOGIC_VECTOR (3 downto 0));
45 end component;
46
47 signal temp_count: STD_LOGIC_VECTOR(3 downto 0);
48 signal temp_sum: STD_LOGIC_VECTOR(7 downto 0);
49 signal operando: STD_LOGIC_VECTOR(7 downto 0);
50 signal riporto: STD_LOGIC;
51
52 begin
53
54 mem: MEM_B
55     port map(
56         CLK => CLK,
57         address => temp_count,
58         value_in => temp_sum,
59         EN_RD => EN_RD,
60         EN_WR => EN_WR,
61         dout => operando
62     );
63
64 adder: RCA
65     port map(
66         X => X,
67         Y => operando,
68         cin => '0',
69         cout => riporto,
70         sum => temp_sum
71     );
72
73 cont: counter_modN
74     port map(
75         clock => CLK,
76         reset => RST,
77         enable => EN_COUNT,
78         counter => temp_count
79     );
80 count <= temp_count;
81 res: process(CLK, EN_WR)
82 begin
83     if (CLK'event AND CLK = '1') then
84         if (EN_WR = '1') then

```

```

85         result <= temp_sum;
86     end if;
87   end if;
88 end process;
89 Y <= operando;
90 end structural;
```

Code 8.9: unità operativa di B in vhdl

Il nodo B è caratterizzato da una sua unità di controllo:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity UCB is
5   Port (
6     CLK, RST: in STD_LOGIC;
7     REQ: in STD_LOGIC;
8     count: in STD_LOGIC_VECTOR(3 downto 0);
9     ACK: out STD_LOGIC;
10    EN_COUNT, EN_RD, EN_WR: out STD_LOGIC
11    --      stat: out STD_LOGIC_VECTOR(2 downto 0)
12  );
13 end UCB;
14
15 architecture Behavioral of UCB is
16
17 type stati is (IDLE, RECEIVE, READ, WRITE);
18 signal current_state: stati;
19 signal next_state: stati;
20
21 begin
22
23 reg_stato: process(CLK, RST)
24 begin
25   if (CLK'event AND CLK = '1') then
26     if (RST = '1') then
27       current_state <= IDLE;
28     else
29       current_state <= next_state;
30     end if;
31   end if;
32 end process;
33
34 change: process(current_state, REQ)
```

```

35 begin
36     CASE current_state is
37         when IDLE =>
38             EN_COUNT <= '0';
39             EN_WR <= '0';
40             ACK <= '0';
41             if REQ = '0' then
42                 next_state <= current_state;
43             else
44                 next_state <= RECEIVE;
45             end if;
46         when RECEIVE =>
47             ACK <= '1';
48             EN_RD <= '1';
49             if REQ = '1' then
50                 next_state <= current_state;
51             else
52                 next_state <= READ;
53             end if;
54         when READ =>
55             ACK <= '0';
56             EN_RD <= '0';
57             next_state <= WRITE;
58         when WRITE =>
59             EN_WR <= '1';
60             EN_COUNT <= '1';
61             next_state <= IDLE;
62     end CASE;
63 end process;
64
65 --stato <= "000" when current_state = IDLE else
66 --      "001" when current_state = RECEIVE else
67 --      "010" when current_state = READ else
68 --      "011" when current_state = WRITE;
69
70
71 end Behavioral;

```

Code 8.10: unità di controllo di B in vhdl

Unità operativa e di controllo costituiscono il nodo nel suo complesso.

---

```

1  library IEEE;

```

```

2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity nodo_B is
5      Port (
6          CLK, RST: in STD_LOGIC;
7          X: in STD_LOGIC_VECTOR(7 downto 0);
8          REQ: in STD_LOGIC;
9          ACK: out STD_LOGIC;
10         --      stato: out STD_LOGIC_VECTOR(2 downto 0);
11         --      count: out std_logic_vector(3 downto 0);
12         Y: out std_logic_vector(7 downto 0);
13         result: out STD_LOGIC_VECTOR(7 downto 0)
14     );
15 end nodo_B;
16
17 architecture structural of nodo_B is
18
19 component unita_operativa_B
20     Port (
21         CLK, RST: in STD_LOGIC;
22         EN_RD, EN_WR, EN_COUNT: in STD_LOGIC;
23         X: in STD_LOGIC_VECTOR(7 downto 0);
24         Y: out std_logic_vector(7 downto 0);
25         count: out STD_LOGIC_VECTOR(3 downto 0);
26         result: out STD_LOGIC_VECTOR(7 downto 0)
27     );
28 end component;
29
30 component UCB
31     Port (
32         CLK, RST: in STD_LOGIC;
33         REQ: in STD_LOGIC;
34         count: in STD_LOGIC_VECTOR(3 downto 0);
35         ACK: out STD_LOGIC;
36         EN_COUNT, EN_RD, EN_WR: out STD_LOGIC
37         --      stato: out STD_LOGIC_VECTOR(2 downto 0)
38     );
39 end component;
40
41 signal temp_en_rd, temp_en_wr, temp_en_count: STD_LOGIC;
42 signal temp_count: STD_LOGIC_VECTOR(3 downto 0);
43
44 begin
45
46     uo: unita_operativa_B
47         port map (

```

```

48      CLK => CLK,
49      RST => RST,
50      EN_RD => temp_en_rd,
51      EN_WR => temp_en_wr,
52      EN_COUNT => temp_en_count,
53      X => X,
54      Y => Y,
55      count => temp_count,
56      result => result
57  );
58
59 UC: UCB
60  port map(
61      CLK => CLK,
62      RST => RST,
63      REQ => REQ,
64      count => temp_count,
65      ACK => ACK,
66      EN_COUNT => temp_en_count,
67      EN_RD => temp_en_rd,
68      EN_WR => temp_en_wr
69      --      stato => stato
70  );
71  --count <= temp_count;
72
73 end structural;

```

Code 8.11: nodo B in vhdl

Il sistema complessivo, è realizzato collegando opportunamente il nodo A e il nodo B, come mostrato in seguito.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity AplusB is
5   Port (
6     CLK_A, CLK_B, RST: in STD_LOGIC;
7     START: in STD_LOGIC;
8     X: out std_logic_vector(7 downto 0);
9     Y: out std_logic_vector(7 downto 0);
10    result: out STD_LOGIC_VECTOR(7 downto 0)
11    --      count_A: out std_logic_vector(3 downto 0);
12    --      count_B: out std_logic_vector(3 downto 0);

```

```

13      --      stato_A: out STD_LOGIC_VECTOR(1 downto 0);
14      --      stato_B: out STD_LOGIC_VECTOR(2 downto 0)
15  );
16 end AplusB;
17
18 architecture structural of AplusB is
19
20 component nodo_A
21 Port (
22     START: in STD_LOGIC;
23     CLK, RST: in STD_LOGIC;
24     X: out STD_LOGIC_VECTOR(7 downto 0);
25     ACK: in STD_LOGIC;
26     REQ: out STD_LOGIC
27     --      count: out std_logic_vector(3 downto 0);
28     --      stato: out STD_LOGIC_VECTOR(1 downto 0)
29 );
30 end component;
31
32 component nodo_B
33 Port (
34     CLK, RST: in STD_LOGIC;
35     X: in STD_LOGIC_VECTOR(7 downto 0);
36     REQ: in STD_LOGIC;
37     ACK: out STD_LOGIC;
38     --      stato: out STD_LOGIC_VECTOR(2 downto 0);
39     --      count: out std_logic_vector(3 downto 0);
40     Y: out std_logic_vector(7 downto 0);
41     result: out STD_LOGIC_VECTOR(7 downto 0)
42 );
43 end component;
44
45 signal temp_X: STD_LOGIC_VECTOR(7 downto 0);
46 signal temp_ACK, temp_REQ: STD_LOGIC;
47 --signal temp_count_A, temp_count_B: STD_LOGIC_VECTOR(3 downto 0);
48 begin
49
50 A: nodo_A
51 port map(
52     START => START,
53     CLK => CLK_A,
54     RST => RST,
55     X => temp_X,
56     ACK => temp_ACK,
57     REQ => temp_REQ
58     --      count => count_A,

```

```

59      --      stato => stato_A
60      );
61
62 B: nodo_B
63     port map(
64         CLK => CLK_B,
65         RST => RST,
66         X => temp_X,
67         REQ => temp_REQ,
68         ACK => temp_ACK,
69         --      stato => stato_B,
70         --      count => count_B,
71         Y => Y,
72         result => result
73     );
74 X <= temp_X;
75 -- count_B <= temp_count_B;
76 end structural;

```

Code 8.12: Sistema complessivo (A più B) in vhdl

Il codice commentato nelle implementazioni mostrate è stato utilizzato ed è utilizzabile in fase di debugging, per verificare che l'handshaking funzioni correttamente, e che tutte le stringhe inviate siano ricevute, sommate e memorizzate in modo coerente.

### 8.1.4 Simulazione

Per la simulazione è necessario utilizzare un testbench.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity tb_AplusB is
6     -- Nessuna porta, e' un testbench
7 end tb_AplusB;
8
9 architecture Behavioral of tb_AplusB is

```

```

10   -- Componente instanziato
11   component AplusB
12     Port (
13       CLK_A, CLK_B, RST: in STD_LOGIC;
14       START: in STD_LOGIC;
15       X: out STD_LOGIC_VECTOR(7 downto 0);
16       Y: out STD_LOGIC_VECTOR(7 downto 0);
17       result: out STD_LOGIC_VECTOR(7 downto 0)
18     );
19   end component;
20
21   -- Segnali interni del testbench
22   signal CLK_A, CLK_B, RST, START: STD_LOGIC := '0';
23   signal X, Y, result: STD_LOGIC_VECTOR(7 downto 0);
24
25   -- Parametri per clock
26   constant CLK_PERIOD_A: time := 10 ns; -- Periodo del clock A
27   constant CLK_PERIOD_B: time := 10 ns; -- Periodo del clock B
28 begin
29   -- Instanza del DUT (Device Under Test)
30   DUT: AplusB
31     Port map(
32       CLK_A => CLK_A,
33       CLK_B => CLK_B,
34       RST => RST,
35       START => START,
36       X => X,
37       Y => Y,
38       result => result
39     );
40
41   -- Generazione del clock A
42   process
43   begin
44     CLK_A <= '0';
45     wait for CLK_PERIOD_A / 2;
46     CLK_A <= '1';
47     wait for CLK_PERIOD_A / 2;
48   end process;
49
50   -- Generazione del clock B
51   process
52   begin
53     CLK_B <= '0';
54     wait for CLK_PERIOD_B / 2;
55     CLK_B <= '1';

```

```
56      wait for CLK_PERIOD_B / 2;
57  end process;

58
59  -- Processo di stimolo
60  process
61  begin
62      -- Reset iniziale
63      RST <= '1';
64      START <= '0';
65      wait for 50 ns; -- Attesa per stabilizzare il reset
66      RST <= '0';
67      wait for 50 ns;

68
69      -- Avvio della somma
70      START <= '1';
71      wait for 20 ns;
72      START <= '0'; -- Disattiva il segnale START

73
74      -- Attendi un po' per consentire al protocollo di completarsi
75      wait for 200 ns;

76
77      -- Termina la simulazione
78      wait;
79  end process;

80
81 end Behavioral;
```

Code 8.13: testbench

Eseguendo la simulazione sull'ambiente di sviluppo Vivado si ottiene la seguente forma d'onda:

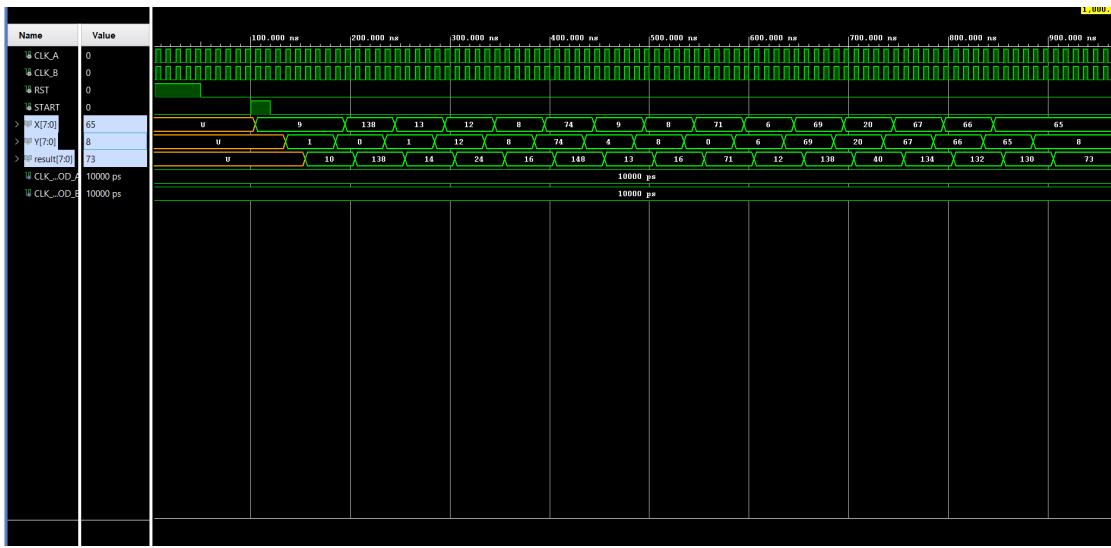


Figura 8.5: Waveform del sistema

Per la conferma del risultato si analizzano le prime locazioni delle memorie di A e B. Si inizia dalla locazione '0':

$$X(0) = 00001001 \text{ che in decimale corrisponde a } 9;$$

$$Y(0) = 00000001 \text{ che in decimale corrisponde a } 1;$$

$$X(0) + Y(0) = 9 + 1 = 10.$$

In locazione '1':  $X(0) = 10001010$  che in decimale corrisponde a 138;

$$Y(0) = 00000000 \text{ che in decimale corrisponde a } 0;$$

$$X(0) + Y(0) = 138 + 0 = 138.$$

In locazione '2':  $X(0) = 00001101$  che in decimale corrisponde a 13;

$$Y(0) = 00000001 \text{ che in decimale corrisponde a } 1;$$

$$X(0) + Y(0) = 13 + 1 = 14.$$

I risultati possono essere considerati coerenti.

# Capitolo 9

## Esercizio 9

Partendo dall'implementazione del processore, operante secondo il modello IJVM, si vuole

- analizzare l'architettura mediante simulazione e fornire e approfondire lo studio per due funzioni;
- modificare un codice operativo, documentando le modifiche effettuate.

### 9.1 Il processore Mic-1

Il processore Mic-1 è un utile esempio didattico con due scopi principali:

1. mostrare come sia possibile realizzare una microarchitettura che implementi un semplice set di istruzioni, usando elementi logici di base;

2. mostrare come la realizzazione di un sistema, apparentemente complesso si riduca in realtà alla progettazione di un'unità operativa e di unità di controllo.

Il set di istruzioni implementato dal Mic-1 è un sottoinsieme di quello della *Java Virtual Machine*, denominato *IJVM*, in quanto opera unicamente su interi.

La particolarità di tale processore è quella di non disporre di registri generali, basando la sua architettura sullo *stack*: infatti le varie istruzioni e logiche non hanno operandi esplicativi, ma sono prelevati secondo la struttura *last-in-first-out*.

L'implementazione di tale processore è detta in *logica microprogrammata*: ciascuna istruzione *IJVM* è implementata come una sequenza di microistruzioni, dette microprocedure; tali sequenze compongono il microprogramma, che è tipicamente memorizzato in una ROM interna al processore.

### 9.1.1 Unità operativa

L'unità operativa del processore è composta da ALU, i suoi ingressi e le sue uscite.

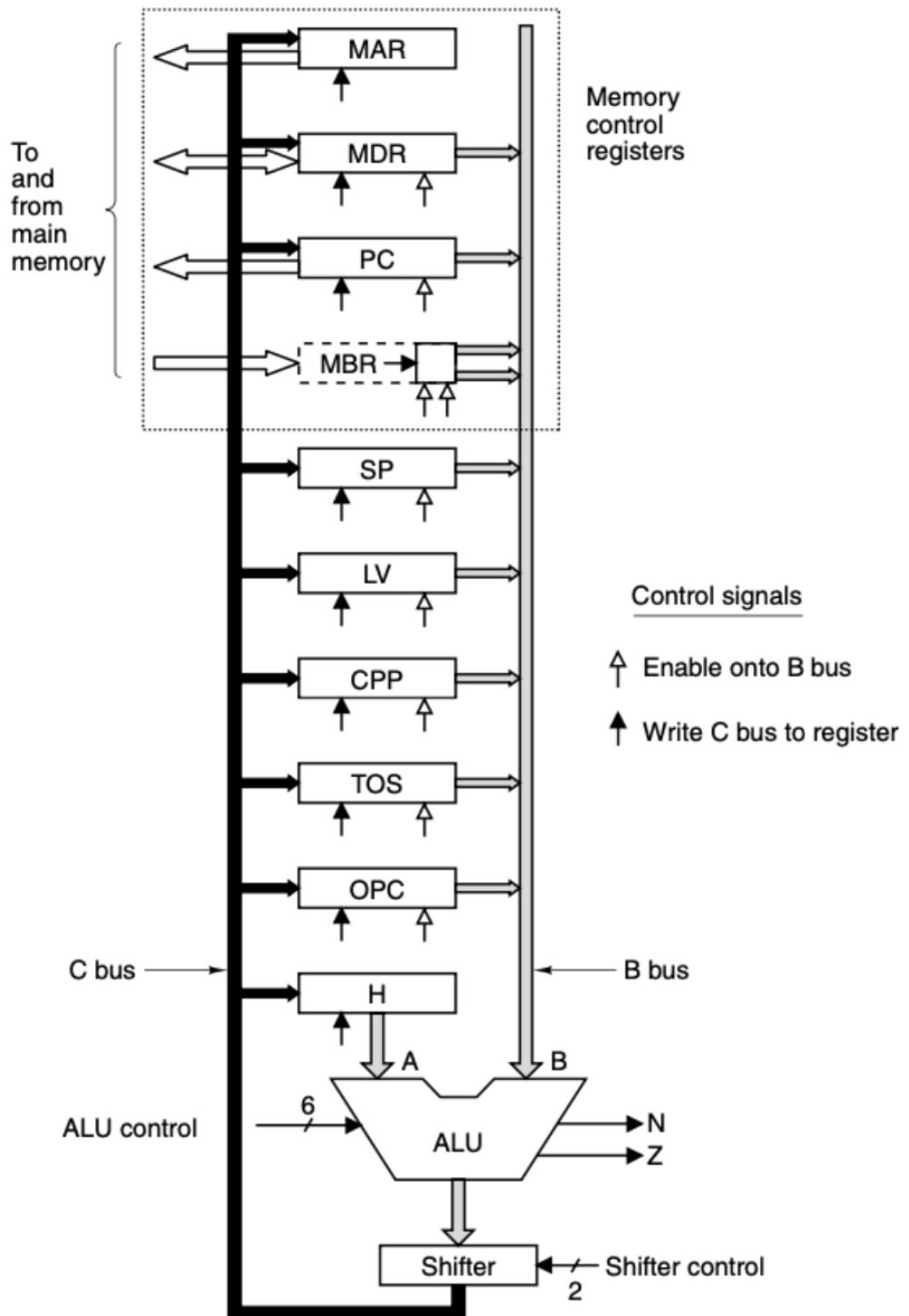


Figura 9.1: Unità operativa del Mic-1

Ogni registro ha dimensione di 32 bit e non è accessibile dal program-

matore, ma solo dal microprogramma.

Vi sono due bus, B e C, collegati rispettivamente al secondo ingresso e all'uscita dell'ALU; il primo ingresso dell'ALU è invece collegato esclusivamente al registro H (*holding*).

Alcuni registri sono cablati in modo da poter essere usati solo per uno scopo specifico:

- registri dell'interfaccia con la memoria
  - MAR: memory address register;
  - MDR: memory data register;
  - PC: program counter;
  - MBR: memory byte register;
- registro che mantiene il primo operando dell'ALU
  - H: holding.

### 9.1.2 Microistruzioni

Per controllare l'unità operativa del processore, sono necessari 29 segnali:

- 9 segnali per controllare la scrittura dei dati dal bus C ai registri;
- 9 segnali per controllare quale registro è collegato al bus B e va in ingresso all'ALU;

- 8 segnali per controllare l'ALU;
- 2 segnali per abilitare lettura/scrittura sull'interfaccia MAR/MDR;
- 1 segnale per abilitare il fetch sull'interfaccia PC/MBR

### 9.1.3 Unità di Controllo

L'unità di controllo del Mic-1 si comporta come un *sequencer*, producendo in ciascun ciclo:

1. lo stato dei segnali di controllo
2. l'indirizzo della prossima microistruzione da eseguire

Di seguito il diagramma completo del Mic-1:

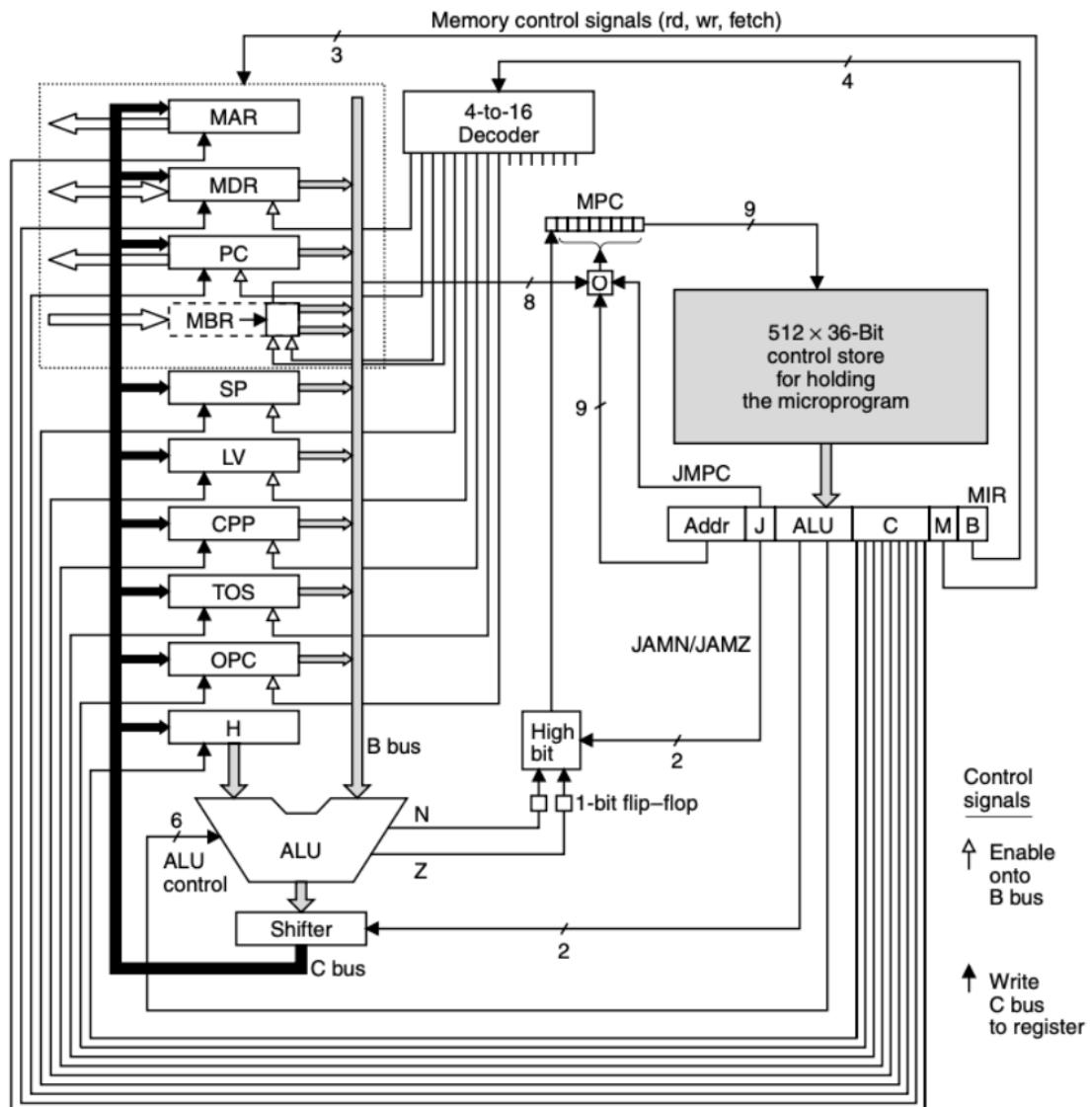


Figura 9.2: Diagramma completo del Mic-1

## 9.2 La microistruzione **ISUB**

La microistruzione **ISUB** altro non è che la sottrazione di due operandi.

Per implementare l'instruzione, viene utilizzato linguaggio MAL, *MicroAssembly Language*, il quale viene convertito dal *microassemblatore*.

L’istruzione è definita come segue

---

```
1     isub = 0x5C:
2         MAR = SP = SP - 1; rd
3         H = TOS
4         MDR = TOS = MDR - H; wr; goto main
```

---

Code 9.1: ISUB

L’indirizzo 0x5C indica l’indirizzo in memoria della prima istruzioni da eseguire. Le restanti sono scritte nei registri successivi.

In tale istruzione si ha come prima istruzione quella di decrementare l’ SP, ovvero lo *stack pointer*, poiché nella fase iniziale punta alla testa dello stack, che può essere richiamata tramite TOS, *Top of Stack*. Inoltre l’indirizzo viene salvato nel MAR, *Memory Address Register* e viene chiamata la funzione di read, rd.

Nella seconda operazione si preleva la testa dello stack tramite TOS e lo si memorizza nel registro H.

Recuperati gli operandi, si effettua l’operazione tra il dato MDR, *Memory Data Register*, e H; si memorizza il risultato con l’istruzione wr nel MDR.

Per caricare l’istruzione nella memoria RAM del processore, si modifica il file `program.ajvm` e lo si compila:

---

```
1      .main
2      .var
3      a
4      .endvar
5      BIPUSH 0xE
6      BIPUSH 0xA
7      ISUB
8      ISTORE a
9      HALT
10     .endmethod
```

---

Code 9.2: ISUB - program.ajvm

dove tramite BIPUSH si caricano gli operandi nello stack, ISUB effettua l'operazione e ISTORE memorizza il risultato nella variabile.

### 9.2.1 Simulazione

Per eseguire la compilazione e quindi modificare il control\_store e la ram, si eseguono i seguenti comandi:

```
antonio@antonio-Standard-PC-Q35-ICH9-2009:~/esercitazione_mic/amic-1$ cmake --build build --target create_control_store
Built target create_control_store
antonio@antonio-Standard-PC-Q35-ICH9-2009:~/esercitazione_mic/amic-1$ cmake --build build --target create_ram
Built target assemble_program
Built target create_ram
antonio@antonio-Standard-PC-Q35-ICH9-2009:~/esercitazione_mic/amic-1$
```

Figura 9.3: Compilazione Control Store e RAM

Un modo per verificare che la compilazione è andata a buon fine, si può controllare come è modificata la RAM:

```
64  -- RAM content
65  signal mem : dp_ar_ram_type := (
66  --BEGIN_WORDS_ENTRY
67 128 => "00000000000000000000000000000000",
68 0 => "0000001000000000000000100000000",
69 1 => "00001010001000000011100010000",
70 2 => "101001100000010011011001011100",
71 3 => "00000000000000000000000000000000",
72 others => (others => '0')
73 --END_WORDS_ENTRY
74 );
75
```

Figura 9.4: Istruzione caricata nella RAM

Si nota infatti che in corrispondenza della istruzione 1 si ha una stringa di 32 bit che altro non è la concatenazione del primo dato, l'indirizzo dell'istruzione BIPUSH, il secondo dato e di nuovo l'indirizzo dell'istruzione BIPUSH. Nell'istruzione 2 si ha l'indirizzo dell'operazione ISUB e quella di ISTORE.

Si può a questo punto effettuare il testbench (ovviamente è stato modificato opportunamente il file `processor_tb.vhd`):

```

antonio@antonio-Standard-PC-Q35-ICH9-2009:~/esercitazione_mic/amic-0$ cmake --build build --target check
Built target index
analyze /home/antonio/esercitazione_mic/amic-0/src/main/vhdl/common_defs.vhd
analyze /home/antonio/esercitazione_mic/amic-0/src/main/vhdl/control_store.vhd
analyze /home/antonio/esercitazione_mic/amic-0/src/main/vhdl/control_unit.vhd
analyze /home/antonio/esercitazione_mic/amic-0/src/main/vhdl/alu.vhd
analyze /home/antonio/esercitazione_mic/amic-0/src/main/vhdl/datapath.vhd
analyze /home/antonio/esercitazione_mic/amic-0/src/main/vhdl/processor.vhd
analyze /home/antonio/esercitazione_mic/amic-0/src/main/vhdl/dp_ar_ran.vhd
analyze /home/antonio/esercitazione_mic/amic-0/src/test/vhdl/processor_tb.vhd
elaborate processor_tb
Built target src.test.vhdl.processor_tb
analyze /home/antonio/esercitazione_mic/amic-0/src/test/vhdl/alu_tb.vhd
elaborate alu_tb
Built target src.test.vhdl.alu_tb
analyze /home/antonio/esercitazione_mic/amic-0/src/test/vhdl/control_unit_tb.vhd
elaborate control_unit_tb
Built target src.test.vhdl.control_unit_tb
analyze /home/antonio/esercitazione_mic/amic-0/src/test/vhdl/datapath_tb.vhd
elaborate datapath_tb
Built target src.test.vhdl.datapath_tb
Test project /home/antonio/esercitazione_mic/amic-0/build
Start 1: src.test.vhdl.alu_tb ..... Passed 0.01 sec
1/4 Test #1: src.test.vhdl.alu_tb ..... Passed 0.01 sec
Start 2: src.test.vhdl.control_unit_tb ..... Passed 0.01 sec
2/4 Test #2: src.test.vhdl.control_unit_tb ..... Passed 0.01 sec
Start 3: src.test.vhdl.datapath_tb ..... Passed 0.02 sec
3/4 Test #3: src.test.vhdl.datapath_tb ..... Passed 0.02 sec
Start 4: src.test.vhdl.processor_tb ..... Passed 0.05 sec
4/4 Test #4: src.test.vhdl.processor_tb ..... Passed 0.05 sec
100% tests passed, 0 tests failed out of 4
Total Test time (real) = 0.10 sec
Built target check

```

Figura 9.5: Testbench ISUB

e questa è la seguente onda risultante

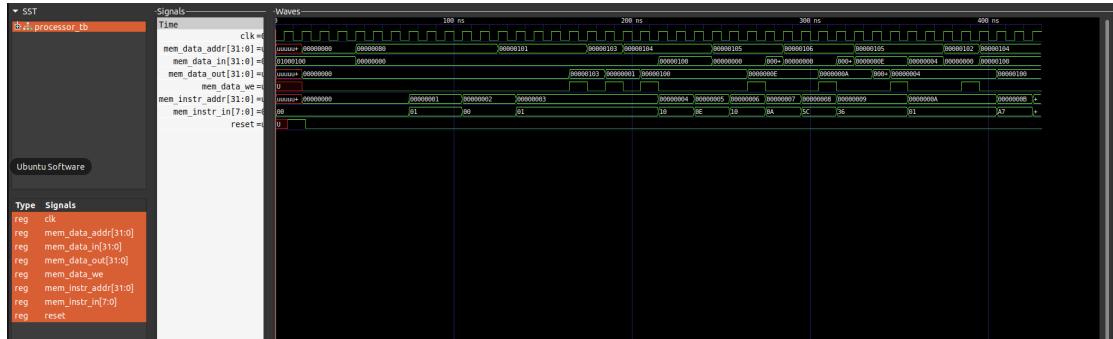


Figura 9.6: Onda ISUB

### 9.3 La microistruzione IOR

La microistruzione IOR effettua l'operazione di *OR* tra i due operandi caricati in memoria.

È definita come segue:

---

```
1     ior = 0xB6:  
2         MAR = SP = SP - 1; rd  
3         H = TOS  
4         MDR = TOS = MDR OR H; wr; goto main
```

---

Code 9.3: IOR

L'address assegnato all'istruzione è 0xB6; viene poi poi memorizzato l'indirizzo del penultimo elemento dello stack, con la funzione `rd`, analogamente al precedente esempio.

Nel registro `H` viene memorizzato `TOS`.

A questo punto è possibile effettuare l'operazione di `OR` tra `MDR` e `H` e con il comando `wr` viene riscritto il registro `MDR` con il risultato.

Si modifica il file `program.ajvm` e lo si compila:

---

```
1     .main  
2     .var  
3     a  
4     .endvar  
5     BIPUSH 0x0  
6     BIPUSH 0x1  
7     IOR  
8     ISTORE a  
9     HALT  
10    .endmethod
```

---

Code 9.4: IOR - program.ajvm

dove tramite BIPUSH si caricano gli operandi nello stack, IOR effettua l'operazione e ISTORE memorizza lo stato nella variabile.

### 9.3.1 Simulazione

Per eseguire la compilazione si eseguono gli stessi comandi in figura 9.3.

Controllando la RAM si puo vedere

```
64  -- RAM content
65  signal mem : dp_ar_ram_type := (
66  --BEGIN_WORDS_ENTRY
67  128 => "0000000000000000000000000000000000000000000000000000000000000000",
68  0 => "0000000100000000000000001000000000000000000000000000000000000000",
69  1 => "0000000100010000000000000000000010000000000000000000000000000000",
70  2 => "10100111000000010011011010110110",
71  3 => "0000000000000000000000000000000000000000000000000000000000000000",
72  others => (others => '0')    I
73  --END_WORDS_ENTRY
74  );
75
```

Figura 9.7: Istruzione caricata nella RAM

dove la differenza rispetto alla precedente immagine della RAM sta nella riga 1, per quanto riguarda gli operandi (ora sono sono 0 e 1), e nella riga 2 per quanto riguarda l'indirizzo della funzione che in questo caso è la IOR.

Si può dunque compilare modificando opportunamente il testbench:

```

antonio@antonio-Standard-PC-Q35-ICH9-2009:~/esercitazione_mtc/mtc$ cmake --build build --target check
Built target index
analyze /home/antonio/esercitazione_mtc/amtc-0/src/math/vhdl/common_defs.vhd
analyze /home/antonio/esercitazione_mtc/amtc-0/src/math/vhdl/control_store.vhd
analyze /home/antonio/esercitazione_mtc/amtc-0/src/math/vhdl/control_unit.vhd
analyze /home/antonio/esercitazione_mtc/amtc-0/src/math/vhdl/alu.vhd
analyze /home/antonio/esercitazione_mtc/amtc-0/src/math/vhdl/datapath.vhd
analyze /home/antonio/esercitazione_mtc/amtc-0/src/math/vhdl/processor.vhd
analyze /home/antonio/esercitazione_mtc/amtc-0/src/math/vhdl/dp_ar_ram.vhd
analyze /home/antonio/esercitazione_mtc/amtc-0/src/test/vhdl/processor_tb.vhd
elaborate processor_tb
Built target src.test.vhdl.processor_tb
analyze /home/antonio/esercitazione_mtc/amtc-0/src/test/vhdl/alu_tb.vhd
elaborate alu_tb
Built target src.test.vhdl.alu_tb
analyze /home/antonio/esercitazione_mtc/amtc-0/src/test/vhdl/control_unit_tb.vhd
elaborate control_unit_tb
Built target src.test.vhdl.datapath_tb
analyze /home/antonio/esercitazione_mtc/amtc-0/src/test/vhdl/datapath_tb.vhd
elaborate datapath_tb
Built target src.test.vhdl.processor_tb
Test project /home/antonio/esercitazione_mtc/amtc-0/build
Start 1: src.test.vhdl.tb..... Passed 0.01 sec
1/4 Test 1: src.test.vhdl.alu_tb..... Passed 0.01 sec
Start 2: src.test.vhdl.control_unit_tb .... Passed 0.02 sec
2/4 Test #2: src.test.vhdl.control_unit_tb .... Passed 0.02 sec
3/4 Test 3: src.test.vhdl.datapath_tb..... Passed 0.02 sec
Start 4: src.test.vhdl.processor_tb..... Passed 0.04 sec
4/4 Test #4: src.test.vhdl.processor_tb ..... Passed 0.04 sec
100% tests passed, 0 tests failed out of 4
Total Test time (real) = 0.08 sec
Built target check
antonio@antonio-Standard-PC-Q35-ICH9-2009:~/esercitazione_mtc/amtc-0$ 
    
```

Figura 9.8: Testbench OR

e questa è la seguente onda risultante

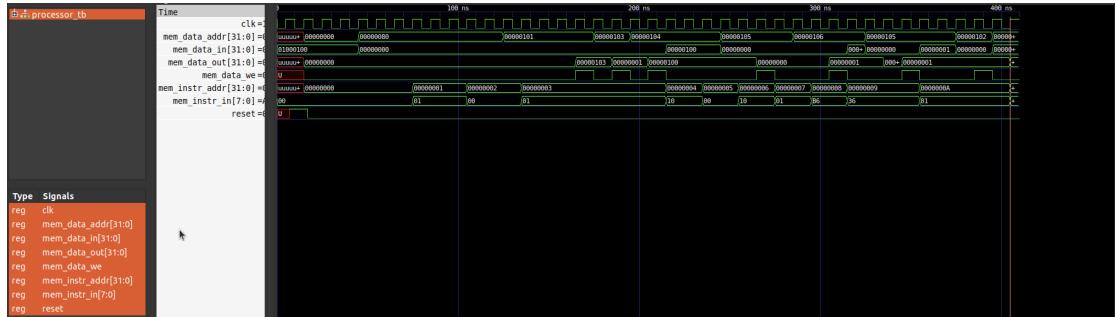


Figura 9.9: Onda IOR

## 9.4 Implementazione della microistruzione IXOR

### IXOR

Si vuole ora implementare la funzione IXOR.

Per far ciò andremo a modificare una funzione già presente nel file `ajvm.mal`, in modo da non andar ad intaccare il codice della build dell'intero sistema.

L'unica accortezza da fare è quella di scegliere una funzione che abbia lo stesso numero di istruzioni di quella che si vuole implementare, op-

pure se avesse un numero inferiore di istruzioni, di controllare se dopo l'ultima istruzione c'è abbastanza spazio libero (lo si può controllare nel file `control_store.vhd`).

In questo caso si sceglie di andar a modificare la funzione `swap` (il nome della funzione non va assolutamente cambiato).

Per semplificare l'implementazione si va a lavorare sull'equazione logica della XOR.

Infatti si può scrivere come segue:

$$\begin{aligned} A \oplus B &= (A \cdot \overline{B}) + (\overline{A} \cdot B) = \\ &= (A \cdot B) + (\overline{A} \cdot \overline{B}) = \\ &= (A \cdot B) \cdot \overline{(A \cdot B)} = \end{aligned}$$

La funzione risulta molto più semplicità e l'implementazione sarà la seguente:

---

```
1     swap = 0x5F:
2         MAR = SP = SP - 1; rd
3         H = TOS
4         OPC = MDR OR H
5         H = H AND MDR
6         H = NOT H
7         MDR = TOS = OPC AND H; wr; goto main
```

---

Code 9.5: IXOR

L'indirizzo in cui è salvata la prima istruzione è 0x5F.

Prima di tutto si effettua una rd del penultimo valore con decrementando SP e ponendolo in MAR; si salva poi in OPC (Registro temporaneo) il risultato della OR tra MDR e H.

Fatto ciò, si memorizza in H, il risultato della AND tra H e MDR, il quale viene negato nell'istruzione successiva.

Nell'ultima istruzione si effettua l'operazione finale di AND tra OPC e H e lo si memorizza in MDR che punta alla testa dello stack. Si modifica il file `program.ajvm` e lo si compila:

---

```
1     .main
2     .var
3     a
4     .endvar
5     BIPUSH 0x0
6     BIPUSH 0x1
7     SWAP
8     ISTORE a
9     HALT
10    .endmethod
```

---

Code 9.6: IOR - program.ajvm

dove tramite BIPUSH si caricano gli operandi nello stack, SWAP effettua l'operazione di XOR e ISTORE memorizza lo stato nella variabile.

Per eseguire la compilazione si eseguono nuovamente gli stessi comandi in figura 9.3.

La RAM è la seguente:

```

64    -- RAM content
65    signal mem : dp_ar_ram_type := (
66      --BEGIN_WORDS_ENTRY
67      128 => "000000000000000000000000000000000000000000000000000000000000000",
68      0 => "000000010000000000000000000000001000000000000000000000000000000",
69      1 => "000000010001000000000000000000001000000000000000000000000000000",
70      2 => "101001110000000010011011001011111",
71      3 => "000000000000000000000000000000000000000000000000000000000000000",
72      others => (others => '0')
73      --END_WORDS_ENTRY
74    );
75
76  begin  -- architecture behavioral
77
```

Figura 9.10: Istruzione caricata nella RAM

dove si può vedere che nella riga 1 vengono caricati gli operandi con la BIPUSH e nella riga 2 viene caricato l'indirizzo della funzione che in questo caso è la SWAP (che effettua la XOR).

Si può dunque compilare modificando opportunamente il testbench:

Figura 9.11: Testbench XOR

e questa è la seguente onda risultante

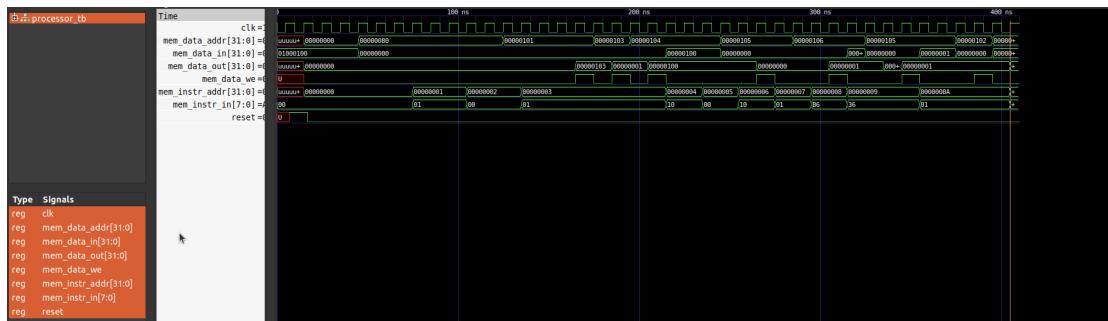


Figura 9.12: Onda XOR

# Capitolo 10

## Esercizio 8

### 10.1 Prova di esame del 19 dicembre 2024

#### 10.1.1 Traccia

Un sistema è composto da 2 nodi, A e B. A include una ROM (progettata come macchina sequenziale con READ sincrono) di 8 locazioni da 4 bit, mentre B include un sommatore parallelo in grado di effettuare la somma di 2 stringhe di 4 bit ciascuna e un registro R di 4 bit. Il sistema opera come segue: all'arrivo di un segnale di start, A inizia a prelevare gli elementi ROM[i] dalla propria memoria e li invia, uno alla volta, a B mediante handshaking. B somma progressivamente le stringhe ricevute utilizzando il sommatore e alla fine inserisce il risultato nel registro R.

### 10.1.2 Progettazione

Il progetto di questo sistema si compone di due nodi fondamentali: il nodo A e il nodo B che comunicano tramite handshaking.

Il nodo A si compone di una memoria di sola lettura (ROM), un contatore e un'unità di controllo che gestisce l'interazione;

il nodo B si compone di un sommatore *Carry Look Ahead*, di un registro e di una unità di controllo per le rispettive operazioni.

Di seguito viene mostrata lo schema a blocchi del sistema, con la suddivisione in due nodi distinti.

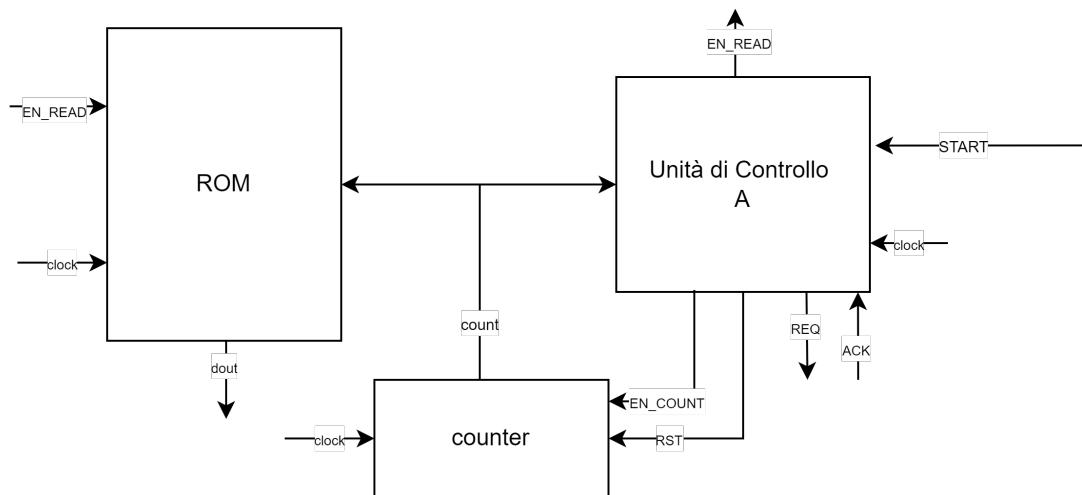


Figura 10.1: Schema a blocchi: nodo A

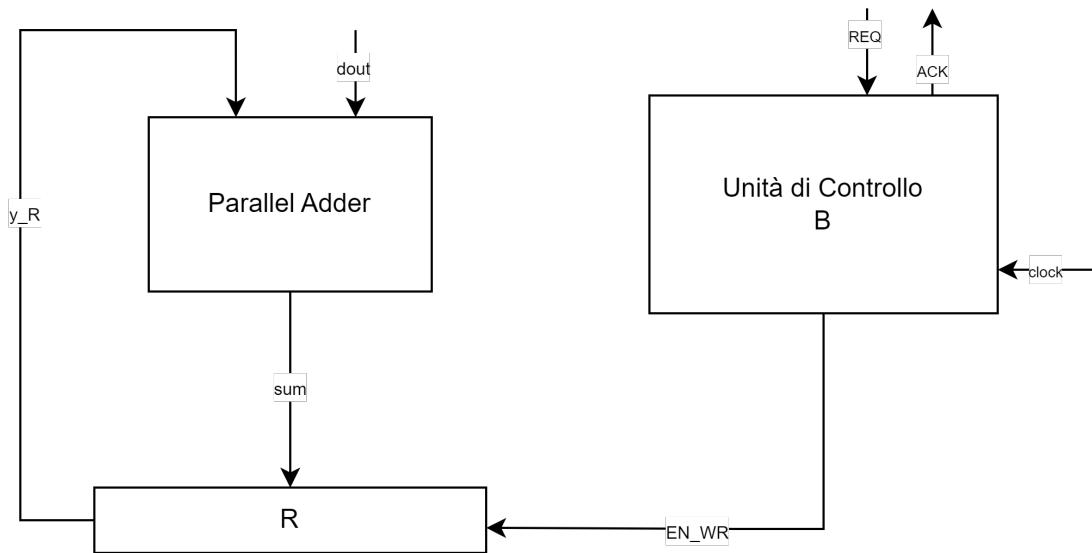


Figura 10.2: Schema a blocchi: nodo B

Nella fase di progettazione, si creano anche gli automi relativi alle unità di controllo dei due nodi, qui mostrati:

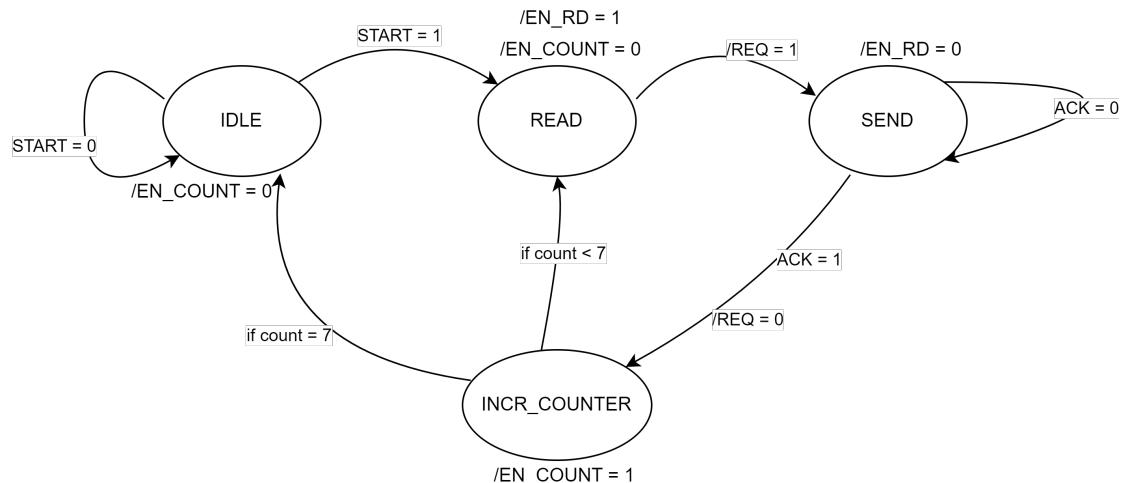


Figura 10.3: Automa unità di controllo: nodo A

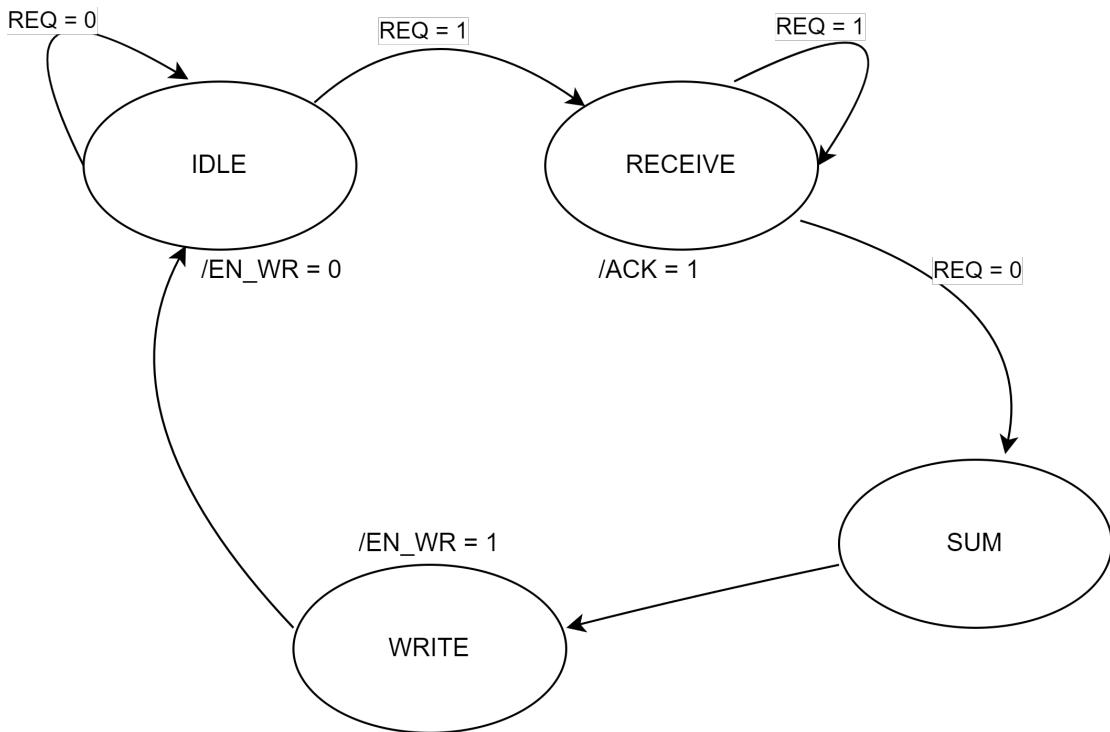


Figura 10.4: Automa unità di controllo: nodo B

### 10.1.3 Implementazione

Per l'implementazione si parte dal nodo A:

partendo dall'unità operativa, essa è composta da una ROM e da un contatore. Si mostrano i rispettivi codici.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity ROM is
6     Port (
7         address: in std_logic_vector(2 downto 0);
8         CLK, EN_RD: in std_logic;
9         dout: out std_logic_vector(3 downto 0)
10    );
11 end ROM;
12
13 architecture Behavioral of ROM is

```

```

14 type MEMO is array(0 to 7) of std_logic_vector(3 downto 0);
15
16 constant ROM: MEMO := (
17     "0000",
18     "1100",
19     "0010",
20     "1010",
21     "0001",
22     "1111",
23     "0101",
24     "0011"
25 );
26
27 begin
28 main: process(CLK)
29 begin
30     if (CLK'event AND CLK = '1') then
31         if (EN_RD = '1') then
32             dout <= ROM(TO_INTEGER(unsigned(address)));
33         end if;
34     end if;
35 end process;
36
37 end Behavioral;

```

Code 10.1: ROM.vhdl

```

1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 use IEEE.NUMERIC_STD.ALL;
6
7 entity counter_mod8 is
8     Port ( clock : in STD_LOGIC;
9             reset : in STD_LOGIC;
10            enable : in STD_LOGIC;
11            counter : out STD_LOGIC_VECTOR (2 downto 0));
12 end counter_mod8;
13
14 architecture Behavioral of counter_mod8 is
15
16 signal c : std_logic_vector (2 downto 0) := (others => '0');
17 begin

```

```
18 counter <= c;
19
20 counter_process: process(clock)
21 begin
22
23     if(rising_edge(clock)) then
24         if reset = '1' then
25             c <= (others => '0');
26         elsif enable = '1' then
27             c <= std_logic_vector(unsigned(c) + 1);
28         end if;
29     end if;
30 end process;
31
32 end Behavioral;
```

---

Code 10.2: Contatore modulo 8.vhdl

Tali componenti vengono collegati tra loro nell'*unità operativa*:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity unita_operativa is
5     Port (
6         CLK, RST, EN_COUNT: in STD_LOGIC;
7         READ: in STD_LOGIC;
8         count: out STD_LOGIC_VECTOR(2 downto 0);
9         dout: out STD_LOGIC_VECTOR(3 downto 0)
10    );
11 end unita_operativa;
12
13 architecture Behavioral of unita_operativa is
14
15 component counter_mod8
16     Port (
17         clock : in STD_LOGIC;
18         reset : in STD_LOGIC;
19             enable : in STD_LOGIC;
20         counter : out STD_LOGIC_VECTOR (2 downto 0));
21 end component;
22
23 component ROM
```

---

```

25 Port (
26     address: in std_logic_vector(2 downto 0);
27     CLK, EN_RD: in std_logic;
28     dout: out std_logic_vector(3 downto 0)
29 );
30 end component;
31
32 signal temp_count: STD_LOGIC_VECTOR(2 downto 0);
33
34 begin
35
36 counter: counter_mod8
37     port map(
38         clock => CLK,
39         reset => RST,
40         enable => EN_COUNT,
41         counter => temp_count
42     );
43
44 mem: ROM
45     port map(
46         address => temp_count,
47         CLK => CLK,
48         EN_RD => READ,
49         dout => dout
50     );
51
52 count <= temp_count;
53 end Behavioral;

```

Code 10.3: CUnità operativa di A in vhdl

Per la gestione delle abilitazioni, si utilizza un'unità di controllo, come si vede dallo schema a blocchi nel paragrafo precedente:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity UCA is
5     Port (
6         START: in STD_LOGIC;
7         CLK, RST: in STD_LOGIC;
8         ACK: in STD_LOGIC;
9         count: in STD_LOGIC_VECTOR(2 downto 0);

```

```

10      REQ: out STD_LOGIC;
11      state: out STD_LOGIC_VECTOR(1 downto 0);
12      EN_RD, EN_COUNT: out STD_LOGIC
13
14  );
15 end UCA;
16
17 architecture Behavioral of UCA is
18 type stati is (IDLE, READ, SEND, INCR_COUNT);
19
20 signal current_state: stati;
21 signal next_state: stati;
22
23 begin
24
25 reg_stato: process(CLK, RST)
26 begin
27     if (CLK'event AND CLK = '1') then
28         if (RST = '1') then
29             current_state <= IDLE;
30         else
31             current_state <= next_state;
32         end if;
33     end if;
34
35 end process;
36
37 change: process(current_state, START, ACK, count)
38 begin
39     EN_RD <= '0';
40     EN_COUNT <= '0';
41     REQ <= '0';
42     CASE current_state is
43         when IDLE =>
44             EN_COUNT <= '0';
45             if (START = '1') then
46                 EN_RD <= '1';
47                 next_state <= READ;
48             else
49                 next_state <= current_state;
50             end if;
51         when READ =>
52             EN_COUNT <= '0';
53             EN_RD <= '0';
54
55             next_state <= SEND;

```

```

56
57     when SEND =>
58         REQ <= '1';
59         if (ACK = '0') then
60             next_state <= current_state;
61         else
62             REQ <= '0';
63
64             next_state <= INCR_COUNT;
65         end if;
66
67
68     when INCR_COUNT =>
69         EN_COUNT <= '1';
70         if count = "111" then
71             next_state <= IDLE;
72         else
73             EN_RD <= '1';
74             next_state <= READ;
75         end if;
76     end CASE;
77 end process;
78 state <= "00" when current_state = IDLE else
79     "01" when current_state = READ else
80     "10" when current_state = SEND else
81     "11" when current_state = INCR_COUNT;
82 end Behavioral;

```

Code 10.4: Control Unit di A.vhdl

Il nodo A nel suo complesso sarà implementato in questo modo:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity A is
5     Port (
6         CLK, RST: in STD_LOGIC;
7         START, ACK: in STD_LOGIC;
8         REQ: out STD_LOGIC;
9         state: out STD_LOGIC_VECTOR(1 downto 0);
10        count: out STD_LOGIC_VECTOR(2 downto 0);
11        dout: out STD_LOGIC_VECTOR(3 downto 0)
12    );
13 end A;
14

```

```

15 architecture structural of A is
16
17 component unita_operativa
18   Port (
19     CLK, RST, EN_COUNT: in STD_LOGIC;
20     READ: in STD_LOGIC;
21     count: out STD_LOGIC_VECTOR(2 downto 0);
22     dout: out STD_LOGIC_VECTOR(3 downto 0)
23   );
24 end component;
25
26 component UCA
27   Port (
28     START: in STD_LOGIC;
29     CLK, RST: in STD_LOGIC;
30     ACK: in STD_LOGIC;
31     count: in STD_LOGIC_VECTOR(2 downto 0);
32     REQ: out STD_LOGIC;
33     state: out STD_LOGIC_VECTOR(1 downto 0);
34     EN_RD, EN_COUNT: out STD_LOGIC
35   );
36 end component;
37
38 signal temp_count: STD_LOGIC_VECTOR(2 downto 0);
39 signal temp_RD, temp_enable: STD_LOGIC;
40 begin
41   uo: unita_operativa
42     port map(
43       CLK => CLK,
44       RST => RST,
45       EN_COUNT => temp_enable,
46       READ => temp_RD,
47       count => temp_count,
48       dout => dout
49     );
50
51   uc: UCA
52     port map(
53       START => START,
54       CLK => CLK,
55       RST => RST,
56       ACK => ACK,
57       count => temp_count,
58       REQ => REQ,
59       state => state,
60       EN_RD => temp_RD,

```

```
61      EN_COUNT => temp_enable  
62  );  
63  
64 count <= temp_count;  
65  
66 end structural;
```

---

Code 10.5: nodo A.vhdl

Si procede ora con l'implementazione del nodo B, le cui componenti sono un registro R per lo storage del risultato e un Carry Look Ahead:

```
1  library IEEE;  
2  use IEEE.STD_LOGIC_1164.ALL;  
3  --registro parallelo-parallelo che mantiene il valore del  
4  --→ moltiplicando Y  
4  entity registro4 is  
5      port( A: in std_logic_vector(3 downto 0);  
6          clk, res, load: in std_logic;  
7          B: out std_logic_vector(3 downto 0));  
8  end registro4;  
9  
10 architecture behavioural of registro4 is  
11     signal temp_b: std_logic_vector(3 downto 0);  
12     begin  
13  
14     R_PP: process(clk, load)  
15         begin  
16             if(clk'event and clk='1') then  
17                 if(res='1') then  
18                     temp_b<= (others=>'0');  
19                 else  
20                     if(load='1') then  
21                         temp_b<=A;  
22                     end if;  
23                 end if;  
24             end if;  
25         end process;  
26         B<=temp_b;  
27     end behavioural;  
28
```

---

Code 10.6: register.vhdl

Il sommatore è stato realizzato con un approccio strutturale, a partire da full adder:

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity full_adder is
5     Port (
6         x, y, cin: in std_logic;
7         cout: out std_logic;
8         s: out std_logic
9
10    );
11 end full_adder;
12
13 architecture dataflow of full_adder is
14
15 begin
16     cout <= (x AND y) OR (cin AND (x XOR y));
17     s <= (x XOR y XOR cin);
18
19 end dataflow;
```

---

Code 10.7: full\_adder.vhdl

---

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CarryLookAhead is
5     generic (N: integer range 1 to 32:= 4);
6     Port (
7         X, Y: in std_logic_vector(N-1 downto 0);
8         cin: in std_logic;
9         cout: out std_logic;
10        sum : out std_logic_vector(N-1 downto 0)
11    );
12 end CarryLookAhead;
13
14 architecture Behavioral of CarryLookAhead is
```

---

```

15 signal c: std_logic_vector(N-1 downto 0);
16 signal temp_sum: std_logic_vector(N-1 downto 0);
17
18 component full_adder is
19 Port (
20     x, y, cin: in std_logic;
21     cout: out std_logic;
22     s: out std_logic
23 );
24 );
25 end component;
26
27 begin
28 c(0) <= cin;
29
30 cin1toN_1: for i in 1 to N-1 generate
31     c(i) <= (X(i-1) and Y(i-1)) OR (c(i-1) and (X(i-1) XOR Y(i-1)));
32 end generate;
33
34 fa0: full_adder
35     port map(
36         x => X(0),
37         y => Y(0),
38         cin => c(0),
39         cout => open,
40         s => temp_sum(0)
41     );
42
43 faltoN_2: for i in 1 to N-2 generate
44     fa: full_adder
45         port map(
46             x => X(i),
47             y => Y(i),
48             cin => c(i),
49             cout => open,
50             s => temp_sum(i)
51         );
52 end generate;
53
54 faN_1: full_adder
55     port map(
56         x => X(N-1),
57         y => Y(N-1),
58         cin => c(N-1),
59         cout => cout,
60         s => temp_sum(N-1)

```

```
61      );
62
63
64 sum <= temp_sum;
65
66
67 end Behavioral;
```

---

Code 10.8: CarryLookAhead.vhdl

Si mostra ora il codice dell'unità operativa:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity unita_operativaB is
5   Port (
6     CLK, RST: in STD_LOGIC;
7     selA: in STD_LOGIC;
8     din: in STD_LOGIC_VECTOR(3 downto 0);
9     load_A: in STD_LOGIC;
10    result: out STD_LOGIC_VECTOR(3 downto 0)
11
12  );
13 end unita_operativaB;
14
15 architecture Behavioral of unita_operativaB is
16
17 component CarryLookAhead
18   generic (N: integer range 1 to 32:= 4);
19   Port (
20     X, Y: in std_logic_vector(N-1 downto 0);
21     cin: in std_logic;
22     cout: out std_logic;
23     sum : out std_logic_vector(N-1 downto 0)
24   );
25 end component;
26
27 component registro4
28   port(
29     A: in std_logic_vector(3 downto 0);
30     clk, res, load: in std_logic;
31     B: out std_logic_vector(3 downto 0));
32 end component;
```

```

33
34 component mux2_1
35 port( x0, x1: in std_logic_vector(3 downto 0);
36           s: in std_logic;
37           y: out std_logic_vector(3 downto 0)
38 );
39 end component;
40
41 signal A_in, u_carry: STD_LOGIC_VECTOR(3 downto 0);
42 signal op2: STD_LOGIC_VECTOR(3 downto 0);
43 signal riporto: STD_LOGIC;
44
45 begin
46
47 reg: registro4
48   port map(
49     A => A_in,
50     clk => CLK,
51     res => RST,
52     load => load_A,
53     B => op2
54   );
55
56 CLA: CarryLookAhead
57   port map(
58     X => op2,
59     Y => din,
60     cin => '0',
61     cout => riporto,
62     sum => u_carry
63   );
64
65 mux: mux2_1
66   port map(
67     x0 => (others => '0'),
68     x1 => u_carry,
69     s => selA,
70     y => A_in
71   );
72
73 res: process(CLK, load_A)
74 begin
75   if (CLK'event AND CLK = '1') then
76     if (load_A = '1') then
77       result <= u_carry;
78     end if;

```

```
79      end if;
80  end process;
81
82 end Behavioral;
```

Code 10.9: unità operativa di B in vhdl

Come già visto, per la gestione delle abilitazioni e del funzionamento si utilizza un'unità di controllo, modellata sulla base degli automi progettati nel paragrafo precedente.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity UCB is
5   Port (
6     CLK, RST: in STD_LOGIC;
7     REQ: in STD_LOGIC;
8     state: out STD_LOGIC_VECTOR(1 downto 0);
9     ACK: out STD_LOGIC;
10    selA, EN_WRITE: out STD_LOGIC:= '0'
11  );
12 end UCB;
13
14 architecture Behavioral of UCB is
15 type stati is (IDLE, RECEIVE, SUM, WRITE);
16
17 signal current_state: stati;
18 signal next_state: stati;
19
20 begin
21
22 reg_stato: process(CLK, RST)
23 begin
24   if (CLK'event AND CLK = '1') then
25     if (RST = '1') then
26       current_state <= IDLE;
27     else
28       current_state <= next_state;
29     end if;
30   end if;
31 end process;
32
```

```

33  change: process(current_state, REQ)
34  begin
35      EN_WRITE <= '0';
36      ACK <= '0';
37      CASE current_state IS
38          WHEN IDLE =>
39              IF REQ = '0' THEN
40                  next_state <= current_state;
41              ELSE
42                  next_state <= RECEIVE;
43              END IF;
44          WHEN RECEIVE =>
45              ACK <= '1';
46              IF REQ = '1' THEN
47                  next_state <= current_state;
48              ELSE
49                  selA <= '1';
50                  next_state <= SUM;
51
52              END IF;
53          WHEN SUM =>
54              ACK <= '0';
55              EN_WRITE <= '1';
56              next_state <= WRITE;
57          WHEN WRITE =>
58
59              EN_WRITE <= '0';
60              next_state <= IDLE;
61
62      END CASE;
63
64  end process;
65  state <= "00" WHEN current_state = IDLE ELSE
66  "01" WHEN current_state = RECEIVE ELSE
67  "10" WHEN current_state = SUM ELSE
68  "11" WHEN current_state = WRITE;
69
70 end Behavioral;

```

Code 10.10: Control Unit di A.vhdl

Il nodo B nel suo complesso viene implementato in questo modo:

---

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;

```

```

3
4 entity B is
5     Port (
6         CLK, RST: in STD_LOGIC;
7         REQ: in STD_LOGIC;
8         din: in STD_LOGIC_VECTOR(3 downto 0);
9         ACK: out STD_LOGIC;
10        state: out STD_LOGIC_VECTOR(1 downto 0);
11        result: out STD_LOGIC_VECTOR(3 downto 0)
12    );
13 end B;
14
15 architecture structural of B is
16     component unita_operativaB
17     Port (
18         CLK, RST: in STD_LOGIC;
19         selA: in STD_LOGIC;
20         din: in STD_LOGIC_VECTOR(3 downto 0);
21         load_A: in STD_LOGIC;
22         result: out STD_LOGIC_VECTOR(3 downto 0)
23     );
24 end component;
25
26 component UCB
27     Port (
28         CLK, RST: in STD_LOGIC;
29         REQ: in STD_LOGIC;
30         state: out STD_LOGIC_VECTOR(1 downto 0);
31         EN_WRITE, ACK, selA: out STD_LOGIC
32     );
33 end component;
34 signal temp_load, temp_en_sum, temp_sel: STD_LOGIC;
35 begin
36
37     uo: unita_operativaB
38     port map(
39         CLK => CLK,
40         RST => RST,
41         selA => temp_sel,
42         din => din,
43         load_A => temp_load,
44         result => result
45     );
46
47
48     uc: UCB

```

```

49      port map(
50          CLK => CLK,
51          RST => RST,
52          REQ => REQ,
53          state => state,
54          EN_WRITE => temp_load,
55          ACK => ACK,
56          selA => temp_sel
57      );
58  );
59
60 end structural;

```

Code 10.11: nodo B.vhdl

Il sistema compolessivo composto dai due nodi creato in precedenza si implementa come segue:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4
5 entity AplusB is
6 Port (
7     CLK_A, CLK_B, RST: in STD_LOGIC;
8     START: in STD_LOGIC;
9     stateA, stateB: out STD_LOGIC_VECTOR(1 downto 0);
10    count: out STD_LOGIC_VECTOR(2 downto 0);
11    result: out STD_LOGIC_VECTOR(3 downto 0)
12 );
13 end AplusB;
14
15 architecture Behavioral of AplusB is
16 component A
17 Port (
18     CLK, RST: in STD_LOGIC;
19     START, ACK: in STD_LOGIC;
20     REQ: out STD_LOGIC;
21     state: out STD_LOGIC_VECTOR(1 downto 0);
22     count: out STD_LOGIC_VECTOR(2 downto 0);
23     dout: out STD_LOGIC_VECTOR(3 downto 0)
24 );
25 end component;
26

```

```

27 component B
28 Port (
29     CLK, RST: in STD_LOGIC;
30     REQ: in STD_LOGIC;
31     din: in STD_LOGIC_VECTOR(3 downto 0);
32     ACK: out STD_LOGIC;
33     state: out STD_LOGIC_VECTOR(1 downto 0);
34     result: out STD_LOGIC_VECTOR(3 downto 0)
35 );
36 end component;
37
38 signal temp_ACK, temp_REQ: STD_LOGIC;
39 signal uA: STD_LOGIC_VECTOR(3 downto 0);
40 begin
41
42 A0: A
43     port map(
44         CLK => CLK_A,
45         RST => RST,
46         START => START,
47         ACK => temp_ACK,
48         REQ => temp_REQ,
49         state => stateA,
50         count => count,
51         dout => uA
52     );
53
54 B0: B
55     port map(
56         CLK => CLK_B,
57         RST => RST,
58         REQ => temp_REQ,
59         din => uA,
60         ACK => temp_ACK,
61         state => stateB,
62         result => result
63     );
64
65 end Behavioral;

```

Code 10.12: AplusB.vhdl

Si osserva anche lo schematico complessivo generato dall'ambiente Vivado:

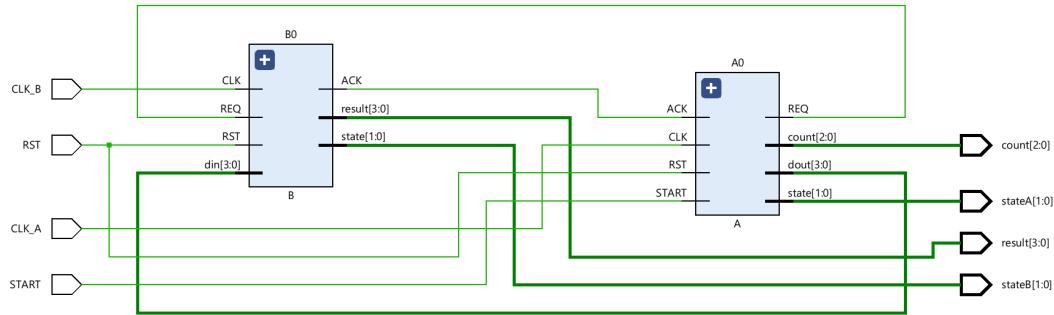


Figura 10.5: Schema a blocchi: nodo B

### 10.1.4 Simulazione

Per procedere alla simulazione, è necessaria un testbench:

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE ieee.numeric_std.ALL;
4
5 ENTITY tb_AplusB IS
6 END tb_AplusB;
7
8 ARCHITECTURE behavior OF tb_AplusB IS
9
10 COMPONENT AplusB
11 PORT (
12     CLK_A : IN STD_LOGIC;
13     CLK_B : IN STD_LOGIC;
14     RST : IN STD_LOGIC;
15     START : IN STD_LOGIC;
16     stateA : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
17     stateB : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
18     count : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
19     result : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
20 );
21 END COMPONENT;
22
23 -- Signals to connect to UUT

```

```

24      SIGNAL CLK_A : STD_LOGIC := '0';
25      SIGNAL CLK_B : STD_LOGIC := '0';
26      SIGNAL RST : STD_LOGIC := '0';
27      SIGNAL START : STD_LOGIC := '0';
28      SIGNAL stateA : STD_LOGIC_VECTOR(1 DOWNTO 0);
29      SIGNAL stateB : STD_LOGIC_VECTOR(1 DOWNTO 0);
30      SIGNAL count : STD_LOGIC_VECTOR(2 DOWNTO 0);
31      SIGNAL result : STD_LOGIC_VECTOR(3 DOWNTO 0);

32
33      -- Clock periods
34      CONSTANT CLK_A_PERIOD : TIME := 10 ns;
35      CONSTANT CLK_B_PERIOD : TIME := 12 ns;

36
37 BEGIN
38
39     uut: AplusB PORT MAP (
40         CLK_A => CLK_A,
41         CLK_B => CLK_B,
42         RST => RST,
43         START => START,
44         stateA => stateA,
45         stateB => stateB,
46         count => count,
47         result => result
48     );
49
50     -- Clock process for CLK_A
51     CLK_A_process : PROCESS
52     BEGIN
53         CLK_A <= '0';
54         WAIT FOR CLK_A_PERIOD/2;
55         CLK_A <= '1';
56         WAIT FOR CLK_A_PERIOD/2;
57     END PROCESS;
58
59     -- Clock process for CLK_B
60     CLK_B_process : PROCESS
61     BEGIN
62         CLK_B <= '0';
63         WAIT FOR CLK_B_PERIOD/2;
64         CLK_B <= '1';
65         WAIT FOR CLK_B_PERIOD/2;
66     END PROCESS;
67
68     -- Stimulus process
69     stim_proc: PROCESS

```

```

70      BEGIN
71          -- Reset the system
72          RST <= '1';
73          WAIT FOR 20 ns;
74          RST <= '0';
75
76          -- Start the operation
77          START <= '1';
78          WAIT FOR 50 ns;
79          START <= '0';
80
81          -- Wait and observe results
82          WAIT FOR 200 ns;
83
84          -- Stop simulation
85          WAIT;
86      END PROCESS;
87
88 END;

```

Code 10.13: testbench.vhdl

E eseguendo tale simulazione si ottiene la seguente waveform:

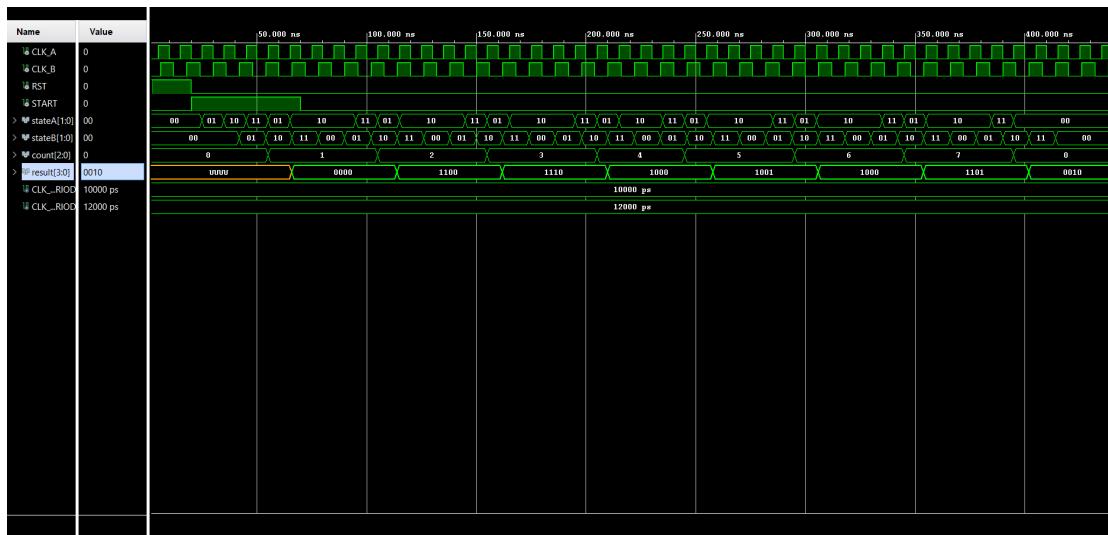


Figura 10.6: Waveform

Per valutare la correttezza della simulazione si ricordano gli elementi di ROM:

$\text{ROM}[0] = 0000$

$\text{ROM}[1] = 1100$

$\text{ROM}[2] = 0010$

$\text{ROM}[3] = 1010$

$\text{ROM}[4] = 0001$

$\text{ROM}[5] = 1111$

$\text{ROM}[6] = 0101$

$\text{ROM}[7] = 0011$

Inizializzando il registro R a 0 e poi sommando progressivamente i valori a due a due si ottiene:

$$0000 + 0000 = 0000$$

$$0000 + 1100 = 1100$$

$$1100 + 0010 = 1110$$

$$1110 + 1010 = 1000$$

$$1000 + 0001 = 1001$$

$$1001 + 1111 = 1000$$

$$1000 + 0101 = 1101$$

$$\mathbf{1101 + 0011 = 0010}$$

Quindi alla fine sul registro sarà memorizzata la stringa 0010, che corrisponde alla somma di tutti gli elementi della ROM presente in A.

# Bibliografia

- [1] Carlo Brandoles. *Introduzione al linguaggio VHDL - Aspetti teorici ed esempi di progettazione.* Politecnico di Milano.
- [2] Rocco di Torrepidula Franca e Somma Alessadra. *Architettura dei Sistemi di Elaborazione - Appunti tratti dalle lezioni del prof N. Mazzocca.*
- [3] Alberto Morriconi. amic-0. [https://github.com/albmoriconi/amic-0.](https://github.com/albmoriconi/amic-0)
- [4] Shira. Prosthetic hand. [https://www.thingiverse.com/thing:1489003.](https://www.thingiverse.com/thing:1489003)