



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

## *Architettura dei Sistemi Digitali*

Anno Accademico 2023/2024

Candidati

**M63001645** Francesco Balassone  
**M63001670** Vincenzo Luigi Bruno  
**M63001627** Luca Pisani

---

*Una macchina sequenziale a una combinatoria: "Hey, che hai combinato?" Lei: "Non ne ho memoria..."*

# Indice

<b>1 Multiplexer</b>	<b>1</b>
1.1 Traccia . . . . .	1
1.2 Progetto e architettura . . . . .	2
1.2.1 Multiplexer 16:1 . . . . .	2
1.2.2 Rete di interconnessione . . . . .	3
1.3 Implementazione . . . . .	4
1.3.1 Multiplexer 16:1 . . . . .	4
1.4 Simulazione . . . . .	6
1.4.1 Multiplexer 16:1 . . . . .	6
1.4.2 Rete di Interconnessione . . . . .	7
1.5 Sintesi su board di sviluppo . . . . .	11
<b>2 Sistema ROM+M</b>	<b>15</b>
2.1 Traccia . . . . .	15
2.2 Progetto e architettura . . . . .	16
2.3 Implementazione . . . . .	17
2.3.1 Memoria ROM 16 locazioni 8 bit . . . . .	17
2.3.2 Macchina combinatoria: shift di 4 locazioni . .	19
2.3.3 Sistema complessivo ROM+M . . . . .	20

2.3.4	Simulazione . . . . .	22
2.4	Sintesi e implementazione su board di sviluppo . . . . .	25
<b>3</b>	<b>Riconoscitore di sequenze 101</b>	<b>27</b>
3.1	Traccia . . . . .	28
3.2	Automa . . . . .	29
3.3	Progetto e architettura . . . . .	29
3.4	Implementazione . . . . .	30
3.5	Simulazione . . . . .	39
3.6	Sintesi su board di sviluppo . . . . .	44
3.7	Timing Analysis . . . . .	57
<b>4</b>	<b>Shift Register</b>	<b>62</b>
4.1	Traccia . . . . .	62
4.2	Progetto e architettura . . . . .	63
4.3	Implementazione: approccio behavioral . . . . .	64
4.4	Implementazione: approccio structural . . . . .	67
4.5	Simulazione: approccio behavioral . . . . .	75
4.6	Simulazione: approccio strutturale . . . . .	76
<b>5</b>	<b>Cronometro</b>	<b>77</b>
5.1	Traccia 4.1 . . . . .	77
5.1.1	Progetto e architettura . . . . .	78
5.1.2	Implementazione . . . . .	79
5.1.3	Simulazione . . . . .	90
5.2	Traccia 5.2 . . . . .	91
5.2.1	Progetto e architettura . . . . .	91

5.2.2	Sintesi su board . . . . .	111
5.3	Traccia 5.3 . . . . .	119
5.3.1	Progetto e architettura . . . . .	119
5.3.2	Implementazione . . . . .	123
5.3.3	Sintesi su Board . . . . .	130
<b>6</b>	<b>Sistema PO _ PC</b>	<b>137</b>
6.1	Traccia . . . . .	137
6.2	Progetto e architettura . . . . .	138
6.3	Implementazione . . . . .	140
6.3.1	Control Unit . . . . .	140
6.3.2	SistemaPO _ PC . . . . .	143
6.3.3	Macchina Combinatoria . . . . .	145
6.4	Simulazione . . . . .	146
6.5	Implementazione su Board . . . . .	146
6.5.1	CU _ sintesi . . . . .	147
6.5.2	SistemaOnBoard . . . . .	150
<b>7</b>	<b>Moltiplicatore Booth</b>	<b>157</b>
7.1	Traccia . . . . .	157
7.2	Progetto e Architettura . . . . .	158
7.3	Implementazione . . . . .	161
7.3.1	Booth Multiplier . . . . .	161
7.3.2	Unità di Controllo . . . . .	164
7.3.3	Unità Operativa . . . . .	168
7.3.4	Shift Register . . . . .	171
7.3.5	Registro M . . . . .	174

7.3.6	Addizionatore - Sottrattore . . . . .	176
7.4	Implementazione su Board . . . . .	179
<b>8</b>	<b>Handshaking</b>	<b>185</b>
8.1	Traccia . . . . .	185
8.2	Progetto e Architettura . . . . .	186
8.3	Implementazione . . . . .	190
8.3.1	Nodo A . . . . .	190
8.3.2	CU A . . . . .	192
8.3.3	Nodo B . . . . .	195
8.3.4	CU B . . . . .	198
8.4	Simulazione . . . . .	200
<b>9</b>	<b>Processore MIC-1</b>	<b>202</b>
9.1	Traccia . . . . .	202
9.2	Architettura del processore . . . . .	202
9.2.1	Datapath . . . . .	204
9.2.2	Control Unit . . . . .	207
9.3	Istruzioni Analizzate . . . . .	209
9.3.1	BIPUSH . . . . .	209
9.3.2	IOR . . . . .	212
9.3.3	Istruzione modificata: IADD3OP . . . . .	215
<b>10</b>	<b>Interfaccia Seriale</b>	<b>220</b>
10.1	Traccia . . . . .	220
10.2	Progetto e architettura . . . . .	221
10.3	Trasmettitore TX - Architettura UART . . . . .	222

10.4 Ricevitore RX - Architettura UART . . . . .	225
10.5 Implementazione progetto . . . . .	227
10.5.1 Unità A . . . . .	227
10.5.2 Automa a stati finiti - CU A . . . . .	228
10.5.3 Unità B . . . . .	230
10.5.4 Automa a stati finiti - CU B . . . . .	230
10.6 Implementazione . . . . .	231
10.6.1 Comunicazione _ Seriale . . . . .	232
10.6.2 CU A . . . . .	233
10.6.3 Unita A . . . . .	236
10.6.4 CU B . . . . .	238
10.6.5 Unita B . . . . .	241
10.7 Simulazione . . . . .	243
<b>11 Switch Multistadio</b>	<b>244</b>
11.1 Traccia . . . . .	244
11.2 Cenni teorici . . . . .	244
11.3 Progetto e architettura . . . . .	248
11.4 Implementazione . . . . .	250
11.5 Simulazione . . . . .	257
<b>12 Appendice</b>	<b>259</b>
12.1 Flip Flop D . . . . .	259
12.2 MUX 2:1 . . . . .	260
12.3 DEMUX 1:2 . . . . .	261
12.4 ROM . . . . .	262
12.5 ROM Comb . . . . .	264

12.6 Contatore MOD M . . . . .	265
12.7 Counter MOD 8 . . . . .	267
12.8 Memoria RW . . . . .	269
12.9 Adder . . . . .	270
12.10 Registro . . . . .	271
12.11 Full Adder . . . . .	273
12.12 Ripple Carry Adder . . . . .	274
12.13 Adder / Subtractor . . . . .	276
12.14 Button Debouncer . . . . .	278

# Chapter 1

## Multiplexer

### 1.1 Traccia

- Progettare, implementare in VHDL e testare mediante simulazione un multiplexer indirizzabile 16:1, utilizzando un approccio di progettazione per composizione a partire da multiplexer 4:1.
- Utilizzando il componente sviluppato al punto precedente, progettare, implementare in VHDL e testare mediante simulazione una rete di interconnessione a 16 sorgenti e 4 destinazioni.
- 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.

## 1.2 Progetto e architettura

### 1.2.1 Multiplexer 16:1

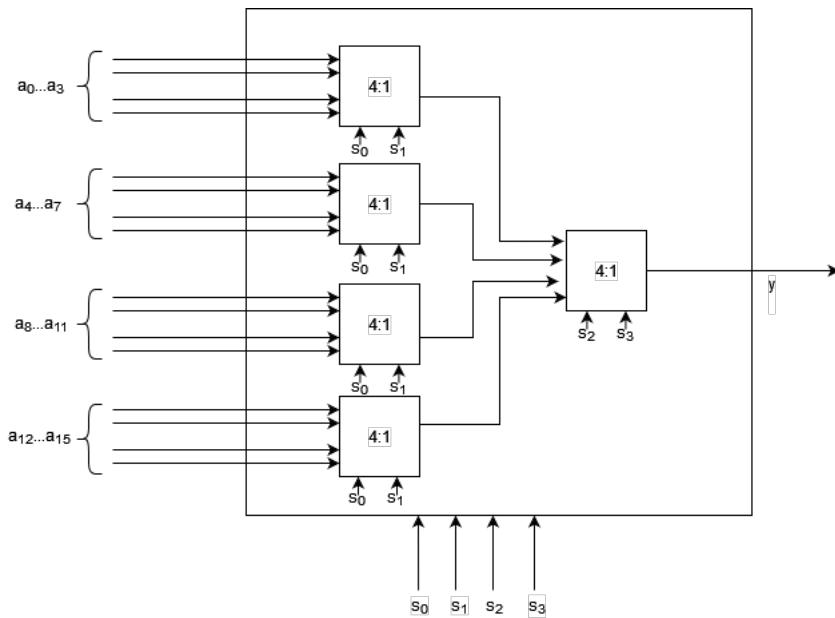


Figure 1.1: MUX 16:1

Il multiplexer 16:1 è un circuito integrato che si caratterizza per la sua capacità di selezionare un **singolo segnale di uscita da sedici segnali di ingresso**, utilizzando **quattro linee di selezione**.

La progettazione di questo dispositivo segue un approccio per composizione, impiegando 5 multiplexer 4:1, i quali sono a loro volta cos-

tituiti da unità 2:1 più semplici. Nella configurazione proposta, i due bit di selezione meno significativi determinano il segnale proveniente dai multiplexer al primo livello, mentre i due bit più significativi selezionano il segnale finale da inviare all'uscita.

### 1.2.2 Rete di interconnessione

La struttura della **rete di interconnessione** si articola attraverso l'impiego di un **multiplexer 16:1** e un **demultiplexer 1:4**, quest'ultimo collegato direttamente all'uscita del multiplexer.

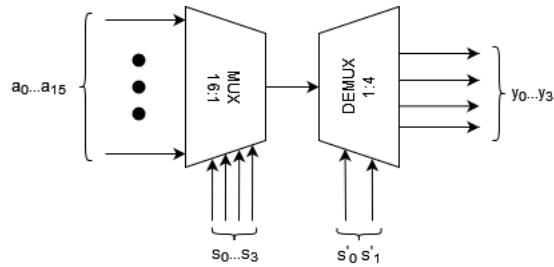


Figure 1.2: Rete di interconnessione

Questa configurazione ci permette di creare una rete di interconnessione efficiente, caratterizzata da sei linee di selezione. Di queste, le prime quattro sono dedicate alla determinazione della sorgente del segnale, mentre le ultime due sono impiegate per definire la destinazione. Anche in questo contesto, viene adottata una strategia che prevede l'integrazione di componenti precedentemente sviluppati, al fine di ottimizzare sia la funzionalità sia l'efficienza complessiva del sistema.

## 1.3 Implementazione

### 1.3.1 Multiplexer 16:1

La struttura VHDL del multiplexer 16 a 1 è illustrata attraverso una definizione di interfaccia e un'implementazione architetturale. L'entità mux\_16\_1 funge da interfaccia esterna e definisce 16 linee di ingresso di dati, 4 linee di selezione e una singola uscita.

```

1 entity mux_16_1 is
2   port (
3     B: in std_logic_vector(15 downto 0); --vettore
4       di ingressi in input (dim=16)
5     S: in std_logic_vector(3 downto 0); --vettore di
6       ingressi di selezione (dim=4)
7     Y: out std_logic --una uscita
8   );
9 end mux_16_1;

```

L'architettura structural di mux\_16\_1 impiega il design per composizione, avvalendosi di cinque istanze del componente mux\_4\_1. Quattro di questi gestiscono i gruppi di ingressi e usano i due bit meno significativi per la selezione. L'output di questi multiplexer di primo livello è poi instradato a un quinto multiplexer di secondo livello, che utilizza i due bit più significativi per determinare il segnale finale da emettere.

```

1 architecture Structural of mux_16_1 is
2

```

```
3 component mux_4_1 is
4 port (
5     A: in std_logic_vector(3 downto 0); --4
6         ingressi dato
7     S: in std_logic_vector(1 downto 0); --2
8         ingressi di selezione
9     Y: out std_logic --1 uscita
10    );
11 end component;
12
13
14 begin
15     --Genero i mux_4_1
16     Mux_0_4: FOR i IN 0 TO 4 GENERATE
17         --Multiplexer di primo livello
18         M_0_3: IF i < 4 GENERATE --Genero i primi 4
19             mux_4_1
20             M: mux_4_1 port map(
21                 --ingressi 3,2,1,0 - 7,6,5,4 -
22                 -- 11,10,9,8 - 15,14,13,12
23                 A => B(i*4+3 downto i*4),
24                 S => S(1 downto 0), --intervengono solo
25                     i primi 2 bit di selezione
26                 Y => U(i)
27                 --le uscite dei multiplexer di primo
```

```

                livello vengono memorizzate in U
25             );
26
27
28 M_4: IF i = 4 GENERATE --Genero l'ultimo
29     mux_4_1 (di secondo livello)
30
31     M: mux_4_1 port map(
32         A => U, --prende in ingresso le uscite
33         dei mu di primo livello
34         S => S(3 downto 2), --ultimi 2 bit di
35         selezione
36         Y => Y --uscita del mux_16_1
37     );
38
39 END GENERATE;
40
41 END GENERATE;
42
43
44 end Structural;

```

Questa progettazione strutturale consente una gestione efficiente e modulare dei segnali all'interno del multiplexer complessivo.

## 1.4 Simulazione

### 1.4.1 Multiplexer 16:1

Nella simulazione, vengono eseguiti diversi test per verificare il corretto funzionamento del multiplexer. Ogni test consiste nel configurare un

certo valore per le linee di ingresso (B), selezionare una specifica linea di ingresso tramite il segnale di selezione (S), e poi verificare che il valore all'uscita (Y) sia quello atteso.

I test includono casi in cui solo una specifica linea di ingresso è alta (per esempio, solo la linea di ingresso 2, 8 o 14), casi in cui più linee di ingresso sono alte (per esempio, le linee di ingresso 7 e 15), e casi in cui tutte le linee di ingresso sono basse o alte. In ogni caso, il valore atteso all'uscita è il valore della linea di ingresso selezionata.

Questi test permettono di verificare che il multiplexer funzioni correttamente in una varietà di scenari, garantendo che possa selezionare correttamente qualsiasi linea di ingresso e inoltrare il suo valore all'uscita. Se tutti i test passano, si può avere fiducia che il multiplexer sia stato implementato correttamente. Se un test fallisce, il messaggio di errore associato aiuta a identificare il problema da risolvere.

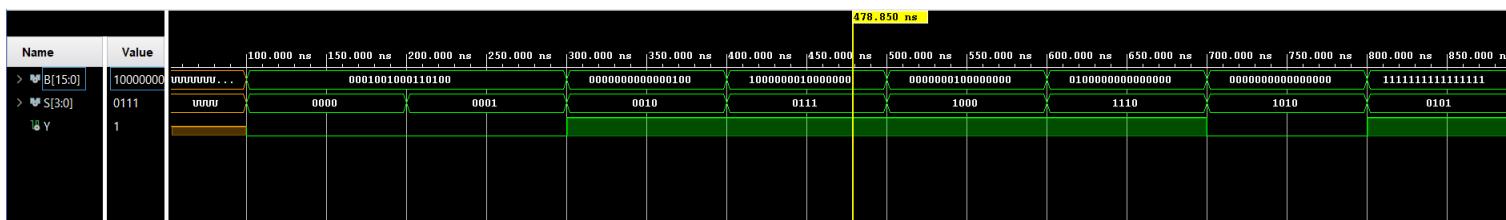


Figure 1.3: Simulazione

### 1.4.2 Rete di Interconnessione

Questo componente è progettato per gestire il flusso di dati tra diverse parti di un sistema digitale. La rete\_interconnessione è costituita da un multiplexer (mux\_16\_1) e un demultiplexer (demux\_1\_4), che la-

vorano insieme per indirizzare i dati da molteplici ingressi a molteplici uscite.

Il multiplexer prende in ingresso un vettore di 16 bit (A) e un segnale di selezione di 4 bit (S). In base al valore del segnale di selezione, il multiplexer sceglie uno dei 16 bit di ingresso e lo invia all'uscita (U).

Questo segnale di uscita viene poi inviato al demultiplexer. Il demultiplexer prende in ingresso un segnale singolo (U) e un segnale di selezione di 2 bit (L). In base al valore del segnale di selezione, il demultiplexer indirizza il segnale di ingresso a una delle 4 uscite (Y).

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 -- Definizione dell'entità rete_interconnessione
5 entity rete_interconnessione is
6     Port (
7         A: in std_logic_vector(15 downto 0); -- Vettore
8             di ingresso di 16 bit
9         S: in std_logic_vector(3 downto 0); -- Vettore
10            di selezione di 4 bit per il Mux
11         L: in std_logic_vector(1 downto 0); -- Vettore
12             di selezione di 2 bit per il Demux
13         Y: out std_logic_vector(3 downto 0) -- Vettore
14             di uscita di 4 bit dal Demux
15     );
16 end rete_interconnessione;
```

```
14 architecture Structural of rete_interconnessione is
15
16     signal U: std_logic; -- Segnale intermedio tra Mux
17     e Demux
18
19     -- Dichiarazione del componente mux_16_1
20     component mux_16_1 is
21         port (
22             B: in std_logic_vector(15 downto 0); -- Ingresso di 16 bit per il Mux
23             S: in std_logic_vector(3 downto 0); -- Selezione di 4 bit per il Mux
24             Y: out std_logic -- Uscita singola dal Mux
25         );
26
27     end component;
28
29     -- Dichiarazione del componente demux_1_4
30     component demux_1_4 is
31         port (
32             A: in std_logic; -- Ingresso singolo per il Demux
33             S: in std_logic_vector(1 downto 0); -- Selezione di 2 bit per il Demux
34             Y: out std_logic_vector(3 downto 0) -- Uscita di 4 bit dal Demux
35         );
36
37     end component;
```

```

35
36 begin
37
38     -- Istanziamento del Mux
39
40     Mux: mux_16_1 port map(
41
42         B => A,    -- Collegamento ingresso A al Mux
43
44         S => S,    -- Collegamento selezione S al Mux
45
46         Y => U      -- Collegamento uscita Mux al segnale
47
48             intermedio U
49
50
51     );
52
53
54     -- Istanziamento del Demux
55
56     Demux: demux_1_4 port map(
57
58         A => U,    -- Collegamento segnale intermedio U
59
60             al Demux
61
62         S => L,    -- Collegamento selezione L al Demux
63
64         Y => Y      -- Collegamento uscita Demux al
65
66             vettore di uscita Y
67
68     );
69
70
71 end Structural;

```

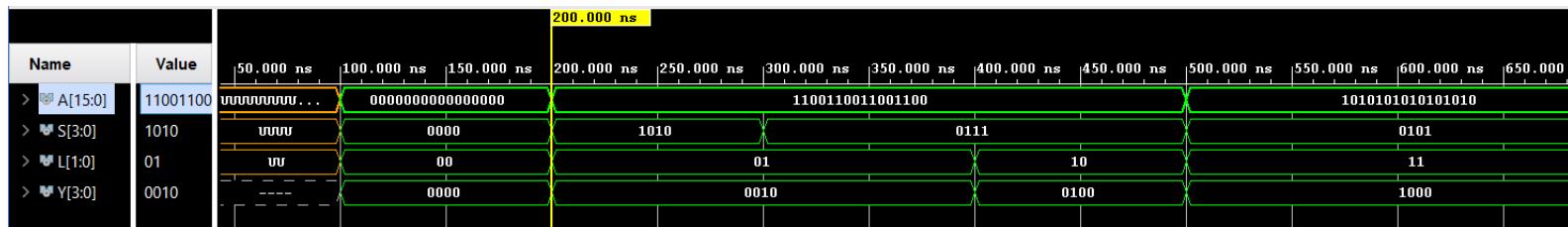


Figure 1.4: Simulazione

## 1.5 Sintesi su board di sviluppo

Per gestire l’acquisizione dei dati in input e dei fili di selezione abbiamo avuto bisogno di un’unità di controllo.

L’unita\\_controllo ha diverse porte di ingresso, tra cui load\\_first\\_part, load\\_second\\_part e load\\_sel, che sono collegati a dei pulsanti. Quando il pulsante load\\_first\\_part viene premuto, i primi 8 bit del segnale di ingresso value8\\_in vengono caricati nel registro reg\\_value. Quando viene premuto il pulsante load\\_second\\_part, i secondi 8 bit di value8\\_in vengono caricati in reg\\_value. Infine, quando viene premuto il pulsante load\\_sel, i valori dei selettori selIn\\_M e selIn\\_D vengono caricati nei registri reg\\_selM e reg\\_selD rispettivamente.

Questi registri memorizzano i valori fino a quando non vengono caricati nuovi valori. I valori memorizzati nei registri vengono poi inviati alle porte di uscita corrispondenti (value16\\_out, selOut\\_M, selOut\\_D), da dove possono essere utilizzati per controllare la rete di interconnessione.

Nel corso della configurazione fisica, abbiamo definito specifiche associazioni tra i pin e le funzionalità del sistema. In particolare, il pin N17 è stato associato al pulsante di reset, il pin M18 al pulsante load\\_sel per caricare i bit di selezione, il pin P17 al pulsante load\\_first\\_part per caricare la prima parte dell’input, e il pin M17 al pulsante load\\_second\\_part per caricare la seconda parte dell’input. Questa decisione di suddividere il caricamento dell’input in due parti

è stata presa a causa della limitazione degli switch, che devono fornire anche i valori di selezione e quindi non sono sufficienti per caricare l'intero input in una sola volta. Questa strategia consente un controllo più granulare del processo di caricamento dell'input, garantendo un funzionamento efficiente del multiplexer.

```
1  ## Clock signal
2  set_property -dict { PACKAGE_PIN E3      IOSTANDARD
3    LVCMOS33 } [get_ports { clock }];
4    #IO_L12P_T1_MRCC_35 Sch=clk100mhz
5
6  create_clock -add -name sys_clk_pin -period 10.00
7    -waveform {0 5} [get_ports {clock}];
8
9
10 ##Switches
11 set_property -dict { PACKAGE_PIN J15     IOSTANDARD
12    LVCMOS33 } [get_ports { switchData[0] }];
13    #IO_L24N_T3_RS0_15 Sch=sw[0]
14
15 set_property -dict { PACKAGE_PIN L16     IOSTANDARD
16    LVCMOS33 } [get_ports { switchData[1] }];
17    #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
18
19 set_property -dict { PACKAGE_PIN M13     IOSTANDARD
20    LVCMOS33 } [get_ports { switchData[2] }];
21    #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
22
23 set_property -dict { PACKAGE_PIN R15     IOSTANDARD
24    LVCMOS33 } [get_ports { switchData[3] }];
25    #IO_L13N_T2_MRCC_14 Sch=sw[3]
26
27 set_property -dict { PACKAGE_PIN R17     IOSTANDARD
28    LVCMOS33 } [get_ports { switchData[4] }];
```

```

#IO_L12N_T1_MRCC_14 Sch=sw[4]

11 set_property -dict { PACKAGE_PIN T18      IOSTANDARD
                      LVCMOS33 } [get_ports { switchData[5] }];

#IO_L7N_T1_D10_14 Sch=sw[5]

12 set_property -dict { PACKAGE_PIN U18      IOSTANDARD
                      LVCMOS33 } [get_ports { switchData[6] }];

#IO_L17N_T2_A13_D29_14 Sch=sw[6]

13 set_property -dict { PACKAGE_PIN R13      IOSTANDARD
                      LVCMOS33 } [get_ports { switchData[7] }];

#IO_L5N_T0_D07_14 Sch=sw[7]

14 set_property -dict { PACKAGE_PIN T8       IOSTANDARD
                      LVCMOS18 } [get_ports { switch_selM[0] }];

#IO_L24N_T3_34 Sch=sw[8]

15 set_property -dict { PACKAGE_PIN U8       IOSTANDARD
                      LVCMOS18 } [get_ports { switch_selM[1] }]; #IO_25_34
                      Sch=sw[9]

16 set_property -dict { PACKAGE_PIN R16      IOSTANDARD
                      LVCMOS33 } [get_ports { switch_selM[2] }];

#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]

17 set_property -dict { PACKAGE_PIN T13      IOSTANDARD
                      LVCMOS33 } [get_ports { switch_selM[3] }];

#IO_L23P_T3_A03_D19_14 Sch=sw[11]

18 set_property -dict { PACKAGE_PIN H6       IOSTANDARD
                      LVCMOS33 } [get_ports { switch_selD[0] }];

#IO_L24P_T3_35 Sch=sw[12]

19 set_property -dict { PACKAGE_PIN U12      IOSTANDARD
                      LVCMOS33 } [get_ports { switch_selD[1] }];

#IO_L20P_T3_A08_D24_14 Sch=sw[13]

```

```

20
21 ## LEDs
22 set_property -dict { PACKAGE_PIN H17      IOSTANDARD
23             LVCMOS33 } [get_ports { value_out[0] }];
24             #IO_L18P_T2_A24_15 Sch=led[0]
25 set_property -dict { PACKAGE_PIN K15      IOSTANDARD
26             LVCMOS33 } [get_ports { value_out[1] }];
27             #IO_L24P_T3_RS1_15 Sch=led[1]
28 set_property -dict { PACKAGE_PIN J13      IOSTANDARD
29             LVCMOS33 } [get_ports { value_out[2] }];
30             #IO_L17N_T2_A25_15 Sch=led[2]
31 set_property -dict { PACKAGE_PIN N14      IOSTANDARD
32             LVCMOS33 } [get_ports { value_out[3] }];
33             #IO_L8P_T1_D11_14 Sch=led[3]

34
35 #Buttons
36 set_property -dict { PACKAGE_PIN N17      IOSTANDARD
37             LVCMOS33 } [get_ports { reset }]; #IO_L9P_T1_DQS_14
38             Sch=btnc
39 set_property -dict { PACKAGE_PIN M18      IOSTANDARD
40             LVCMOS33 } [get_ports { load_sel }];
41             #IO_L4N_T0_D05_14 Sch=btnu
42 set_property -dict { PACKAGE_PIN P17      IOSTANDARD
43             LVCMOS33 } [get_ports { load_first_part }];
44             #IO_L12P_T1_MRCC_14 Sch=btnl
45 set_property -dict { PACKAGE_PIN M17      IOSTANDARD
46             LVCMOS33 } [get_ports { load_second_part }];
47             #IO_L10N_T1_D15_14 Sch=btnr

```

# Chapter 2

## Sistema ROM+M

In questo esercizio viene chiesto di progettare, implementare in VHDL e testare mediante simulazione, un sistema S composto da una memoria ROM combinatoria di 16 locazioni da 8 bit ciascuna e da una macchina combinatoria M. La ROM é progettata per essere un dispositivo di sola lettura, la macchina combinatoria M opera su un dato di 8 bit prelevato dalla ROM in base all'indirizzo fornito. Caratteristica fondamentale é il fatto che la macchina M trasforma il dato a 8 bit in ingresso in un nuovo valore a 4 bit.

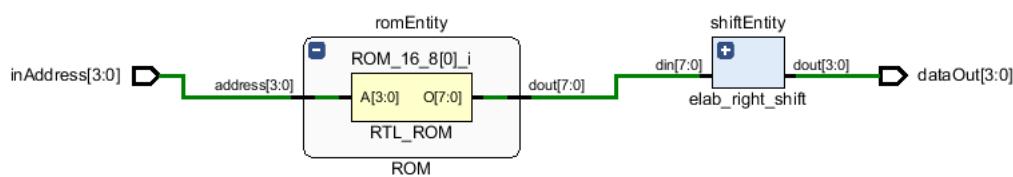
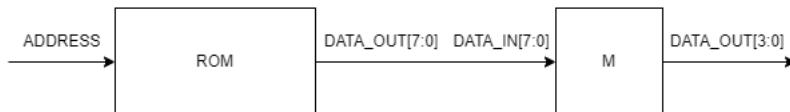
### 2.1 Traccia

- Progettare, implementare in VHDL e testare mediante simulazione un sistema S composto da una ROM puramente combinatoria di 16 locazioni da 8 bit ciascuna e da una macchina

combinatoria M che opera come segue: fornito al sistema un indirizzo A di 4 bit, il sistema restituisce il valore contenuto nella ROM all'indirizzo A opportunamente "trasformato" attraverso la macchina M. Il comportamento della macchina M è totalmente a scelta dello studente, l'unico vincolo è che essa prenda in ingresso 8 bit e ne fornisca in uscita 4.

- 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.

## 2.2 Progetto e architettura



Una memoria ROM (Read Only Memory) è un tipo di memoria digitale il cui contenuto non può essere modificato una volta definito in fase di programmazione. E' spesso utilizzata per immagazzinare dati costanti

che devono essere accessibili durante l'esecuzione di un circuito digitale. E' stata implementata una memoria ROM con 16 locazioni, o celle, ciascuna da 8 bit, utilizzando un array di vettori implementabile in VHDL.

## 2.3 Implementazione

### 2.3.1 Memoria ROM 16 locazioni 8 bit

L'entità ROM funge da interfaccia esterna e definisce 4 linee di ingresso dati, corrispondenti all'indirizzo A di 4 bit, e 8 linee di uscita che andranno a rappresentare il contenuto della memoria alla locazione puntata dall'indirizzo A.

```
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(3 downto 0);
8         dout      : out std_logic_vector(7 downto 0)
9     );
10 end entity ROM;
11
12 architecture RTL of ROM is
13     type MEMORY_16_8 is array (0 to 15) of
```

```
        std_logic_vector(7 downto 0);

14      constant ROM_16_8 : MEMORY_16_8 := (
15          x"00",
16          x"11",
17          x"22",
18          x"33",
19          x"44",
20          x"55",
21          x"66",
22          x"77",
23          x"88",
24          x"99",
25          x"aa",
26          x"bb",
27          x"cc",
28          x"dd",
29          x"ee",
30          x"ff"
31      );
32
33      begin
34          main : process(address)
35          begin
36              dout <= ROM_16_8(to_integer(unsigned(address)));
37          end process main;
38
39      end architecture RTL;
```

All'interno dell' architecture viene definito un tipo (type MEM-

ORY\_16\_8) per un array di 16 elementi, ognuno di 8 bit e con la keyword constant ROM\_16\_8 viene definito il contenuto della memoria ROM. A seguito dello statement "begin" viene dichiarato un processo chiamato "main", sensibile al segnale "address" inserito nella apposita sensitivity list. Al segnale "dout" viene assegnato il dato corrispondente all' indirizzo specificato. Il valore di address viene dapprima convertito in un intero senza segno ("unsigned") e poi in uno normale ("to\_integer") per accedere all' elemento corrispondente nell' array "ROM\_16\_8".

### 2.3.2 Macchina combinatoria: shift di 4 posizioni

E' stato deciso di elaborare il dato in uscita dalla memoria ROM\_16\_8 attraverso una macchina combinatoria in grado di effettuare uno shift a destra di 4 posizioni in modo da spostare i bit in entrata dal 7 al 4 nelle posizioni 3 downto 0 sul segnale di uscita "dout".

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity elab_right_shift is
6     port(
7         din : in std_logic_vector(7 downto 0);
8         dout: out std_logic_vector(3 downto 0)
9     );
10 end entity elab_right_shift;
```

```
11  
12 architecture dataflow of elab_right_shift is  
13 begin  
14     dout <= din(7 downto 4); --shift right di 4  
15     posizioni  
16 end architecture dataflow;
```

L' entità "elab\_right\_shift" viene dichiarata: presenta un vettore di otto linee in ingresso dedicate ad accogliere il dato in ingresso ed un vettore di quattro linee in uscita.

### 2.3.3 Sistema complessivo ROM+M

Nella prima parte viene dichiarata un entità "Sistema" con due port:

- "inAddress": un input di 4 bit che rappresenta l'indirizzo della ROM.
- "dataOut": un output di 4 bit che rappresenta il dato in uscita dal sistema.

```
1 entity Sistema is  
2     port (  
3         inAddress : in  std_logic_vector(3 downto 0);  
4         dataOut      : out std_logic_vector(3 downto 0)  
5     );  
6 end entity Sistema;
```

A seguire viene definita l'architettura di tipo structural: viene impiegato un design per composizione, i componenti (ROM\_16\_8 e elab\_right\_shift) vengono istanziati per poi essere opportunamente collegati tra loro ed alle linee di ingresso ed uscita definite dall'interfaccia del sistema.

```
1 architecture RTL of Sistema is
2
3     component ROM is
4
5         port (
6             address : in std_logic_vector(3 downto 0);
7             dout      : out std_logic_vector(7 downto 0)
8         );
9
10    end component ROM;
11
12
13    component elab_right_shift is
14
15        port (
16            din   : in std_logic_vector(7 downto 0);
17            dout  : out std_logic_vector(3 downto 0)
18        );
19
20    end component elab_right_shift;
21
22
23    signal rom_out : std_logic_vector(7 downto 0);
24
25
26 begin
27
28     romEntity: ROM port map (
29
30         address => inAddress, --mappa l'ingresso della
31
32             ROM all'ingresso del sistema
33
34             --quando un indirizzo viene fornito al
```

```
    sistema, viene passato alla ROM  
22      dout    => rom_out --i dati in uscita dalla ROM  
          vengono passati al segnale  
23  );  
24  
25  shiftEntity: elab_right_shift port map (  
26      din  => rom_out, --i dati di uscita ROM vengono  
          passati come ingresso alla macchina  
          combinatoria  
27      dout => dataOut  --mappo l'uscita della  
          macchina all'uscita del sistema  
28  );  
29  
30 end architecture RTL;
```

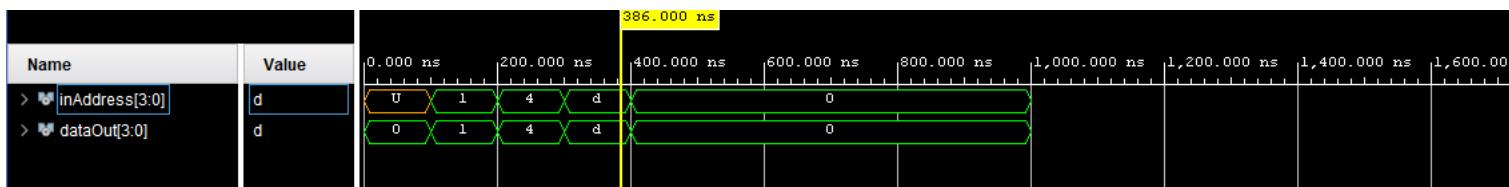
E' stato definito un segnale "rom\_out" di 8 bit in modo da memorizzare il dato in uscita dalla ROM prima di passarlo al componente di shift (segnale di appoggio).

### 2.3.4 Simulazione

```
1 library IEEE;  
2 use IEEE.Std_logic_1164.all;  
3 use IEEE.Numeric_Std.all;  
4  
5 entity Sistema_tb is  
6 end;
```

```
8  architecture bench of Sistema_tb is
9
10 component Sistema
11
12     port (
13         inAddress : in  std_logic_vector(3 downto 0);
14         dataOut      : out std_logic_vector(3 downto 0)
15     );
16
17 end component;
18
19
20 begin
21
22     uut: Sistema port map ( inAddress => inAddress,
23                               dataOut      => dataOut );
24
25     stimulus: process
26
27     begin
28
29         wait for 100ns;
30
31         inAddress <= "0001";
32         wait for 100ns;
33         assert dataOut = "0001" report "Test fallito"
34             severity error;
35
36         inAddress <= "0100";
```

```
35      wait for 100ns;
36
37      assert dataOut = "0100" report "Test fallito"
38          severity error;
39
40      inAddress <= "1101";
41
42      wait for 100ns;
43
44      assert dataOut = "1101" report "Test fallito"
45          severity error;
46
47      inAddress <= (others => '0');
48
49      wait for 100ns;
50
51      assert dataOut = "0" report "Test fallito per tutti
52          gli zeri" severity error;
53
54      inAddress <= (others => '0');
55
56      wait for 100 ns;
57
58      assert dataOut = "1" report "Test fallito per tutti
59          gli uno" severity error;
60
61      wait;
62
63
64  end process;
65
66
67 end;
```



## 2.4 Sintesi e implementazione su board di sviluppo

Una volta completata la scrittura e la simulazione del codice VHDL, il prossimo passo è la sintesi e l'implementazione del design sulla board di sviluppo.

Abbiamo specificato i constraint fisici per poter collegare le linee di ingresso ai primi 4 switch della scheda e le linee di uscita ai primi 4 led.

```

1 ##Switches
2 set_property -dict { PACKAGE_PIN J15      IOSTANDARD
                      LVCMOS33 } [get_ports { inAddress[0] }];
                      #IO_L24N_T3_RS0_15 Sch=sw[0]
3 set_property -dict { PACKAGE_PIN L16      IOSTANDARD
                      LVCMOS33 } [get_ports { inAddress[1] }];
                      #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
4 set_property -dict { PACKAGE_PIN M13      IOSTANDARD
                      LVCMOS33 } [get_ports { inAddress[2] }];
                      #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
5 set_property -dict { PACKAGE_PIN R15      IOSTANDARD
                      LVCMOS33 } [get_ports { inAddress[3] }];
                      #IO_L13N_T2_MRCC_14 Sch=sw[3]
```

```
1  ## LEDs
2  set_property -dict { PACKAGE_PIN H17      IOSTANDARD
3          LVCMOS33 } [get_ports { dataOut[0] }];
4          #IO_L18P_T2_A24_15 Sch=led[0]
5
6  set_property -dict { PACKAGE_PIN K15      IOSTANDARD
7          LVCMOS33 } [get_ports { dataOut[1] }];
8          #IO_L24P_T3_RS1_15 Sch=led[1]
9
10 set_property -dict { PACKAGE_PIN J13     IOSTANDARD
11        LVCMOS33 } [get_ports { dataOut[2] }];
12        #IO_L17N_T2_A25_15 Sch=led[2]
13
14 set_property -dict { PACKAGE_PIN N14     IOSTANDARD
15        LVCMOS33 } [get_ports { dataOut[3] }];
16        #IO_L8P_T1_D11_14 Sch=led[3]
```

# Chapter 3

## Riconoscitore di sequenze

### 101

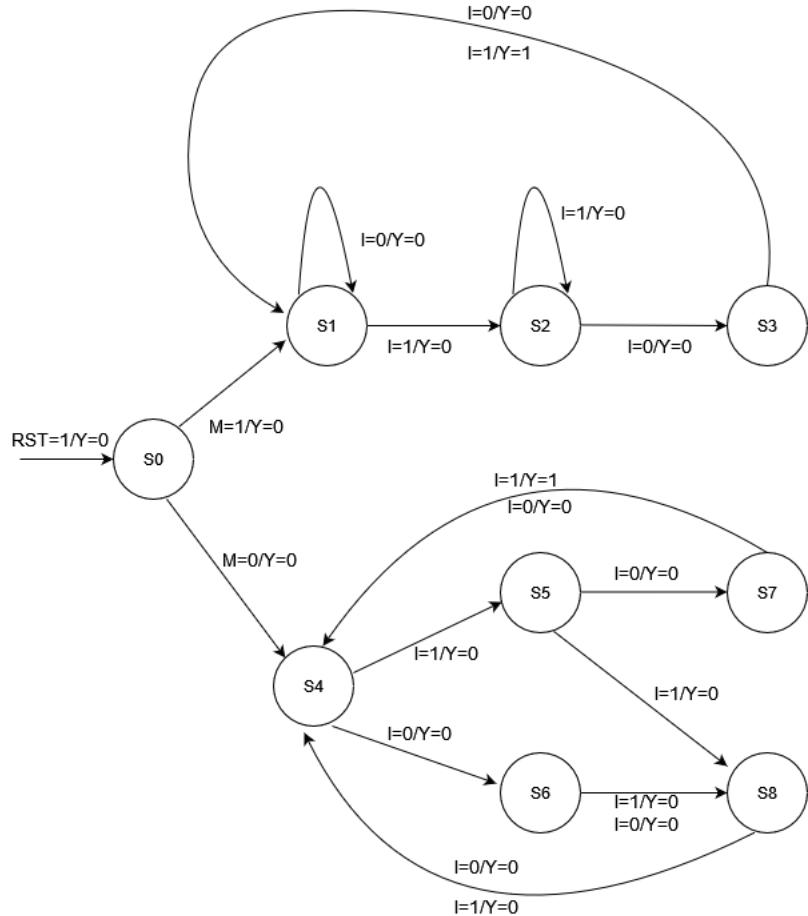
Questo esercizio richiede la realizzazione di un riconoscitore di sequenza, una macchina sequenziale elementare. Una macchina sequenziale é per definizione un sistema la cui uscita in un certo istante dipende non solo dall'ingresso applicato nello stesso istante ma anche agli ingressi applicati negli istanti precedenti. Tali ingressi altro non sono che lo 'stato' del sistema che rappresenta in qualche modo la 'memoria' della macchina. Il progetto di macchine sequenziali avviene generalmente attraverso il modellamento di un automa a stati finiti di Mealy o di Moore rispettivamente se l'uscita dipende dallo stato e dall'ingresso o soltanto dallo stato.

### 3.1 Traccia

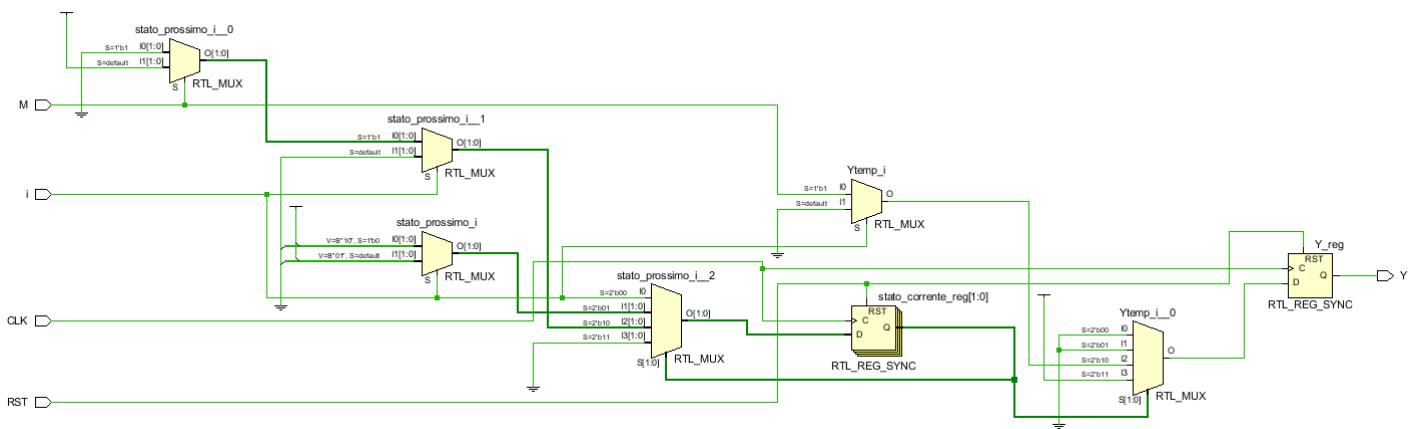
Progettare, implementare in VHDL e testare mediante simulazione una macchina in grado di riconoscere la sequenza 101. La macchina prende in ingresso un segnale binario  $i$  che rappresenta il dato, un segnale  $A$  di tempificazione e un segnale  $M$  di modo, che ne disciplina il funzionamento, e fornisce un'uscita  $Y$  alta quando la sequenza viene riconosciuta. In particolare,

- se  $M=0$ , la macchina valuta i bit seriali in ingresso a gruppi di 3 (sequenze non sovrapposte), se  $M=1$ , la macchina valuta i bit seriali in ingresso uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta (sequenze parzialmente sovrapposte).
- 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 tempificazione  $A$ , che deve essere ottenuto a partire dal clock della board. Infine, l'uscita  $Y$  può essere codificata utilizzando un led.

## 3.2 Automata



## 3.3 Progetto e architettura



## 3.4 Implementazione

Si é scelto di implementare una macchina di Mealy che riconosce una specifica sequenza di input ('i') e genera un uscita ('Y') in base alla sequenza fornita in ingresso. In questo caso particolare é stato inoltre necessario realizzare due automi distinti in quanto il segnale M funge da selettore di comportamento della macchina. La selezione di un opportuno M da parte dell'utente permetterá quindi di passare da un automa all'altro. Quando M=0, la macchina valuta i bit seriali in ingresso a gruppi di 3, quando M=1 invece i bit vengono valutati uno alla volta. Essendo inoltre il riconoscitore una macchina sequenziale, ha bisogno di un segnale di temporizzazione (CLK) che permette alla macchina di passare da uno stato all'altro ad ogni fronte di salita. A seguito della dichiarazione dell'entity, quindi dell'interfaccia del nostro sistema, si é deciso di adottare un approccio Behavioral in modo da poter scrivere una descrizione accurata per l'automa a stati finiti rappresentato in figura. In particolare si é deciso di adottare un pattern con due 'process' in modo creare un design quanto piú simile al modello fondamentale di Huffman che divide la parte di memorizzazione del sistema dalla parte elaborativa.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Riconoscitore_101 is
5     port (
```

```
6      i: in std_logic;
7
8      RST: in std_logic;
9
10     CLK: in std_logic;
11
12     M: in std_logic;
13
14     Y: out std_logic;
15
16     state: out std_logic_vector(3 downto 0)
17
18   );
19
20 end Riconoscitore_101;
21
22
23 architecture Behavioral of Riconoscitore_101 is
24
25 type stato is (S0, S1, S2, S3, S4, S5, S6, S7, S8);
26
27
28 signal stato_corrente: stato := S0;
29
30 signal stato_prossimo: stato;
31
32 signal Ytemp: std_logic;
33
34
35 begin
36
37
38 stato_uscita: process(i, stato_corrente)
39 begin
40
41
42 case stato_corrente is
43
44 when S0 =>
45
46     if (M='0') then
47
48         stato_prossimo <= S4;
49
50         Ytemp <= '0';
51
52     else
53
54
55 end if;
56
57
58     if (M='1') then
59
60         stato_prossimo <= S5;
61
62         Ytemp <= '1';
63
64     else
65
66
67 end if;
68
69
70     if (RST='1') then
71
72         stato_prossimo <= S0;
73
74         Ytemp <= '0';
75
76     else
77
78
79 end if;
80
81
82     if (CLK='1') then
83
84         stato_prossimo <= S6;
85
86         Ytemp <= '0';
87
88     else
89
90
91 end if;
92
93
94     if (M='0' and RST='0') then
95
96         stato_prossimo <= S1;
97
98         Ytemp <= '0';
99
100    else
101
102
103 end if;
104
105
106     if (CLK='0') then
107
108         stato_prossimo <= S2;
109
110         Ytemp <= '0';
111
112     else
113
114
115 end if;
116
117
118     if (M='1' and RST='0') then
119
120         stato_prossimo <= S3;
121
122         Ytemp <= '1';
123
124     else
125
126
127 end if;
128
129
130     if (CLK='1') then
131
132         stato_prossimo <= S7;
133
134         Ytemp <= '0';
135
136     else
137
138
139 end if;
140
141
142     if (M='0' and RST='0') then
143
144         stato_prossimo <= S8;
145
146         Ytemp <= '0';
147
148     else
149
150
151 end if;
152
153
154     if (CLK='0') then
155
156         stato_prossimo <= S0;
157
158         Ytemp <= '0';
159
160     else
161
162
163 end if;
164
165
166     if (M='1' and RST='0') then
167
168         stato_prossimo <= S6;
169
170         Ytemp <= '0';
171
172     else
173
174
175 end if;
176
177
178     if (CLK='1') then
179
180         stato_prossimo <= S5;
181
182         Ytemp <= '1';
183
184     else
185
186
187 end if;
188
189
190     if (M='0' and RST='0') then
191
192         stato_prossimo <= S4;
193
194         Ytemp <= '0';
195
196     else
197
198
199 end if;
200
201
202     if (CLK='0') then
203
204         stato_prossimo <= S3;
205
206         Ytemp <= '0';
207
208     else
209
210
211 end if;
212
213
214     if (M='1' and RST='0') then
215
216         stato_prossimo <= S2;
217
218         Ytemp <= '1';
219
220     else
221
222
223 end if;
224
225
226     if (CLK='1') then
227
228         stato_prossimo <= S1;
229
230         Ytemp <= '0';
231
232     else
233
234
235 end if;
236
237
238     if (M='0' and RST='0') then
239
240         stato_prossimo <= S0;
241
242         Ytemp <= '0';
243
244     else
245
246
247 end if;
248
249
250     if (CLK='0') then
251
252         stato_prossimo <= S0;
253
254         Ytemp <= '0';
255
256     else
257
258
259 end if;
260
261
262     if (M='1' and RST='0') then
263
264         stato_prossimo <= S0;
265
266         Ytemp <= '0';
267
268     else
269
270
271 end if;
272
273
274     if (CLK='1') then
275
276         stato_prossimo <= S0;
277
278         Ytemp <= '0';
279
280     else
281
282
283 end if;
284
285
286     if (M='0' and RST='0') then
287
288         stato_prossimo <= S0;
289
290         Ytemp <= '0';
291
292     else
293
294
295 end if;
296
297
298     if (CLK='0') then
299
300         stato_prossimo <= S0;
301
302         Ytemp <= '0';
303
304     else
305
306
307 end if;
308
309
310     if (M='1' and RST='0') then
311
312         stato_prossimo <= S0;
313
314         Ytemp <= '0';
315
316     else
317
318
319 end if;
320
321
322     if (CLK='1') then
323
324         stato_prossimo <= S0;
325
326         Ytemp <= '0';
327
328     else
329
330
331 end if;
332
333
334     if (M='0' and RST='0') then
335
336         stato_prossimo <= S0;
337
338         Ytemp <= '0';
339
340     else
341
342
343 end if;
344
345
346     if (CLK='0') then
347
348         stato_prossimo <= S0;
349
350         Ytemp <= '0';
351
352     else
353
354
355 end if;
356
357
358     if (M='1' and RST='0') then
359
360         stato_prossimo <= S0;
361
362         Ytemp <= '0';
363
364     else
365
366
367 end if;
368
369
370     if (CLK='1') then
371
372         stato_prossimo <= S0;
373
374         Ytemp <= '0';
375
376     else
377
378
379 end if;
380
381
382     if (M='0' and RST='0') then
383
384         stato_prossimo <= S0;
385
386         Ytemp <= '0';
387
388     else
389
390
391 end if;
392
393
394     if (CLK='0') then
395
396         stato_prossimo <= S0;
397
398         Ytemp <= '0';
399
400     else
401
402
403 end if;
404
405
406     if (M='1' and RST='0') then
407
408         stato_prossimo <= S0;
409
410         Ytemp <= '0';
411
412     else
413
414
415 end if;
416
417
418     if (CLK='1') then
419
420         stato_prossimo <= S0;
421
422         Ytemp <= '0';
423
424     else
425
426
427 end if;
428
429
430     if (M='0' and RST='0') then
431
432         stato_prossimo <= S0;
433
434         Ytemp <= '0';
435
436     else
437
438
439 end if;
440
441
442     if (CLK='0') then
443
444         stato_prossimo <= S0;
445
446         Ytemp <= '0';
447
448     else
449
450
451 end if;
452
453
454     if (M='1' and RST='0') then
455
456         stato_prossimo <= S0;
457
458         Ytemp <= '0';
459
460     else
461
462
463 end if;
464
465
466     if (CLK='1') then
467
468         stato_prossimo <= S0;
469
470         Ytemp <= '0';
471
472     else
473
474
475 end if;
476
477
478     if (M='0' and RST='0') then
479
480         stato_prossimo <= S0;
481
482         Ytemp <= '0';
483
484     else
485
486
487 end if;
488
489
490     if (CLK='0') then
491
492         stato_prossimo <= S0;
493
494         Ytemp <= '0';
495
496     else
497
498
499 end if;
500
501
502     if (M='1' and RST='0') then
503
504         stato_prossimo <= S0;
505
506         Ytemp <= '0';
507
508     else
509
510
511 end if;
512
513
514     if (CLK='1') then
515
516         stato_prossimo <= S0;
517
518         Ytemp <= '0';
519
520     else
521
522
523 end if;
524
525
526     if (M='0' and RST='0') then
527
528         stato_prossimo <= S0;
529
530         Ytemp <= '0';
531
532     else
533
534
535 end if;
536
537
538     if (CLK='0') then
539
540         stato_prossimo <= S0;
541
542         Ytemp <= '0';
543
544     else
545
546
547 end if;
548
549
550     if (M='1' and RST='0') then
551
552         stato_prossimo <= S0;
553
554         Ytemp <= '0';
555
556     else
557
558
559 end if;
560
561
562     if (CLK='1') then
563
564         stato_prossimo <= S0;
565
566         Ytemp <= '0';
567
568     else
569
570
571 end if;
572
573
574     if (M='0' and RST='0') then
575
576         stato_prossimo <= S0;
577
578         Ytemp <= '0';
579
580     else
581
582
583 end if;
584
585
586     if (CLK='0') then
587
588         stato_prossimo <= S0;
589
590         Ytemp <= '0';
591
592     else
593
594
595 end if;
596
597
598     if (M='1' and RST='0') then
599
600         stato_prossimo <= S0;
601
602         Ytemp <= '0';
603
604     else
605
606
607 end if;
608
609
610     if (CLK='1') then
611
612         stato_prossimo <= S0;
613
614         Ytemp <= '0';
615
616     else
617
618
619 end if;
620
621
622     if (M='0' and RST='0') then
623
624         stato_prossimo <= S0;
625
626         Ytemp <= '0';
627
628     else
629
630
631 end if;
632
633
634     if (CLK='0') then
635
636         stato_prossimo <= S0;
637
638         Ytemp <= '0';
639
640     else
641
642
643 end if;
644
645
646     if (M='1' and RST='0') then
647
648         stato_prossimo <= S0;
649
650         Ytemp <= '0';
651
652     else
653
654
655 end if;
656
657
658     if (CLK='1') then
659
660         stato_prossimo <= S0;
661
662         Ytemp <= '0';
663
664     else
665
666
667 end if;
668
669
670     if (M='0' and RST='0') then
671
672         stato_prossimo <= S0;
673
674         Ytemp <= '0';
675
676     else
677
678
679 end if;
680
681
682     if (CLK='0') then
683
684         stato_prossimo <= S0;
685
686         Ytemp <= '0';
687
688     else
689
690
691 end if;
692
693
694     if (M='1' and RST='0') then
695
696         stato_prossimo <= S0;
697
698         Ytemp <= '0';
699
700     else
701
702
703 end if;
704
705
706     if (CLK='1') then
707
708         stato_prossimo <= S0;
709
710         Ytemp <= '0';
711
712     else
713
714
715 end if;
716
717
718     if (M='0' and RST='0') then
719
720         stato_prossimo <= S0;
721
722         Ytemp <= '0';
723
724     else
725
726
727 end if;
728
729
730     if (CLK='0') then
731
732         stato_prossimo <= S0;
733
734         Ytemp <= '0';
735
736     else
737
738
739 end if;
740
741
742     if (M='1' and RST='0') then
743
744         stato_prossimo <= S0;
745
746         Ytemp <= '0';
747
748     else
749
750
751 end if;
752
753
754     if (CLK='1') then
755
756         stato_prossimo <= S0;
757
758         Ytemp <= '0';
759
760     else
761
762
763 end if;
764
765
766     if (M='0' and RST='0') then
767
768         stato_prossimo <= S0;
769
770         Ytemp <= '0';
771
772     else
773
774
775 end if;
776
777
778     if (CLK='0') then
779
780         stato_prossimo <= S0;
781
782         Ytemp <= '0';
783
784     else
785
786
787 end if;
788
789
790     if (M='1' and RST='0') then
791
792         stato_prossimo <= S0;
793
794         Ytemp <= '0';
795
796     else
797
798
799 end if;
800
801
802     if (CLK='1') then
803
804         stato_prossimo <= S0;
805
806         Ytemp <= '0';
807
808     else
809
810
811 end if;
812
813
814     if (M='0' and RST='0') then
815
816         stato_prossimo <= S0;
817
818         Ytemp <= '0';
819
820     else
821
822
823 end if;
824
825
826     if (CLK='0') then
827
828         stato_prossimo <= S0;
829
830         Ytemp <= '0';
831
832     else
833
834
835 end if;
836
837
838     if (M='1' and RST='0') then
839
840         stato_prossimo <= S0;
841
842         Ytemp <= '0';
843
844     else
845
846
847 end if;
848
849
850     if (CLK='1') then
851
852         stato_prossimo <= S0;
853
854         Ytemp <= '0';
855
856     else
857
858
859 end if;
860
861
862     if (M='0' and RST='0') then
863
864         stato_prossimo <= S0;
865
866         Ytemp <= '0';
867
868     else
869
870
871 end if;
872
873
874     if (CLK='0') then
875
876         stato_prossimo <= S0;
877
878         Ytemp <= '0';
879
880     else
881
882
883 end if;
884
885
886     if (M='1' and RST='0') then
887
888         stato_prossimo <= S0;
889
890         Ytemp <= '0';
891
892     else
893
894
895 end if;
896
897
898     if (CLK='1') then
899
900         stato_prossimo <= S0;
901
902         Ytemp <= '0';
903
904     else
905
906
907 end if;
908
909
910     if (M='0' and RST='0') then
911
912         stato_prossimo <= S0;
913
914         Ytemp <= '0';
915
916     else
917
918
919 end if;
920
921
922     if (CLK='0') then
923
924         stato_prossimo <= S0;
925
926         Ytemp <= '0';
927
928     else
929
930
931 end if;
932
933
934     if (M='1' and RST='0') then
935
936         stato_prossimo <= S0;
937
938         Ytemp <= '0';
939
940     else
941
942
943 end if;
944
945
946     if (CLK='1') then
947
948         stato_prossimo <= S0;
949
950         Ytemp <= '0';
951
952     else
953
954
955 end if;
956
957
958     if (M='0' and RST='0') then
959
960         stato_prossimo <= S0;
961
962         Ytemp <= '0';
963
964     else
965
966
967 end if;
968
969
970     if (CLK='0') then
971
972         stato_prossimo <= S0;
973
974         Ytemp <= '0';
975
976     else
977
978
979 end if;
980
981
982     if (M='1' and RST='0') then
983
984         stato_prossimo <= S0;
985
986         Ytemp <= '0';
987
988     else
989
990
991 end if;
992
993
994     if (CLK='1') then
995
996         stato_prossimo <= S0;
997
998         Ytemp <= '0';
999
1000 else
1001
1002
1003 end if;
1004
1005
1006 end process;
1007
1008
1009 end behavioral;
```

```
34         stato_prossimo <= S1;
35
36     end if;
37
38 when S1 =>
39
40     if(i='1') then
41
42         stato_prossimo <= S2;
43
44         Ytemp <= '0';
45
46     else
47
48         stato_prossimo <= S1;
49
50         Ytemp <= '0';
51
52     end if;
53
54
55 when S2 =>
56
57     if(i='1') then
58
59         stato_prossimo <= S2;
60
61         Ytemp <= '0';
62
63     else
64
65         stato_prossimo <= S3;
66
67         Ytemp <= '0';
68
69     end if;
70
71
72 when S3 =>
73
74     if(i='1') then
75
76         stato_prossimo <= S1;
77
78         Ytemp <= '1';
79
80     else
81
82         stato_prossimo <= S1;
```

```
62           Ytemp <= '0';
63       end if;
64
65   when S4 =>
66
67       if(i='1') then
68
69           stato_prossimo <= S5;
70
71           Ytemp <= '0';
72
73       else
74
75           stato_prossimo <= S6;
76
77           Ytemp <= '0';
78
79       end if;
80
81
82
83   when S5 =>
84
85       if(i='1') then
86
87           stato_prossimo <= S8;
88
89           Ytemp <= '0';
90
91       else
92
93           stato_prossimo <= S7;
94
95           Ytemp <= '0';
96
97       end if;
98
99
100
101  when S6 =>
102
103      Ytemp <= '0';
104
105      stato_prossimo <= S8;
106
107
108
109  when S7 =>
110
111      if(i='1') then
112
113          stato_prossimo <= S4;
```

```
90          Ytemp <= '1';
91      else
92          stato_prossimo <= S4;
93          Ytemp <= '0';
94      end if;
95
96      when S8 =>
97          Ytemp <= '0';
98          stato_prossimo <= S4;
99
100     end case;
101
102 end process;
103
104 -- Processo sequenziale per la memoria e l'uscita
105 -- sincronizzata con il clock
106 mem: process(CLK)
107 begin
108     if rising_edge(CLK) then
109         if RST = '1' then
110             stato_corrente <= S0;
111             Y <= '0';
112         else
113             stato_corrente <= stato_prossimo;
114             -- Sincronizza Y con il clock
115             Y <= Ytemp;
116         end if;
117     end if;
```

```

117      end process;

118

119      with stato_corrente select
120          state <= x"0" when S0,
121                  x"1" when S1,
122                  x"2" when S2,
123                  x"3" when S3,
124                  x"4" when S4,
125                  x"5" when S5,
126                  x"6" when S6,
127                  x"7" when S7,
128                  x"8" when S8,
129                  x"9" when others;
130
131  end Behavioral;

```

Si è definita una variabile di tipo enumerazione ('stato') per rappresentare gli stati della macchina ('S0','S1','S2','S3') e sono stati definiti i segnali per mantenere lo stato corrente e successivo. Nel primo process (combinatorio), indicato dalla albel 'stato\_uscita' è stata descritta l'evoluzione del sistema in base ai passaggi di stato dei due automi. Nel secondo process (sequenziale), indicato dalla label 'mem', si gestisce la memoria della macchina a stati e l'uscita sincronizzata con il clock. Se il segnale di reset (RST) è '1', inizializza lo stato corrente a S0 e l'uscita a '0'. Altrimenti, passa allo stato successivo sincronizzato con il fronte di salita del clock (rising\_edge(CLK)) e imposta l'uscita (Y)

al valore temporaneo ( $Y_{temp}$ ) calcolato nel processo combinatorio. A seguire un'implementazione seguendo il pattern a singolo process:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4
5 entity Riconoscitore_101 is
6     port( i: in std_logic;
7             RST, CLK, M: in std_logic;
8             Y: out std_logic
9     );
10 end Riconoscitore_101;
11
12 -- versione con 1 process:
13 architecture Behavioral of Riconoscitore_101 is
14
15 type stato is (S0, S1, S2, S3);
16 signal stato_corrente : stato := S0;
17 attribute fsm_encoding : string;
18 attribute fsm_encoding of stato_corrente : signal is
19     "one_hot";
20
21 begin
22     stato_uscita_mem: process(CLK)
23         begin
24             if rising_edge(CLK) then
25                 if ( RST = '1' ) then

```

```
26         stato_corrente <= S0;
27
28     else
29
30         if (M='1') then
31
32             case stato_corrente is
33
34                 when S0 =>
35
36                     if( i = '0' ) then
37
38                         stato_corrente <= S0;
39
40                         Y <= '0';
41
42                     end if;
43
44                 when S1 =>
45
46                     if( i = '0' ) then
47
48                         stato_corrente <= S2;
49
50                         Y <= '0';
51
52                     end if;
53
54                 when S2 =>
55
56                     if( i = '0' ) then
57
58                         stato_corrente <= S0;
59
60                         Y <= '0';
61
62                     else
63
64                         stato_corrente <= S0;
65
66                         Y <= '1';
67
```

```
54         end if;
55
56         when others =>
57             stato_corrente <= S0;
58             Y <= '0';
59
60         end case;
61
62     else
63
64         case stato_corrente is
65
66             when S0 =>
67                 if i = '1' then
68                     stato_corrente <= S1;
69                 else
70                     stato_corrente <= S0;
71
72             end if;
73
74             Y <= '0';
75
76             when S1 =>
77                 if i = '0' then
78                     stato_corrente <= S2;
79                 else
80                     stato_corrente <= S1;
81
82             end if;
83
84             Y <= '0';
85
86             when S2 =>
87                 if i = '1' then
88                     stato_corrente <= S3;
89                 else
90                     stato_corrente <= S0;
91
92             end if;
93
94             Y <= '0';
95
```

```

82      when S3 =>
83          -- Aggiungo uno stato infU+FFFpfr
84          -- gestire la non sovrapposizione.
85          stato_corrente <= S0; -- Torna allo
86          stato iniziale dopo aver
87          riconosciuto la sequenza.
88          Y <= '1'; -- Sequenza riconosciuta,
89          l'uscita diventa '1'.
90
91      when others =>
92          stato_corrente <= S0;
93          Y <= '0';
94
95      end case;
96
97      end if;
98
99      end if;
100
101  end process;
102
103
104
105  end Behavioral;

```

## 3.5 Simulazione

Si è proceduto alla simulazione della macchina attraverso una apposita test bench: L'architettura definisce un componente istanziato (Riconoscitore\_101) con le relative porte di input e output. Vengono dichiarati segnali per gli input (i, CLK, RST, M) e l'output (Y). Inoltre, viene definito il periodo del clock (CLK\_period); Il processo

'CLK\_process' genera un segnale di clock (CLK) con un periodo definito (CLK\_period). Il clock viene impostato prima a '0', poi dopo metà del periodo a '1', e questo ciclo continua indefinitamente. Questo processo simula il comportamento di un clock; Il processo 'stim\_proc' simula il comportamento di un ambiente di test. Inizializza gli input (i, CLK, RST, M) con valori predefiniti e successivamente applica una sequenza di stimoli per testare il componente.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Riconoscitore_101_TB is
5 end Riconoscitore_101_TB;
6
7 architecture Behavioral of Riconoscitore_101_TB is
8
9 COMPONENT Riconoscitore_101
10    PORT(
11        i : IN std_logic;
12        RST,CLK,M : IN std_logic;
13        Y : OUT std_logic;
14        state: OUT std_logic_vector(3 downto 0)
15    );
16 END COMPONENT;
17
18 --Inputs
19 signal i : std_logic := '0';
20 signal CLK : std_logic := '0';

```

```
21 signal RST : std_logic := '0';
22 signal M : std_logic := 'U';
23
24 --Outputs
25 signal Y : std_logic;
26 signal state: std_logic_vector(3 downto 0);
27
28 -- Clock period definitions
29 constant CLK_period : time := 10 ns;
30
31 BEGIN
32
33 uut: Riconoscitore_101 port map(
34     i => i,
35     CLK => CLK,
36     M => M,
37     RST => RST,
38     Y => Y,
39     state => state
40 );
41
42 -- Clock process definitions
43 CLK_process :process
44 begin
45     CLK <= '0';
46     wait for CLK_period/2;
47     CLK <= '1';
48     wait for CLK_period/2;
```

```
49      end process;  
50  
51  
52      -- Stimulus process  
53      stim_proc: process  
54      begin  
55          RST <= '1';  
56          wait for 100 ns;  
57  
58          RST <= '0';  
59  
60          M<='1';  
61  
62          wait for 10 ns;  
63  
64          i<='0';  
65          wait for 10 ns;  
66          i<='0';  
67          wait for 10 ns;  
68          i<='1';  
69          wait for 10 ns;  
70          i<='0';  
71          wait for 10 ns;  
72          i<='1';  
73          wait for 10 ns;  
74          i<='1';  
75          wait for 10 ns;  
76          i<='0';
```

```
77      wait for 10 ns;
78      i<='0';
79
80      M<='0';
81      wait for 10 ns;
82      RST<='1';
83      wait for 20 ns;
84      RST <= '0';
85
86      wait for 7.5 ns;
87
88      i<='0';
89      wait for 10 ns;
90      i<='0';
91      wait for 10 ns;
92      i<='1';
93      wait for 10 ns;
94      i<='1';
95      wait for 10 ns;
96      i<='0';
97      wait for 10 ns;
98      i<='1';
99      wait for 10 ns;
100     i<='0';
101     wait for 10 ns;
102     i<='1';
103     wait for 10 ns;
104     i<='0';
```

```

105
106
107     wait;
108 end process;
109
110 END;
```

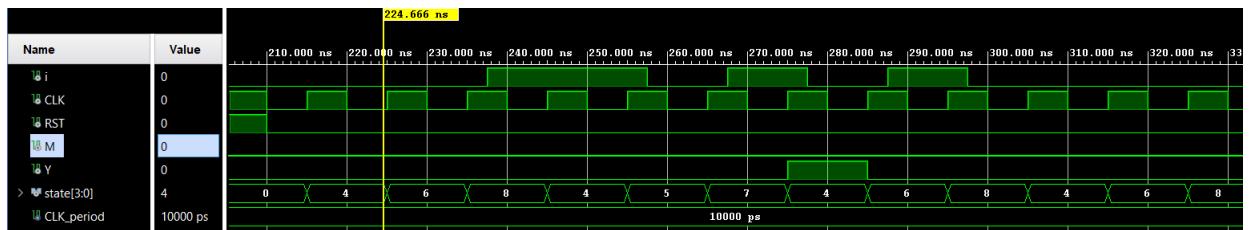


Figure 3.1: M=0

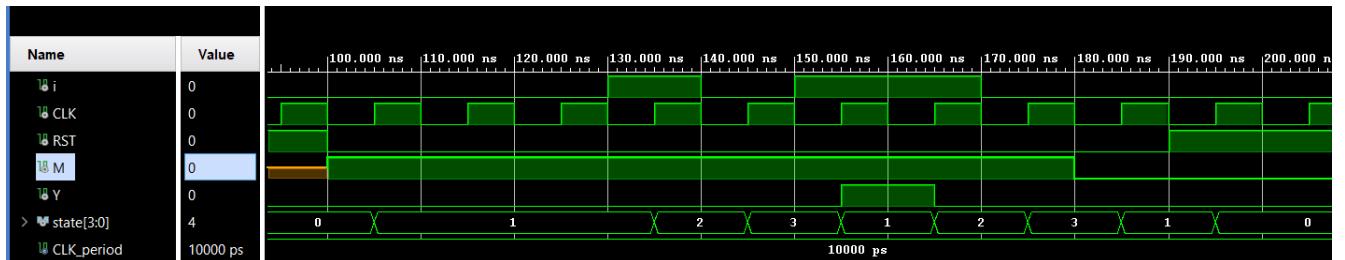


Figure 3.2: M=1

## 3.6 Sintesi su board di sviluppo

Nel processo di sintesi sulla scheda, abbiamo implementato due button debouncer per filtrare i segnali di lettura dell'ingresso e della modalità di funzionamento del riconoscitore di sequenza. Questo garantisce che i segnali siano puliti e privi di rumore indesiderato. Nei constraint fisici, abbiamo specificato che i primi due switch sono utilizzati per

l'ingresso i e per la modalità di funzionamento m rispettivamente.

Per quanto riguarda le indicazioni visive, il primo LED è utilizzato per indicare se la sequenza è stata riconosciuta dal riconoscitore. Gli LED restanti mostrano i vari stati modificati, fornendo un feedback visivo utile sia per l'utente che per il debugging.

Per quanto riguarda l'interfaccia utente, abbiamo associato il bottone di reset al pin N17. Il pin P17 è stato associato al bottone per la lettura della modalità di funzionamento del riconoscitore (m\_read), mentre il pin M17 è stato associato al bottone per la lettura dell'ingresso (i\_read). Questa configurazione consente un controllo intuitivo e diretto del riconoscitore di sequenza attraverso l'interfaccia utente della scheda

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Riconoscitore_onBoard is
5     port (
6         RST: in std_logic;
7         CLK: in std_logic;
8         i: in std_logic;
9         i_read: in std_logic;
10        M: in std_logic;
11        m_read: in std_logic;
12        Y: out std_logic;
13        state: out std_logic_vector(3 downto 0)
14    );

```

```
15  end Riconoscitore_onBoard;  
16  
17  architecture Structural of Riconoscitore_onBoard is  
18  
19  component ButtonDebouncer is  
20      generic(  
21          CLK_period: integer := 10;  
22          btn_noise_time: integer := 10000000  
23      );  
24      port(  
25          RST: in std_logic;  
26          CLK: in std_logic;  
27          BTN: in std_logic;  
28          CLEARED_BTN: out std_logic -- segnale di  
29              output ripulito  
30      );  
31  
32  end component;  
33  
34  component Riconoscitore_101 is  
35      port(  
36          i: in std_logic;  
37          RST: in std_logic;  
38          CLK: in std_logic;  
39          M: in std_logic;  
40          i_read: in std_logic;  
41          M_read: in std_logic;  
42          Y: out std_logic;  
43          state: out std_logic_vector(3 downto 0)
```

```
42      );
43  end component;
44
45
46  signal cleared_i: std_logic;
47  signal cleared_m: std_logic;
48
49 begin
50
51  deb_i: ButtonDebouncer generic map(
52      CLK_period => 10,
53      btn_noise_time => 10000000
54  )
55  port map(
56      RST => RST,
57      CLK => CLK,
58      BTN => i_read,
59      CLEARED_BTN => cleared_i
60  );
61
62  deb_m: ButtonDebouncer generic map(
63      CLK_period => 10,
64      btn_noise_time => 10000000
65  )
66  port map(
67      RST => RST,
68      CLK => CLK,
69      BTN => m_read,
```

```

70          CLEARED_BTN => cleared_m
71      );
72
73  ric: Riconoscitore_101 port map(
74      i => i,
75      CLK => CLK,
76      RST => RST,
77      M => M,
78      i_read => cleared_i,
79      m_read => cleared_m,
80      Y => Y,
81      state => state
82  );
83
84 end Structural;

```

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Riconoscitore_101 is
5     port (
6         i: in std_logic;
7         RST: in std_logic;
8         CLK: in std_logic;
9         M: in std_logic;
10        i_read: in std_logic;
11        M_read: in std_logic;
12        Y: out std_logic;
13        state: out std_logic_vector(3 downto 0)

```

```
14      );
15 end Riconoscitore_101;
16
17 architecture Behavioral of Riconoscitore_101 is
18
19 type stato is (S0, S1, S2, S3, S4, S5, S6, S7, S8);
20
21 signal stato_corrente: stato := S0;
22 signal stato_prossimo: stato;
23 signal Ytemp: std_logic;
24
25 begin
26
27     stato_uscita: process(i, stato_corrente)
28 begin
29
30     case stato_corrente is
31         when S0 =>
32             if (M='0') then
33                 stato_prossimo <= S4;
34                 Ytemp <= '0';
35             else
36                 stato_prossimo <= S1;
37                 Ytemp <= '0';
38             end if;
39
40         when S1 =>
41             if (i='1') then
```

```
42         stato_prossimo <= S2;
43
44     else
45
46         stato_prossimo <= S1;
47
48         Ytemp <= '0';
49
50     end if;
51
52
53 when S2 =>
54
55     if(i='1') then
56
57         stato_prossimo <= S2;
58
59         Ytemp <= '0';
60
61     else
62
63         stato_prossimo <= S3;
64
65         Ytemp <= '0';
66
67     end if;
68
69
70 when S3 =>
71
72     if(i='1') then
73
74         stato_prossimo <= S1;
75
76         Ytemp <= '1';
77
78     else
79
80         stato_prossimo <= S1;
81
82         Ytemp <= '0';
83
84     end if;
85
86
87 when S4 =>
88
89     if(i='1') then
90
91         stato_prossimo <= S5;
```

```
70          Ytemp <= '0';
71
72      else
73
74          stato_prossimo <= S6;
75
76          Ytemp <= '0';
77
78      end if;
79
80
81      when S5 =>
82
83          if (i='1') then
84
85              stato_prossimo <= S8;
86
87              Ytemp <= '0';
88
89          else
90
91              stato_prossimo <= S7;
92
93              Ytemp <= '0';
94
95          end if;
96
97
98      when S6 =>
99
100
101         Ytemp <= '0';
102
103         stato_prossimo <= S8;
104
105
106
107      when S7 =>
108
109          if (i='1') then
110
111              stato_prossimo <= S4;
112
113              Ytemp <= '1';
114
115          else
116
117              stato_prossimo <= S4;
118
119              Ytemp <= '0';
120
121          end if;
122
123
124
```

```
98      when S8 =>
99          Ytemp <= '0';
100         stato_prossimo <= S4;
101
102     end case;
103
104 end process;
105
106 -- Processo sequenziale per la memoria e l'uscita
107     sincronizzata con il clock
108
109 mem: process(CLK)
110 begin
111     if rising_edge(CLK) then
112         if RST = '1' then
113             stato_corrente <= S0;
114             Y <= '0';
115         else
116             if(stato_corrente = S0 and m_read='1')
117                 then
118                     stato_corrente <= stato_prossimo;
119                     Y <= Ytemp;
120                 elsif(stato_corrente /= S0 and
121                     i_read='1') then
122                     stato_corrente <= stato_prossimo;
123                     Y <= Ytemp;
124                 end if;
125             end if;
126         end if;
```

```

123      end process;

124

125      with stato_corrente select
126          state <= x"0" when S0,
127              x"1" when S1,
128              x"2" when S2,
129              x"3" when S3,
130              x"4" when S4,
131              x"5" when S5,
132              x"6" when S6,
133              x"7" when S7,
134              x"8" when S8,
135              x"9" when others;
136
137  end Behavioral;
```

```

1 ## Clock signal
2 set_property -dict { PACKAGE_PIN E3      IO_STANDARD
3                         LVCMOS33 } [get_ports { CLK }]; #IO_L12P_T1_MRCC_35
4                         Sch=clk100mhz
5
6 create_clock -add -name sys_clk_pin -period 10.00
7                         -waveform {0 5} [get_ports {CLK}];
8
9
10 ##Switches
11 set_property -dict { PACKAGE_PIN J15      IO_STANDARD
12                         LVCMOS33 } [get_ports { i }]; #IO_L24N_T3_RS0_15
13                         Sch=sw[0]
14
15 set_property -dict { PACKAGE_PIN L16      IO_STANDARD
```

```

    LVCMOS33 } [get_ports { M }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]

9
10 ## LEDs
11 set_property -dict { PACKAGE_PIN H17      IOSTANDARD
    LVCMOS33 } [get_ports { Y }];
#IO_L18P_T2_A24_15
    Sch=led[0]
12 #set_property -dict { PACKAGE_PIN K15      IOSTANDARD
    LVCMOS33 } [get_ports { LED[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
13 #set_property -dict { PACKAGE_PIN J13      IOSTANDARD
    LVCMOS33 } [get_ports { LED[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
14 #set_property -dict { PACKAGE_PIN N14      IOSTANDARD
    LVCMOS33 } [get_ports { LED[3] }];
#IO_L8P_T1_D11_14
    Sch=led[3]
15 #set_property -dict { PACKAGE_PIN R18      IOSTANDARD
    LVCMOS33 } [get_ports { LED[4] }];
#IO_L7P_T1_D09_14
    Sch=led[4]
16 #set_property -dict { PACKAGE_PIN V17      IOSTANDARD
    LVCMOS33 } [get_ports { LED[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
17 #set_property -dict { PACKAGE_PIN U17      IOSTANDARD
    LVCMOS33 } [get_ports { LED[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
18 #set_property -dict { PACKAGE_PIN U16      IOSTANDARD
    LVCMOS33 } [get_ports { LED[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]

```

```

19 #set_property -dict { PACKAGE_PIN V16      IOSTANDARD
20   LVCMOS33 } [get_ports { LED[8] }];
21   #IO_L16N_T2_A15_D31_14 Sch=led[8]
22
23 #set_property -dict { PACKAGE_PIN T15      IOSTANDARD
24   LVCMOS33 } [get_ports { LED[9] }];
25   #IO_L14N_T2_SRCC_14 Sch=led[9]
26
27 #set_property -dict { PACKAGE_PIN U14      IOSTANDARD
28   LVCMOS33 } [get_ports { LED[10] }];
29   #IO_L22P_T3_A05_D21_14 Sch=led[10]
30
31 #set_property -dict { PACKAGE_PIN T16      IOSTANDARD
32   LVCMOS33 } [get_ports { LED[11] }];
33   #IO_L15N_T2_DQS_DOUT_CS0_B_14 Sch=led[11]
34
35 set_property -dict { PACKAGE_PIN V15      IOSTANDARD
36   LVCMOS33 } [get_ports { state[0] }];
37   #IO_L16P_T2_CSI_B_14 Sch=led[12]
38
39 set_property -dict { PACKAGE_PIN V14      IOSTANDARD
40   LVCMOS33 } [get_ports { state[1] }];
41   #IO_L22N_T3_A04_D20_14 Sch=led[13]
42
43 set_property -dict { PACKAGE_PIN V12      IOSTANDARD
44   LVCMOS33 } [get_ports { state[2] }];
45   #IO_L20N_T3_A07_D23_14 Sch=led[14]
46
47 set_property -dict { PACKAGE_PIN V11      IOSTANDARD
48   LVCMOS33 } [get_ports { state[3] }];
49   #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
50
51
52 ##Buttons
53
54 #set_property -dict { PACKAGE_PIN C12      IOSTANDARD
55   LVCMOS33 } [get_ports { CPU_RESETN }];

```

```
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn  
30 set_property -dict { PACKAGE_PIN N17      IOSTANDARD  
                  LVCMOS33 } [get_ports { RST }]; #IO_L9P_T1_DQS_14  
                                         Sch=btnc  
31 #set_property -dict { PACKAGE_PIN M18      IOSTANDARD  
                  LVCMOS33 } [get_ports { BTNU }]; #IO_L4N_T0_D05_14  
                                         Sch=btnu  
32 set_property -dict { PACKAGE_PIN P17      IOSTANDARD  
                  LVCMOS33 } [get_ports { m_read }];  
#IO_L12P_T1_MRCC_14 Sch=btnl  
33 set_property -dict { PACKAGE_PIN M17      IOSTANDARD  
                  LVCMOS33 } [get_ports { i_read }];  
#IO_L10N_T1_D15_14 Sch=btnr  
34 #set_property -dict { PACKAGE_PIN P18      IOSTANDARD  
                  LVCMOS33 } [get_ports { BTND }];  
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```

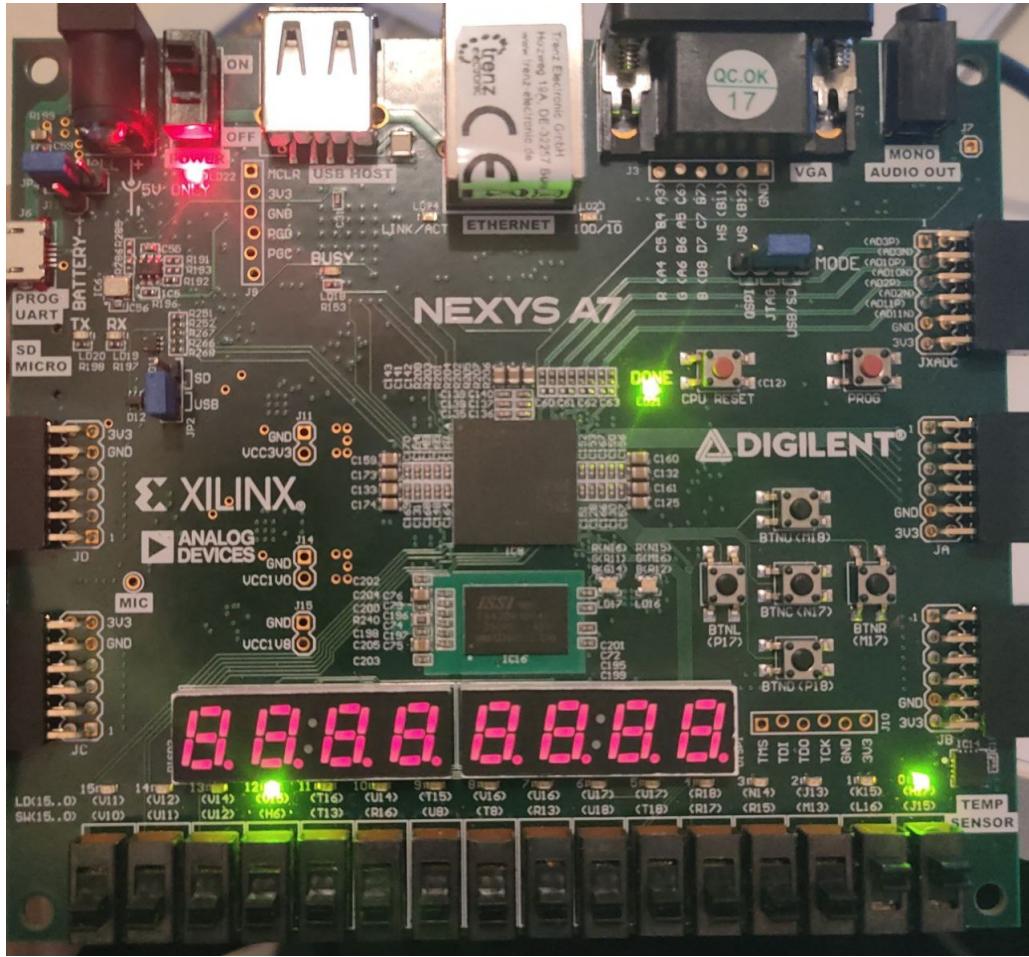


Figure 3.3: Esempio di sequenza riconosciuta

### 3.7 Timing Analysis

L’analisi della tempificazione può essere fatta a valle della sintesi ma in tal caso i valori saranno delle stime abbastanza approssimate. I parametri di configurazione della timing analysis devono essere opportunamente settati. Per verificare quale sia la frequenza massima di funzionamento (FMAX) di un design è possibile diminuire progressivamente la frequenza del clock del design ed eseguire una timing analysis finché non si ottiene uno slack negativo (worst negative slack). Prima

The screenshot shows the Vivado interface with the following details:

- Project Summary**: Shows the current project is "Riconoscitore\_onBoard.vhd".
- Device**: Shows the device is "Nexys-A7-50T-Master.xdc".
- File Path**: C:/MIVADO/timing analysis riconoscitore/timing analysis riconoscitore.srcts/\_constrs\_1/imports/digilent-xdc-master/Nexys-A7-50T-Master.xdc
- Tool Bar**: Includes icons for search, file, zoom, and various tools.
- Text Editor**: Displays the XDC constraint file content, defining a clock signal and switches.
- Timing Analysis Results**:
  - Tcl Console**, **Messages**, **Log**, **Reports**, **Design Runs**, **DRC**, **Methodology**, **Power**, **Timing** (selected).
  - Design Timing Summary**:
 

Setup			Hold			Pulse Width		
Worst Negative Slack (WNS):	4,660 ns		Worst Hold Slack (WHS):	0,212 ns		Worst Pulse Width Slack (WPWS):	4,500 ns	
Total Negative Slack (TNS):	0,000 ns		Total Hold Slack (THS):	0,000 ns		Total Pulse Width Negative Slack (TPWNS):	0,000 ns	
Number of Failing Endpoints:	0		Number of Failing Endpoints:	0		Number of Failing Endpoints:	0	
Total Number of Endpoints:	215		Total Number of Endpoints:	215		Total Number of Endpoints:	82	
  - Message**: All user specified timing constraints are met.

di tutto è stata effettuata una timing analysis basandosi sulla frequenza di base dell'oscillatore presente sulla board di sviluppo. Si è specificato nel file .xdc che definisce i constraint del design in questione il periodo del clock, denominato "sys\_clk\_pin", ed in quanti istanti avvengono i fronti di salita e di discesa del segnale, rispettivamente pari a 10ns e (0 5) (figura: 3.7).

Si è scelto di visualizzare il "Report Timing Summary" incrementando il numero massimo di percorsi negativi per ogni endpoint da 1 a 10, in modo da avere una visione di insieme completa. Si procede quindi ad analizzare il summary (figura 3.7)

Si nota che il Worst Negative Slack è un valore abbastanza alto avendo inserito 10ns come periodo del clock. Per questo motivo è pre-

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	4.660	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[1]/R	4.573	0.918	3.655	10.0
Path 2	4.660	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[2]/R	4.573	0.918	3.655	10.0
Path 3	4.660	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[3]/R	4.573	0.918	3.655	10.0
Path 4	4.660	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[4]/R	4.573	0.918	3.655	10.0
Path 5	4.709	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[29]/R	4.494	0.918	3.576	10.0
Path 6	4.709	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[30]/R	4.494	0.918	3.576	10.0
Path 7	4.709	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[31]/R	4.494	0.918	3.576	10.0
Path 8	4.791	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[5]/R	4.406	0.918	3.488	10.0
Path 9	4.791	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[6]/R	4.406	0.918	3.488	10.0
Path 10	4.791	3		31 deb_m/count_reg[2]/C	deb_m/count_reg[7]/R	4.406	0.918	3.488	10.0

sente ancora del margine miglioramento per la frequenza di lavoro del design. Il Total Negative Slack pari a 0.00ns indica che tutti i timing constraint specificati dall' utente sono rispettati, così come viene segnalato dalla scritta in figura 3.7.

Sotto la dicitura "Check Timing" (fig. 3.7) si nota la presenza di diversi elementi marcati in giallo con un livello di severity elevato. Tali avvertimenti stanno ad indicare la presenza di input e output (nel nostro caso) per i quali non sono stati specificati timing constraint.

Selezionando la voce "Intra-Clock Path" è possibile visualizzare tutti i percorsi tra due celle sequenziali governate dallo stesso clock,

The screenshot shows the Vivado interface with the following tabs at the top: Project Summary, Device, Riconoscitore\_onBoard.vhd, and Nexys-A7-50T-Master.xdc. Below the tabs is a toolbar with various icons. The main area displays a script or constraint file with code related to clock signals and switches. Below this is a table titled "Intra-Clock Paths - sys\_clk\_pin - Setup" showing timing analysis results for 10 paths. The table includes columns for Name, Slack, Levels, High Fanout, From, To, Total Delay, Logic Delay, Net Delay, and Requirement.

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	0.244	7	2	deb_m/count_reg[1]C	deb_m/count_reg[25]D	3.433	1.904	1.529	4.0
Path 2	0.354	3	31	deb_m/count_reg[7]C	deb_m/count_reg[13]R	3.163	0.828	2.335	4.0
Path 3	0.354	3	31	deb_m/count_reg[7]C	deb_m/count_reg[15]R	3.163	0.828	2.335	4.0
Path 4	0.354	3	31	deb_m/count_reg[7]C	deb_m/count_reg[16]R	3.163	0.828	2.335	4.0
Path 5	0.354	3	31	deb_m/count_reg[7]C	deb_m/count_reg[17]R	3.163	0.828	2.335	4.0
Path 6	0.354	3	31	deb_m/count_reg[7]C	deb_m/count_reg[18]R	3.163	0.828	2.335	4.0
Path 7	0.354	3	31	deb_m/count_reg[7]C	deb_m/count_reg[19]R	3.163	0.828	2.335	4.0
Path 8	0.354	3	31	deb_m/count_reg[7]C	deb_m/count_reg[20]R	3.163	0.828	2.335	4.0
Path 9	0.426	7	2	deb_m/count_reg[1]C	deb_m/count_reg[27]D	3.297	1.921	1.376	4.0
Path 10	0.441	3	31	deb_i/count_reg[11]C	deb_i/count_reg[26]R	2.962	0.890	2.072	4.0

nel nostro caso "sys\_clk\_pin" definito nel file constraint. E' possibile vedere il valore dello slack per ogni percorso da una determinata sorgente ad una destinazione. Visto che lo slack risulta positivo si è provato a modificare il constraint temporale selezionando un periodo pari a 4ns con una waveform pari a (0 2) al fine di ottimizzare la tempificazione e di conseguenza il lavoro del design:

Dal Design Timing Summary si evince che il Worst Negative Slack risulta ancora positivo anche se molto piccolo. Ciò si traduce in un margine ancora disponibile ma molto piccolo. Si può anche notare quanto sia diminuito lo Slack per ogni percorso tra le celle:

Si è quindi provato ad ottimizzare ancora di più il design selezionando un periodo del clock pari a 3.5 ed una relativa waveform (0 1.75):

## CHAPTER 3. RICONOSCITORE DI SEQUENZE 101

The screenshot shows the Vivado IDE interface. The top bar displays "Project Summary" and "Nexys-A7-50T-Master.xdc". Below the top bar, the code editor shows a portion of the Verilog file containing timing constraints:

```

5
6 ## Clock signal
7 set_property -dict { PACKAGE_PIN E3 IO_STANDARD LVCMOS33 } [get_ports { CLK }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
8 create_clock -add -name sys_clk_pin -period 3.5 -waveform {0 1.75} [get_ports { CLK }];
9
10
11 ##Switches
12 set_property -dict { PACKAGE_PIN J15 IO_STANDARD LVCMOS33 } [get_ports { i }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
<

```

The "Timing" tab is selected in the bottom navigation bar. The "Design Timing Summary" section shows the following results:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	<b>-0.138 ns</b>	Worst Hold Slack (WHS):	<b>0.239 ns</b>	Worst Pulse Width Slack (WPWS):	<b>1.250 ns</b>
Total Negative Slack (TNS):	<b>-4.118 ns</b>	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWWS):	0.000 ns
Number of Failing Endpoints:	<b>54</b>	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	221	Total Number of Endpoints:	221	Total Number of Endpoints:	84

A note at the bottom states: "Timing constraints are not met."

Dal Design Timing Summary si evince che il Worst Negative Slack è negativo ed il Total Negative Slack (pari alla somma di tutti gli slack) è ancora negativo ed abbastanza elevato. Ciò significa che ci sarà più di un percorso con slack negativo la cui somma sarà proprio -4.118ns:

Nel caso di questo specifico design è possibile spingersi ad un periodo massimo di 4ns (volendo usare numeri interi) in quanto selezionando 3ns non è possibile soddisfare i requisiti descritti dai timing constraint.

The screenshot shows the Vivado IDE interface with the "Intra-Clock Paths - sys\_clk\_pin - Setup" table selected. The table details 10 paths from the system clock pin to various deb\_i/count\_reg components. The columns include Name, Slack, Levels, High Fanout, From, To, Total Delay, Logic Delay, Net Delay, Requirement, and Status.

Name	Slack ^ 1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Status
Path 1	<b>-0.138</b>	8	2	deb_i/count_reg[2]/C	deb_i/count_reg[30]/D	3.348	2.148	1.200	3.5	s
Path 2	<b>-0.133</b>	6	2	deb_i/count_reg[2]/C	deb_i/count_reg[22]/D	3.352	1.920	1.432	3.5	s
Path 3	<b>-0.132</b>	7	2	deb_i/count_reg[2]/C	deb_i/count_reg[28]/D	3.352	2.013	1.339	3.5	s
Path 4	<b>-0.128</b>	3	32	deb_i/count_reg[18]/C	deb_i/count_reg[6]/R	3.050	0.828	2.222	3.5	s
Path 5	<b>-0.124</b>	3	32	deb_i/count_reg[18]/C	deb_i/count_reg[22]/R	3.137	0.828	2.309	3.5	s
Path 6	<b>-0.124</b>	3	32	deb_i/count_reg[18]/C	deb_i/count_reg[25]/R	3.137	0.828	2.309	3.5	s
Path 7	<b>-0.124</b>	3	32	deb_i/count_reg[18]/C	deb_i/count_reg[27]/R	3.137	0.828	2.309	3.5	s
Path 8	<b>-0.124</b>	3	32	deb_i/count_reg[18]/C	deb_i/count_reg[29]/R	3.137	0.828	2.309	3.5	s
Path 9	<b>-0.124</b>	3	32	deb_i/count_reg[18]/C	deb_i/count_reg[3]/R	3.137	0.828	2.309	3.5	s
Path 10	<b>-0.120</b>	3	32	deb_i/count_reg[18]/C	deb_i/count_reg[12]/R	3.132	0.828	2.304	3.5	s

# Chapter 4

## Shift Register

### 4.1 Traccia

- Progettare, implementare in VHDL e testare mediante simulazione un registro a scorrimento di N bit in grado di shiftare a destra o a sinistra di un numero Y variabile di posizioni a seconda di una opportuna selezione. In particolare, i valori possibili di Y sono 1 e 2. L'utente tramite selezione deve scegliere di quante posizioni shiftare. Il componente deve essere realizzato utilizzando sia un a) approccio comportamentale sia un b) approccio strutturale.
- Nota: il numero di bit del registro deve essere implementato come un generic, e dall'esterno deve poter essere scelta la modalità di funzionamento mediante opportuni segnali di selezione.

## 4.2 Progetto e architettura

Per quanto riguarda l'approccio comportamentale, la soluzione del problema è stata relativamente semplice: abbiamo iniziato dalle equazioni che governano il componente e le abbiamo implementate in uno script semplice contenuto all'interno di un process. Il comportamento della macchina è in questo caso determinato dagli ingressi di selezione che permettono di scegliere la direzione e l'ampiezza dello shift e che consentono, in corrispondenza di un ingresso, di precaricare un valore nel registro. Il componente è stato realizzato con un comportamento sincrono, quindi è abilitato esclusivamente in corrispondenza dei fronti di salita del clock.

L'approccio strutturale, d'altra parte, è stato più complesso da progettare. Abbiamo iniziato dall'implementazione del singolo flip-flop D e poi abbiamo collegato tutti i flip flop utilizzati attraverso una rete di interconnessione composta da multiplexer. Questo ci ha permesso di specificare, attraverso gli appositi ingressi di selezione, le particolari interconnessioni e quindi la direzione e l'ampiezza dello shift. Abbiamo poi dovuto affrontare un ulteriore problema per il caricamento parallelo dei dati, introducendo un meccanismo addizionale di selezione.

Il design realizzato prevede la creazione di uno shift register in grado di effettuare operazioni di shift sia a destra che a sinistra, di 1 o 2 posizioni, in base alle scelte dell'utente.

### 4.3 Implementazione: approccio behavioral

L’interfaccia dello shift register include un clock, un segnale di reset, un segnale di load per il caricamento del vettore di dati in ingressi, un segnale di selezione e un vettore di dati in uscita. Il segnale di selezione sel è particolarmente importante poiché determina il comportamento del registro a scorrimento. A seconda del suo valore, il registro esegue uno shift a sinistra o a destra di uno o due bit.

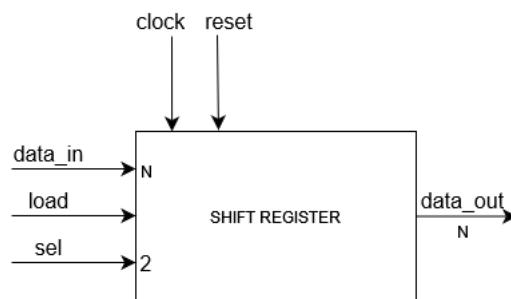


Figure 4.1: Interfaccia Shift Register da N bit

Abbiamo utilizzato il costrutto process per svolgere l’operazione di shift o di load sul fronte di salita del clock. Inoltre è stato anche implementato un reset sincrono con il clock. Infine l’uscita del registro viene aggiornata al di fuori dal process, così da forzarne continuamente l’assegnazione concorrente. Questo comportamento assicura che l’uscita rifletta sempre lo stato attuale del registro, consentendo una risposta rapida e accurata alle variazioni dei segnali di ingresso.

```

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

```

```
5      generic(
6          N: integer := 8 -- Numero di bit del registro
7      );
8
9      port (
10         clk: in std_logic;
11         reset: in std_logic;
12         load: in std_logic;
13         data_in: in std_logic_vector(N-1 downto 0);
14         sel: in std_logic_vector(1 downto 0); --
15             Segnale di selezione per lo shift
16         data_out: out std_logic_vector(N-1 downto 0)
17     );
18
19 end shift_register;
20
21
22 architecture Behavioral of shift_register is
23     signal temp_data: std_logic_vector(N-1 downto 0);
24
25 begin
26
27     proc: process(clk)
28     begin
29         if rising_edge(clk) then
30             if reset = '1' then
31                 temp_data <= (others => '0');
32             else
33                 if load='1' then
34                     temp_data <= data_in;
35                 else
36                     case sel is
```

```

32           when "00" => -- Shift a
33               sinistra di una posizione
34               temp_data <= temp_data(N-2
35                   downto 0) & '0';
36           when "01" => -- Shift a destra
37               di una posizione
38               temp_data <= '0' &
39                   temp_data(N-1 downto 1);
40           when "10" => -- Shift a
41               sinistra di due posizioni
42               temp_data <= temp_data(N-3
43                   downto 0) & "00";
44           when "11" => -- Shift a destra
45               di due posizioni
46               temp_data <= "00" &
47                   temp_data(N-1 downto 2);
48           when others =>
49               temp_data <= temp_data; --
50               Nessun cambiamento
51       end case;
52   end if;
53 end if;
54 end if;
55 end process;
56
57 data_out <= temp_data;
58
59
60 end Behavioral;

```

## 4.4 Implementazione: approccio structural

L'architettura strutturale dello shift register è costruita attorno ai flip flop D, che operano in modalità edge triggered sul fronte di salita del clock. Ogni flip flop D è associato a due multiplexer. Il primo multiplexer è un MUX 4:1, che svolge il ruolo di selezionare gli ingressi dei flip flop. Questo MUX 4:1 consente le diverse modalità di shift, a seconda della selezione effettuata. Questo riflette il comportamento osservato nell'approccio comportamentale. Il secondo multiplexer è un MUX 2:1. Questo prende in ingresso l'uscita del MUX 4:1 e il valore i-esimo dell'input. Questo MUX 2:1 serve a caricare i dati iniziali quando il segnale di selezione è alto. In sintesi, la combinazione di flip flop D e due livelli di multiplexer per ogni flip flop consente di implementare un registro a scorrimento con un comportamento configurabile. Questo design strutturale offre una grande flessibilità, permettendo di adattare facilmente il comportamento dello shift register alle esigenze specifiche dell'applicazione.

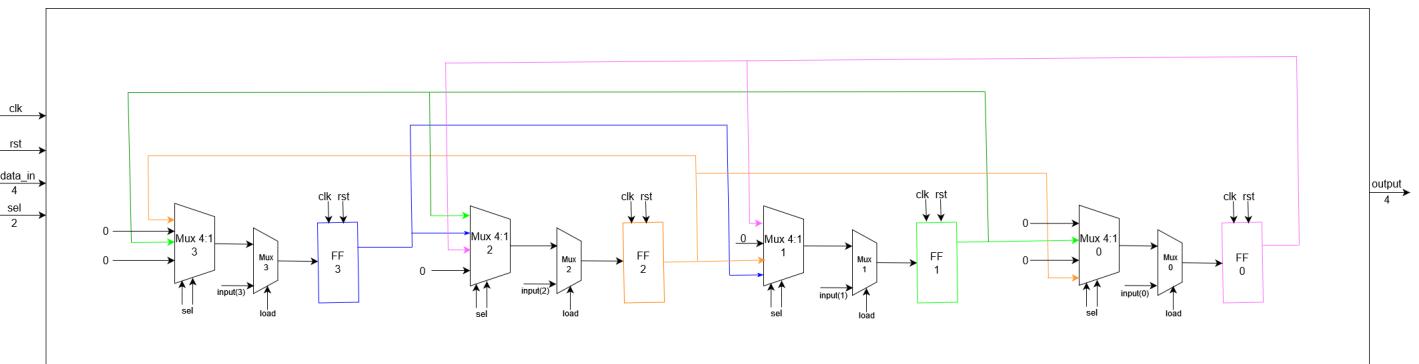


Figure 4.2: Shift Register Strutturale da N bit

Nel progetto del registro a scorrimento, abbiamo definito specifici segnali per gestire le interconnessioni tra le varie componenti:

- `out_FFD`: Questo segnale mantiene i valori di output dei flip flop. Questi valori sono utilizzati sia per visualizzare l'output del registro a scorrimento, sia come ingressi per i MUX 4:1 per gestire le operazioni di shift.
- `out_mux4`: Questo segnale mantiene i valori di output dei MUX 4:1. Questi valori vengono poi utilizzati come ingressi per i MUX 2:1, insieme al bit  $i$ -esimo del vettore di dati in ingresso.
- `out_mux2`: Questo segnale mantiene i valori che saranno poi utilizzati come ingressi per i flip flop.

Per facilitare l'interconnessione tra le uscite dei flip flop e i multiplexer, e per gestire l'implementazione con parametri generici, abbiamo utilizzato dei costrutti for-generate nel codice. Questi costrutti permettono di creare facilmente e velocemente le connessioni necessarie, rendendo il codice più leggibile e manutenibile.

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity shift_register is
6
7 generic(N: integer := 4);
```

```
8  port (
9      clk      : in std_logic;
10     reset    : in std_logic;
11     load     : in std_logic;
12     input    : in std_logic_vector(N-1 downto 0);
13     sel      : in std_logic_vector(1 downto 0);
14     output   : out std_logic_vector(N-1 downto 0)
15 );
16
17 end shift_register;
18
19 architecture Structural of shift_register is
20     signal out_mux4: std_logic_vector(N-1 downto 0);
21     signal out_mux2: std_logic_vector(N-1 downto 0);
22     signal out_FFD: std_logic_vector(N-1 downto 0);
23
24 component FF_D
25     port(
26         clk, reset, d: in std_logic;
27         y: out std_logic
28     );
29 end component;
30
31 component mux_2_1
32     port(
33         A : in std_logic_vector(1 downto 0);
34         S : in std_logic;
35         Y : out std_logic
```

```
36      );
37
38
39 component mux_4_1
40
41     port (
42         A: in std_logic_vector(3 downto 0);
43         S: in std_logic_vector(1 downto 0);
44         Y: out std_logic
45     );
46
47 begin
48
49     --Primo componente (N-1)
50     Mux4_Primeo: mux_4_1
51
52         port map(
53             A(0) => out_FFD(N-2),
54             A(1) => '0',
55             A(2) => out_FFD(N-3),
56             A(3) => '0',
57             S => sel,
58             Y => out_mux4(N-1)
59         );
60
61     Mux2_Primeo: mux_2_1
62
63         port map(
64             A(0) => out_mux4(N-1),
65             A(1) => input(N-1),
```

```
64          s => load,
65          y => out_mux2(N-1)
66      );
67
68 FF_Primo: FF_D
69     port map(
70         clk => clk,
71         reset => reset,
72         d => out_mux2(N-1),
73         y => out_FFD(N-1)
74     );
75
76 --Secondo componente (N-2)
77 Mux4_Secondo: mux_4_1
78     port map(
79         A(0) => out_FFD(N-3),
80         A(1) => out_FFD(N-1),
81         A(2) => out_FFD(N-4),
82         A(3) => '0',
83         S => sel,
84         Y => out_mux4(N-2)
85     );
86
87 Mux2_Secondo: mux_2_1
88     port map(
89         A(0) => out_mux4(N-2),
90         A(1) => input(N-2),
91         s => load,
```

```
92          y => out_mux2(N-2)
93      );
94
95  FF_Secondo: FF_D
96
97      port map(
98
99          clk => clk,
100
101         reset => reset,
102
103         d => out_mux2(N-2),
104
105         y => out_FFD(N-2)
106
107     );
108
109
110 Mux4_intermedi: for i in N-3 downto 2 generate
111
112     mux: mux_4_1
113
114         port map(
115
116             A(0) => out_FFD(i-1),
117
118             A(1) => out_FFD(i+1),
119
120             A(2) => out_FFD(i-2),
121
122             A(3) => out_FFD(i+2),
123
124             S => sel,
125
126             Y => out_mux4(i)
127
128         );
129
130     end generate Mux4_intermedi;
131
132
133 Mux2_intermedi: for i in N-3 downto 2 generate
134
135     mux: mux_2_1
136
137         port map(
138
139             A(0) => out_mux4(i),
140
141             A(1) => input(i),
```

```
120           s => load,
121           y => out_mux2(i)
122       );
123   end generate Mux2_intermedi;
124
125 FF_intermedi: for i in N-3 downto 2 generate
126     FF: FF_D
127       port map(
128         clk => clk,
129         reset => reset,
130         d => out_mux2(i),
131         y => out_FFD(i)
132       );
133   end generate FF_intermedi;
134
135 Mux4_penultimo: mux_4_1
136   port map(
137     A(0) => out_FFD(0),
138     A(1) => out_FFD(2),
139     A(2) => '0',
140     A(3) => out_FFD(3),
141     S => sel,
142     Y => out_mux4(1)
143   );
144
145 Mux2_penultimo: mux_2_1
146   port map(
147     A(0) => out_mux4(1),
```

```
148      A(1) => input(1),
149      s => load,
150      y => out_mux2(1)
151  );
152
153 FF_penultimo: FF_D
154     port map(
155         clk => clk,
156         reset => reset,
157         d => out_mux2(1),
158         y => out_FFD(1)
159     );
160
161 Mux4_ultimo: mux_4_1
162     port map(
163         A(0) => '0',
164         A(1) => out_FFD(1),
165         A(2) => '0',
166         A(3) => out_FFD(2),
167         S => sel,
168         Y => out_mux4(0)
169     );
170
171 Mux2_ultimo: mux_2_1
172     port map(
173         A(0) => out_mux4(0),
174         A(1) => input(0),
175         s => load,
```

```
176      y  => out_mux2(0)
177  );
178
179  FF_ultimo: FF_D
180    port map(
181      clk => clk,
182      reset => reset,
183      d  => out_mux2(0),
184      y  => out_FFD(0)
185  );
186
187  output <= out_FFD;
188
189 end Structural;
```

## 4.5 Simulazione: approccio behavioral

La simulazione dello shift register è stata fatta testando tutte e 4 le operazioni definite. Dopo un reset iniziale è stato fatto il load dei dati iniziali, ovvero "10101010". Poi è sono stati effettuati lo shift left di 1 (sel = "00"), lo shift right di 1 (sel = "01"), lo shift left di 2 (sel = "10") e infine lo shift right di 2 (sel = "11").

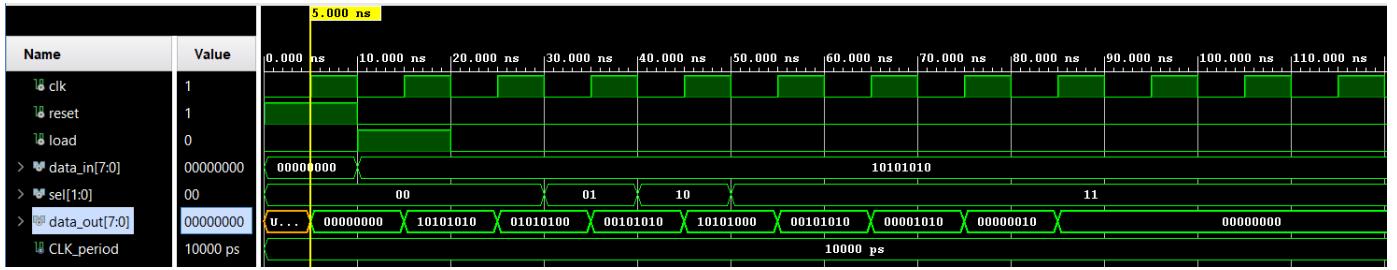


Figure 4.3: Simulazione Shift Register Comportamentale

## 4.6 Simulazione: approccio strutturale

Anche in questo caso sono state testate tutte e 4 le operazioni di shift definite. Però in questo abbiamo testato uno shift register da 4 bit. La sequenza di operazioni è la stessa di quella nel caso comportamentale.

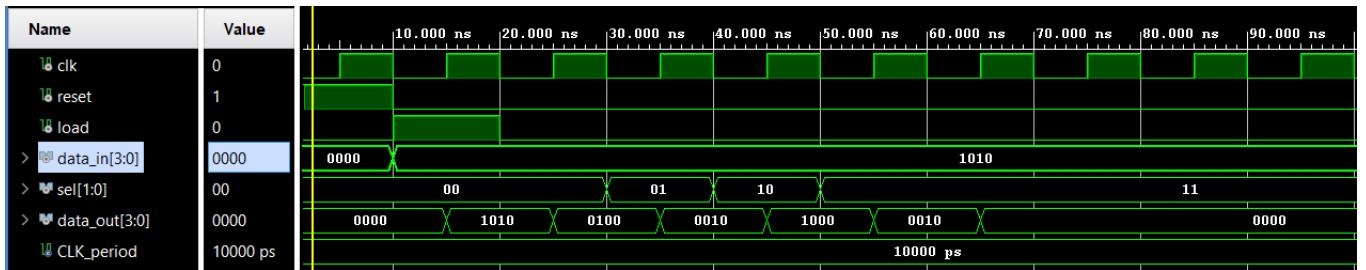


Figure 4.4: Simulazione Shift Register Strutturale

# Chapter 5

## Cronometro

### 5.1 Traccia 4.1

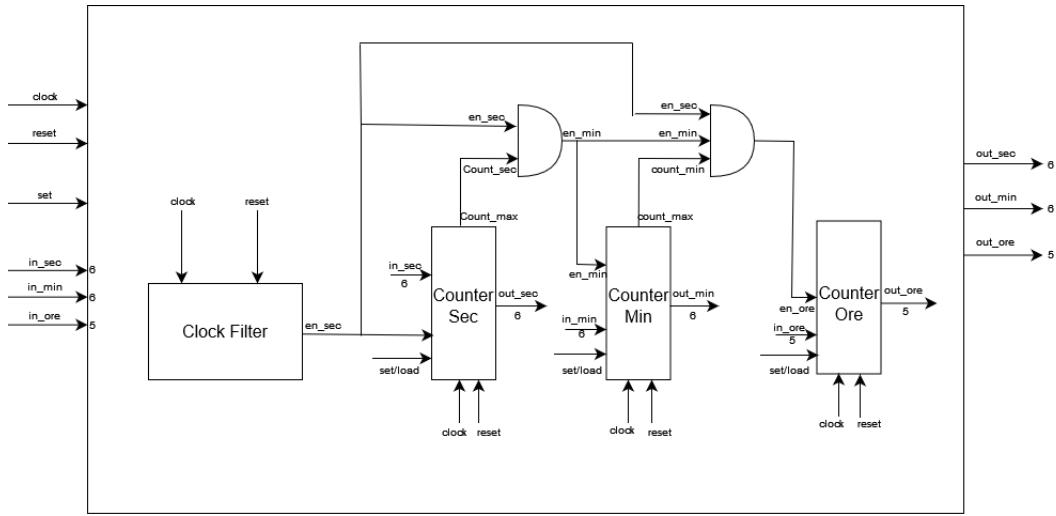
- Progettare, implementare in VHDL e testare mediante simulazione un cronometro, in grado di scandire secondi, minuti e ore a partire da una base dei tempi prefissata (es. si consideri il clock a disposizione sulla board). Il progetto deve prevedere la possibilità di inizializzare il cronometro con un valore iniziale, sempre espresso in termini di ore, minuti e secondi, mediante un opportuno ingresso di set, e deve prevedere un ingresso di reset per azzerare il tempo. Il componente deve essere realizzato utilizzando un approccio strutturale, collegando opportunamente dei contatori secondo uno schema a scelta.

### 5.1.1 Progetto e architettura

Il componente essenziale del cronometro è il contatore modulo N. Questo contatore è un componente fondamentale del sistema, con un ruolo chiave nel controllo del flusso di dati.

Il cronometro è un dispositivo composto da quattro componenti distinti: un clock filter e 3 contatori. Abbiamo sviluppato un componente fortemente sincrono, quindi tutti e quattro i componenti rispondono allo stesso segnale di clock, fornito dalla scheda, ma incrementano il conteggio solo quando tutti i contatori precedenti hanno raggiunto il loro valore massimo e quindi hanno un segnale di uscita alto.

Il primo componente funziona principalmente come base temporale, il suo compito principale è quindi quello di agire come un divisore di frequenza, restituendo, a partire dal segnale di clock della scheda a 100MHz, un segnale di conteggio a 1Hz. I contatori successivi sono due contatori modulo sessanta, che contano rispettivamente i secondi e i minuti, e un contatore modulo ventiquattro, per il conteggio delle ore. Un diagramma dettagliato dell’interconnessione dei contatori è illustrato nella figura 5.1.1.



### 5.1.2 Implementazione

#### Contatore

Il contatore è definito come un’entità con diverse porte di ingresso e uscita, tra cui il clock (clk), il reset (reset), l’abilitazione (enable), il caricamento (load), l’input (input), l’output (y) e il segnale di conteggio massimo (countMax).

L’architettura comportamentale del contatore è definita attraverso due processi principali: logica\\_conteggio e proc\\_count\\_max. Il primo processo gestisce la logica di conteggio, incrementando una variabile temporanea (temp\_y) ad ogni fronte discendente del clock, a condizione che il segnale di abilitazione sia alto. Se la variabile raggiunge il valore massimo (modulo-1), viene azzerata, permettendo l’inizio di un nuovo ciclo di conteggio. Inoltre, è possibile impostare il valore della variabile in modo arbitrario attraverso il segnale di caricamento.

Il secondo processo, proc\_count\_max, gestisce il segnale di con-

teggio massimo. Questo segnale diventa ‘1’ ogni volta che la variabile temporanea raggiunge il valore massimo, indicando la fine di un ciclo di conteggio.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity counter is
6     generic(
7         modulo: integer := 8;
8         n: integer := 3
9     );
10    port (
11        clk: in std_logic;
12        reset: in std_logic;
13        enable: in std_logic;
14        load: in std_logic;
15        input: in std_logic_vector(n-1 downto 0);
16        y: out std_logic_vector(n-1 downto 0);
17        countMax: out std_logic
18    );
19 end counter;
20
21 architecture Behavioral of counter is
22
23 signal temp_y: std_logic_vector(n-1 downto 0) :=
24     (others => '0');
25 signal temp_count: std_logic := '0';
```

```
25
26 begin
27
28     y <= temp_y;
29
30     logica_conteggio: process(clk)
31 begin
32     if falling_edge(clk) then
33         if (reset = '1') then
34             temp_y <= (others => '0');
35         elsif (load = '1') then
36             temp_y <= input;
37         elsif (enable = '1') then
38             --vediamo il conteggio+FFFA privato al
39             --massimo valore (modulo-1)
40             if (temp_y =
41                 std_logic_vector(TO_UNSIGNED(modulo-1, n)))
42                 then
43                     temp_y <= (others => '0');
44                 else
45                     temp_y <=
46                         std_logic_vector(unsigned(temp_y)
47                             + 1);
48                 end if;
49             end if;
50         end if;
51     end process;
```

```

48      proc_count_max: process(clk)
49
50        begin
51
52          if rising_edge(clk) then
53
54            if (reset = '1') then
55
56              temp_count <= '0';
57
58            elsif (temp_y =
59                std_logic_vector(TO_UNSIGNED(modulo-1,n)))
60
61              then
62
63              temp_count <= '1';
64
65            else
66
67              temp_count <= '0';
68
69            end if;
70
71          end if;
72
73        end process;
74
75
76        countMax <= temp_count;
77
78
79      end Behavioral;

```

## Clock\_filter

```

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 := 1
9

```

```

--frequenza desiderata 1Hz

8      );
9
10     Port (
11         clock_in : in STD_LOGIC;
12         reset : in STD_LOGIC;
13         clock_out : out STD_LOGIC -- attenzione:
14             nphi+FFFFDj vero clock ma un impulso che
15             si U+FFFDisato come enable
16     );
17
18 end clock_filter;

19
20 architecture Behavioral of clock_filter is
21
22 signal clockfx : std_logic := '0';
23
24 constant count_max_value : integer :=
25     CLKIN_freq/ (CLKOUT_freq)-1;
26
27 begin
28
29     clock_out <= clockfx;
30
31     count_for_division: process(clock_in)
32
33         variable counter : integer range 0 to count_max_value
34             := 0;
35
36         begin
37
38             if (clock_in = '1') then
39                 counter := counter + 1;
40             end if;
41
42             if (counter = count_max_value) then
43                 clockfx := not clockfx;
44                 counter := 0;
45             end if;
46
47         end process;
48
49     end;

```

```

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;

```

**Cronometro** Questo cronometro è un componente fondamentale del sistema, con un ruolo chiave nel controllo del flusso di dati. Il cronometro è definito come un'entità con diverse porte di ingresso e uscita, tra cui il clock (clk), il reset (rst), il set (set), l'input per i secondi (in\_sec), i minuti (in\_min) e le ore (in\_ore), e l'output per i secondi (out\_sec), i minuti (out\_min) e le ore (out\_ore). L'architettura strutturale del cronometro è definita attraverso l'instanziazione di due componenti principali: clock\_filter e counter. Il primo componente genera un seg-

nale en\_sec che abilita il contatore dei secondi. Questo contatore incrementa il conteggio dei secondi ad ogni fronte discendente del clock, a condizione che il segnale di abilitazione sia alto. Se il conteggio raggiunge il valore massimo (60), viene azzerato, dando avvio a un nuovo ciclo di conteggio. Inoltre, è possibile impostare il valore del conteggio in modo arbitrario attraverso il segnale di set.

Quando il conteggio dei secondi raggiunge il massimo, viene generato un segnale en\_min che abilita il contatore dei minuti. Questo contatore funziona in modo simile al contatore dei secondi, ma conta i minuti invece dei secondi. Infine, quando il conteggio dei minuti raggiunge il massimo, viene generato un segnale en\_ore che abilita il contatore delle ore.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity cronometro is
5   port (
6     clk: in std_logic;          -- Segnale
7                 di clock in ingresso.
8     rst: in std_logic;          -- Segnale
9                 di reset in ingresso.
10    set: in std_logic;          -- Segnale
11                 per impostare i valori iniziali.
12    in_sec: in std_logic_vector(5 downto 0);  --
13                 Valore iniziale per i secondi.
14    in_min: in std_logic_vector(5 downto 0);  --
15
```

```

    Valore iniziale per i minuti.

11   in_ore: in std_logic_vector(4 downto 0); --  

    Valore iniziale per le ore.  

12   out_sec: out std_logic_vector(5 downto 0); --  

    Output dei secondi.  

13   out_min: out std_logic_vector(5 downto 0); --  

    Output dei minuti.  

14   out_ore: out std_logic_vector(4 downto 0) --  

    Output delle ore.  

15 );  

16 end cronometro;  

17  

18 architecture Structural of cronometro is  

19  

20   -- Segnali interni per abilitare i contatori e per  

21   -- segnalare il conteggio massimo.  

22   signal en_sec: std_logic := '0';  

23   signal en_min: std_logic := '0';  

24   signal en_ore: std_logic := '0';  

25   signal count_sec: std_logic := '0';  

26   signal count_min: std_logic := '0';  

27   signal count_ore: std_logic := '0';  

28  

29 component clock_filter  

30   generic(  

31     CLKIN_freq : integer := 100000000; --  

      Frequenza di clock della scheda (100MHz).  

      CLKOUT_freq : integer := 1           --

```

```
        Frequenza desiderata di output (1Hz) .  
32            );  
33    Port  (  
34        clock_in : in STD_LOGIC;           -- Clock  
35                    in ingresso.  
36        reset : in STD_LOGIC;             -- Segnale  
37                    di reset.  
38        clock_out : out STD_LOGIC          -- Clock  
39                    ridotto in uscita.  
40    );  
41 end component;  
42  
43  
44 component counter  
45 generic(  
46     modulo: integer := 8;  
47     n: integer := 3  
48 );  
49 port(  
50     clk: in std_logic;  
51     reset: in std_logic;  
52     enable: in std_logic;  
53     load: in std_logic;  
54     input: in std_logic_vector(n-1 downto 0);  
55     y: out std_logic_vector(n-1 downto 0);  
56     countMax: out std_logic  
57 );  
58 end component;
```

```
56 begin
57
58     -- Instanziazione del componente clock_filter per
59     -- generare il segnale "en_sec".
60
61     base_dei_tempi: clock_filter
62         generic map( 100000000, 100000000)
63         port map(
64             clock_in => clk,
65             reset => rst,
66             clock_out => en_sec
67         );
68
69
70     counter_secondi: counter
71         generic map(
72             60,                                     -- Conta
73             fino a 60 per i secondi.
74             6                                         -- --
75             Larghezza del contatore di 6 bit.
76         )
77
78         port map(
79             clk => clk,
80             reset => rst,
81             enable => en_sec,
82             load => set,
83             input => in_sec,
84             y => out_sec,
85             countMax => count_sec
86         );
87
```

```
81
82    -- Abilitazione del contatore dei minuti quando i
83        secondi raggiungono il massimo.
84
85    en_min <= en_sec and count_sec;
86
87    counter_minuti: counter
88        generic map(
89            60,                                     -- Conta
90                fino a 60 per i minuti.
91            6                                     --
92                Larghezza del contatore di 6 bit.
93
94        )
95
96        port map(
97            clk => clk,
98            reset => rst,
99            enable => en_min,
100           load => set,
101           input => in_min,
102           y => out_min,
103           countMax => count_min
104       );
105
106
107    -- Abilitazione del contatore delle ore quando i
108        minuti raggiungono il massimo.
109
110    en_ore <= en_sec and en_min and count_min;
111
112
113    counter_ore: counter
114        generic map(
```

```

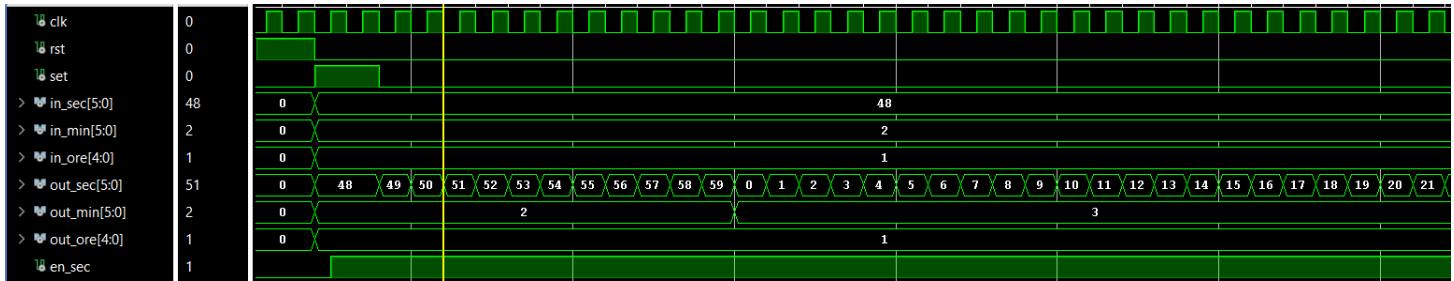
105      24,          -- Conta
106      fino a 24 per le ore.
107      5           --
108      Larghezza del contatore di 5 bit.
109      )
110
111      port map(
112          clk => clk,
113          reset => rst,
114          enable => en_ore,
115          load => set,
116          input => in_ore,
117          y => out_ore,
118          countMax => count_ore
119      );
120
121
122      end Structural;

```

### 5.1.3 Simulazione

Nel corso della simulazione, abbiamo impostato il cronometro a un tempo iniziale di 01:02:48. Questo ha permesso di osservare l’evoluzione del cronometro nel tempo. Come previsto, abbiamo notato che il conteggio si incrementa ad ogni fronte discendente del clock, rispecchiando fedelmente il comportamento di un cronometro reale. Questo fenomeno è stato osservato per tutte le unità di tempo: secondi, minuti e ore. In particolare, abbiamo potuto constatare che, ad ogni fronte

descendente del clock, il contatore dei secondi avanzava di un'unità. Quando questo raggiungeva il massimo di 60, veniva azzerato e il contatore dei minuti veniva incrementato di uno.



## 5.2 Traccia 5.2

- 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).

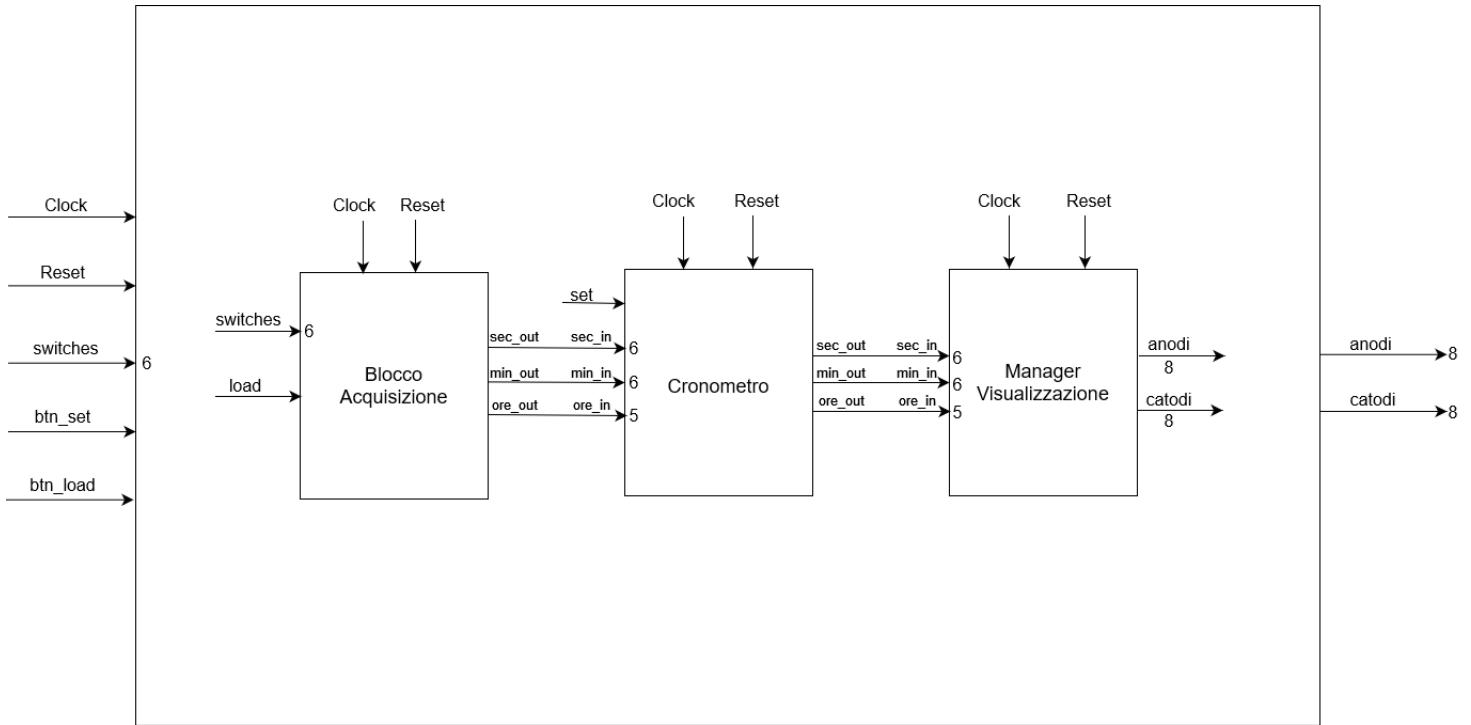
### 5.2.1 Progetto e architettura

Per l'implementazione su scheda, è necessaria un'architettura più sofisticata che si compone di tre componenti principali:

- **Blocco Acquisizione:** Questo componente è responsabile della

gestione degli input per l'impostazione dell'orario iniziale. Il suo ruolo è di fornire al cronometro i valori da impostare nei registri, facilitando così l'input dell'orario desiderato.

- **Cronometro:** Il cronometro rimane invariato rispetto alla sua implementazione precedente. Gli input per l'impostazione dell'orario sono forniti dal Blocco Acquisizione, mentre la gestione dell'output è delegata al Manager Visualizzazione.
- **Manager Visualizzazione:** Questo componente ha il compito di visualizzare l'output sul display a 7 segmenti della scheda. Per farlo, prende continuamente in input il valore corrente del cronometro, converte i valori in formato decimale e gestisce i valori di output che gli anodi e i catodi devono assumere per visualizzare correttamente l'orario sul display.


 Figure 5.1: Architettura complessiva (Cronometro<sub>onBoard</sub>)

**Blocco Acquisizione** Il blocco di acquisizione è definito come un'entità con diverse porte di ingresso e uscita, tra cui il clock (clk), il reset (reset), gli switch (switches), il caricamento (load), e le uscite per i secondi (sec\_out), i minuti (min\_out) e le ore (ore\_out). L'architettura strutturale del blocco di acquisizione è definita attraverso l'istanziazione di 5 componenti principali: un Button Debouncer, l'unità di controllo e tre registri di acquisizione. Il primo componente gestisce il debouncing del pulsante di caricamento, eliminando eventuali oscillazioni indesiderate. La control Unit gestisce il caricamento dei dati, fornendo segnali di caricamento separati per i secondi, i minuti e le ore. Infine, troviamo i registri di acquisizione, che memorizzano i valori degli switch in ingresso quando il segnale di caricamento cor-

rispondente è alto.

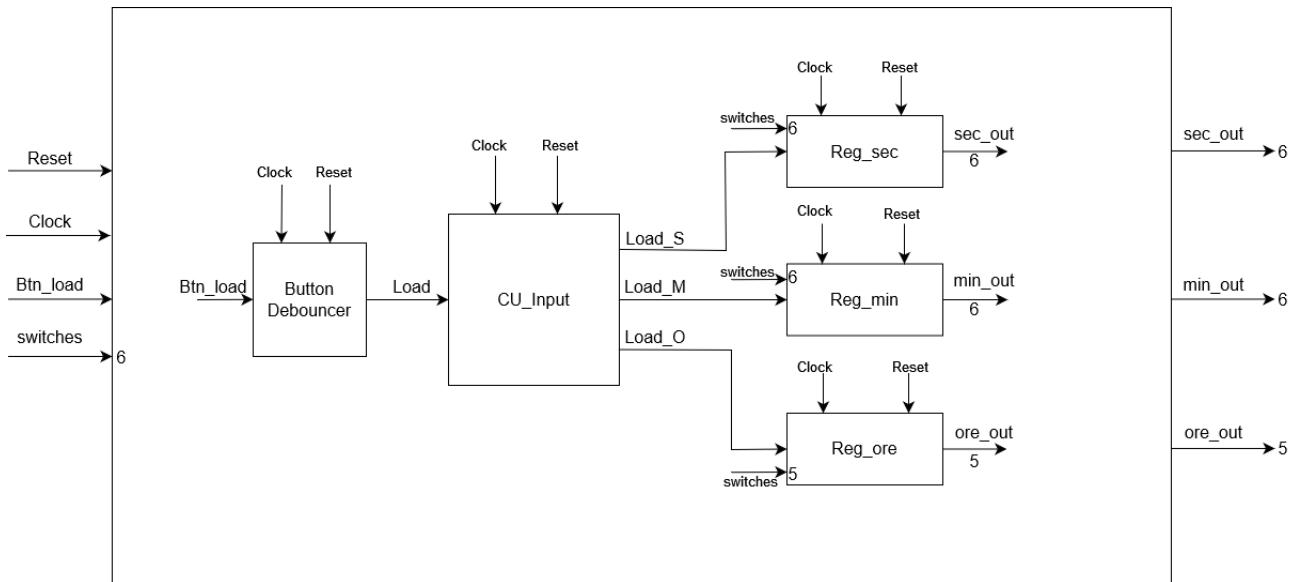


Figure 5.2: Blocco-Acquisizione

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Blocco_acquisizione is
5     port(
6         clk: in std_logic;
7         reset: in std_logic;
8         switches: in std_logic_vector(5 downto 0);
9         load: in std_logic;
10        sec_out: out std_logic_vector(5 downto 0);
11        min_out: out std_logic_vector(5 downto 0);
12        ore_out: out std_logic_vector(4 downto 0)
13    );
14 end Blocco_acquisizione;
15
16 architecture Structural of Blocco_acquisizione is

```

```
17
18 component CU_input
19
20     port (
21         clk: in std_logic;
22         reset: in std_logic;
23         load: in std_logic;
24         load_sec: out std_logic;
25         load_min: out std_logic;
26         load_ore: out std_logic
27     );
28
29 end component;
30
31
32 component ButtonDebouncer
33
34     generic (
35         CLK_period: integer := 10;    -- periodo del
36
37         clock in nanosec
38
39         btn_noise_time: integer := 10000000
40
41             --durata dell'oscillazione in nanosec
42     );
43
44     Port (
45
46         RST : in STD_LOGIC;
47
48         CLK : in STD_LOGIC;
49
50         BTN : in STD_LOGIC;
51
52         CLEARED_BTN : out STD_LOGIC
53
54     );
55
56 end component;
57
58
59 component registro_acq is
```

```
43      generic (
44          N: integer := 8 --dimensione del registro
45      );
46
47      port (
48          clk: in std_logic;
49          reset: in std_logic;
50          load: in std_logic; --segnale di
51                      --caricamento/scrittura
52          data_in: in std_logic_vector(N-1 downto 0);
53          data_out: out std_logic_vector(N-1 downto 0)
54      );
55
56      end component;
57
58      signal load_temp: std_logic := '0';
59
60 begin
61
62     deb: ButtonDebouncer
63     port map(
64         RST => reset,
65         CLK => clk,
66         BTN => load,
67         CLEARED_BTN => load_temp
68     );
69
```

```
70      cu: CU_input
71      port map(
72          clk => clk,
73          reset => reset,
74          load => load_temp,
75          load_sec => load_sec_temp,
76          load_min => load_min_temp,
77          load_ore => load_ore_temp
78      );
79
80      reg_sec: registro_acq
81      generic map(6)
82      port map(
83          clk => clk,
84          reset => reset,
85          load => load_sec_temp,
86          data_in => switches,
87          data_out => sec_out
88      );
89
90      reg_min: registro_acq
91      generic map(6)
92      port map(
93          clk => clk,
94          reset => reset,
95          load => load_min_temp,
96          data_in => switches,
97          data_out => min_out
```

```

98      );
99
100     reg_ore: registro_acq
101     generic map(5)
102     port map(
103         clk => clk,
104         reset => reset,
105         load => load_ore_temp,
106         data_in => switches(4 downto 0),
107         data_out => ore_out
108     );
109
110 end Structural;

```

Per quanto riguarda l'unità di controllo, L'FSM è definito attraverso vari stati, tra cui idle, carica\_secondi, ok\_secondi, carica\_minuti, ok\_minuti, carica\_ore, e ok\_ore. L'FSM inizia nello stato idle e transita in altri stati in base ai segnali di input load e reset. Un segnale di clock orchestra le transizioni tra gli stati. Quando il segnale load è attivato, l'FSM si muove sequenzialmente attraverso gli stati per caricare i secondi, i minuti e le ore. Se il segnale reset viene attivato in qualsiasi momento, l'FSM ritorna al suo stato idle. Ogni stato di caricamento (carica\_secondi, carica\_minuti, carica\_ore) ha un corrispondente stato di conferma (ok\_secondi, ok\_minuti, ok\_ore) che indica il completamento del caricamento. Questo è stato fatto poichè gli switch non erano sufficienti ad acquisire tutti i vaori necessari per

settare l'orario.

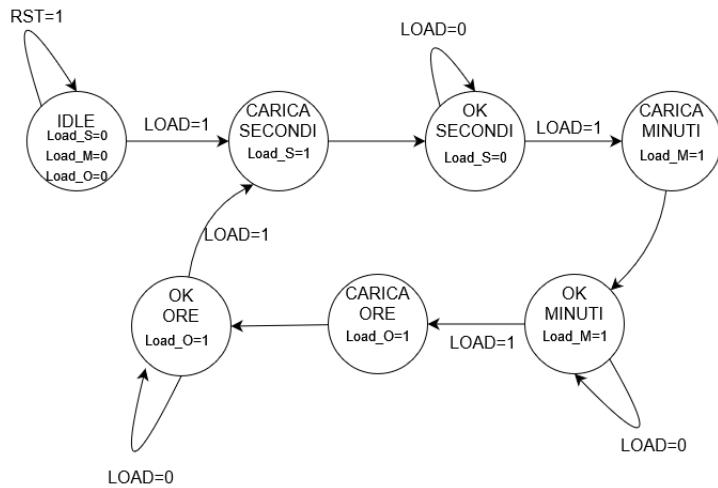


Figure 5.3: FSM CU\_input

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CU_input is
5   port (
6     clk: in std_logic;
7     reset: in std_logic;
8     load: in std_logic;
9     load_sec: out std_logic;
10    load_min: out std_logic;
11    load_ore: out std_logic
12  );
13 end CU_input;
14
15 architecture Behavioral of CU_input is
16
17 type stato is (idle, carica_secondi, ok_secondi,
  
```

```
    carica_minuti, ok_minuti, carica_ore, ok_ore);  
18 signal stato_corrente: stato := idle;  
19  
20 begin  
21  
22 proc: process(clk)  
23 begin  
24 if(rising_edge(clk)) then  
25 if(reset = '1') then  
26     stato_corrente <= idle;  
27     load_sec <= '0';  
28     load_min <= '0';  
29     load_ore <= '0';  
30 else  
31 case stato_corrente is  
32 when idle =>  
33     load_sec <= '0';  
34     load_min <= '0';  
35     load_ore <= '0';  
36     if(load = '1') then  
37         stato_corrente <=  
38             carica_secondi;  
39     else  
40         stato_corrente <= idle;  
41     end if;  
42 when carica_secondi =>  
43     load_sec <= '1';  
     stato_corrente <= ok_secondi;
```

```
44      when ok_secondi =>
45          load_sec <= '0';
46          if(load = '1') then
47              stato_corrente <=
48                  carica_minuti;
49          else
50              stato_corrente <=
51                  ok_secondi;
52          end if;
53
54      when carica_minuti =>
55          load_min <= '1';
56          stato_corrente <= ok_minuti;
57
58      when ok_minuti =>
59          load_min <= '0';
60          if(load = '1') then
61              stato_corrente <=
62                  carica_ore;
63          else
64              stato_corrente <= ok_minuti;
65          end if;
66
67      when carica_ore =>
68          load_ore <= '1';
69          stato_corrente <= ok_ore;
70
71      when ok_ore =>
72          load_ore <= '0';
73          if(load = '1') then
74              stato_corrente <=
75                  carica_secondi;
```

```
68         else
69             stato_corrente <= ok_ore;
70         end if;
71     end case;
72 end if;
73 end if;
74 end process;
75
76 end Behavioral;
```

**Manager Visualizzazione** Questo componente svolge un ruolo cruciale per garantire una visualizzazione accurata dei valori del cronometro sul display a 7 segmenti in formato decimale. Riguardo al display a 7 segmenti, si attivano esclusivamente le prime sei cifre e il terzo e quinto puntino. Questa configurazione è adeguata per rappresentare l'orario nel formato standard 00:00:00. In questo modo, l'orario viene visualizzato in maniera chiara e intuitiva, facilitando la lettura da parte dell'utente.

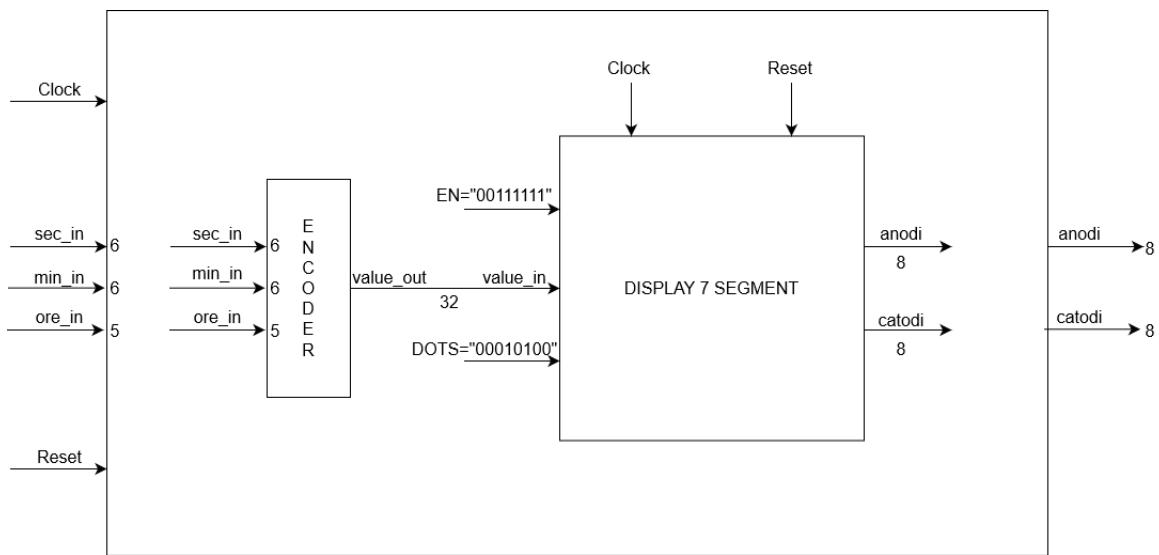


Figure 5.4: Manager\_Visualizzazione

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity manager_visualizzazione is
5 port (
6     clk: in std_logic;
7     reset: in std_logic;
8     sec_in: in std_logic_vector(5 downto 0);
9     min_in: in std_logic_vector(5 downto 0);
10    ore_in: in std_logic_vector(4 downto 0);
11    anodi: out std_logic_vector(7 downto 0);
12    catodi: out std_logic_vector(7 downto 0)
13 );
14 end manager_visualizzazione;
15
16 architecture Structural of manager_visualizzazione is
17 
```

```
18 component encoder is
19   port(
20     sec_in: in std_logic_vector(5 downto 0);
21     min_in: in std_logic_vector(5 downto 0);
22     ore_in: in std_logic_vector(4 downto 0);
23     value_out: out std_logic_vector(31 downto 0)
24   );
25 end component;
26
27 component display_seven_segments is
28   Generic(
29     CLKIN_freq : integer := 100000000;
30     CLKOUT_freq : integer := 500
31   );
32   Port ( CLK : in STD_LOGIC;
33         RST : in STD_LOGIC;
34         VALUE : in STD_LOGIC_VECTOR (31 downto
35           0);
36         ENABLE : in STD_LOGIC_VECTOR (7 downto
37           0); -- decide quali cifre abilitare
38         DOTS : in STD_LOGIC_VECTOR (7 downto
39           0); -- decide quali punti visualizzare
40         ANODES : out STD_LOGIC_VECTOR (7 downto
41           0);
42         CATHODES : out STD_LOGIC_VECTOR (7
43           downto 0));
44   end component;
```

```

41      signal value_out_temp: std_logic_vector(31 downto
42          0) := (others => '0');
43
44
45  enc: encoder
46
47      port map(
48          sec_in => sec_in,
49          min_in => min_in,
50          ore_in => ore_in,
51          value_out => value_out_temp
52      );
53
54  dis: display_seven_segments
55
56      generic map(
57          CLKIN_freq => 100000000,
58          CLKOUT_freq => 500
59      )
60
61      port map(
62          CLK => clk,
63          RST => reset,
64          VALUE => value_out_temp,
65          ENABLE => "00111111", --accendiamo tutte le
66          cifre tranne le prime due
67          DOTS => "00010100", --accendiamo solo i
68          punti tra ore e min e tra min e sec
69          ANODES => anodi,
70          CATHODES => catodi

```

```

66      );
67
68 end Structural;

```

L'**encoder** è definito come un'entità con diverse porte di ingresso e uscita, tra cui i secondi (sec\_in), i minuti (min\_in), le ore (ore\_in), e l'output (value\_out). L'architettura dell'encoder è definita attraverso vari segnali temporanei (sec\_temp, min\_temp, ore\_temp) e una serie di istruzioni di selezione. Il suo scopo è quello di trasformare il valore di secondi, minuti e ore (che è in binario) dapprima in decimale e successivamente trasformare le due cifre (unità e decine) separatamente in binario. Considerando che per ogni cifra ci possono essere al massimo 4 bit, per i valori di secondi, minuti e ore avremo bisogno di al massimo 8 bit per rappresentare le due cifre componenti. Ciò è stato fatto in quanto le cifre così trasformate saranno rappresentate sul display a 7 segmenti in decimale. In particolare, l'encoder prende in ingresso i secondi, i minuti e le ore, li converte in formato binario e li concatena in un unico output di 32 bit.

```

1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.numeric_std.ALL;
5
6 entity encoder is
7   Port (
8     sec_in: in std_logic_vector(5 downto 0);

```

```

9      min_in: in std_logic_vector(5 downto 0);
10     ore_in: in std_logic_vector(4 downto 0);
11     value_out: out std_logic_vector(31 downto 0)
12   );
13 end encoder;
14
15 architecture Dataflow of Encoder is
16
17   signal sec_temp : std_logic_vector (7 downto 0) :=
18     (others => '0');
19   signal min_temp : std_logic_vector (7 downto 0) :=
20     (others => '0');
21   signal ore_temp : std_logic_vector (7 downto 0) :=
22     (others => '0');
23
24 begin
25
26   value_out <= "00000000" & ore_temp & min_temp &
27     sec_temp;
28
29   -- Prima cifra dei secondi
30   with to_integer(unsigned(sec_in)) select
31     sec_temp (3 downto 0) <= "0000" when
32       0|10|20|30|40|50,
33                               "0001" when
34       1|11|21|31|41|51,
35                               "0010" when
36       2|12|22|32|42|52,
37                               "0011" when
38       3|13|23|33|43|53,
39                               "0100" when
40       4|14|24|34|44|54,
41                               "0101" when
42       5|15|25|35|45|55,
43                               "0110" when
44       6|16|26|36|46|56,
45                               "0111" when
46       7|17|27|37|47|57,
47                               "1000" when
48       8|18|28|38|48|58,
49                               "1001" when
50       9|19|29|39|49|59,
51                               "1010" when
52       10|20|30|40|50,
53                               "1011" when
54       11|21|31|41|51,
55                               "1100" when
56       12|22|32|42|52,
57                               "1101" when
58       13|23|33|43|53,
59                               "1110" when
60       14|24|34|44|54,
61                               "1111" when
62       15|25|35|45|55;
63
64   end;

```

```

30          "0011" when
31          3|13|23|33|43|53,
32          "0100" when
33          4|14|24|34|44|54,
34          "0101" when
35          5|15|25|35|45|55,
36          "0110" when
37          6|16|26|36|46|56,
38          "0111" when
39          7|17|27|37|47|57,
40          "1000" when
41          8|18|28|38|48|58,
42          "1001" when
43          9|19|29|39|49|59,
44          "1111" when others;

45
46
-- Seconda cifra dei secondi
with to_integer(unsigned(sec_in)) select
    sec_temp (7 downto 4) <= "0000" when 0 to 9,
                    "0001" when 10 to
                    19,
                    "0010" when 20 to
                    29,
                    "0011" when 30 to
                    39,
                    "0100" when 40 to
                    49,
                    "0101" when 50 to

```

```

59,
47           "1111" when others;

48
49 -- Prima cifra dei minuti
50 with to_integer(unsigned(min_in)) select
51     min_temp (3 downto 0) <= "0000" when
52         0|10|20|30|40|50,
53             "0001" when
54                 1|11|21|31|41|51,
55             "0010" when
56                 2|12|22|32|42|52,
57             "0011" when
58                 3|13|23|33|43|53,
59             "0100" when
60                 4|14|24|34|44|54,
61             "0101" when
62                 5|15|25|35|45|55,
63             "0110" when
64                 6|16|26|36|46|56,
65             "0111" when
66                 7|17|27|37|47|57,
67             "1000" when
68                 8|18|28|38|48|58,
69             "1001" when
70                 9|19|29|39|49|59,
71             "1111" when others;

72
73 -- Seconda cifra dei minuti

```

```
64      with to_integer(unsigned(min_in)) select
65          min_temp (7 downto 4) <= "0000" when 0 to 9,
66                                  "0001" when 10 to
67                                      19,
68                                  "0010" when 20 to
69                                      29,
70                                  "0011" when 30 to
71                                      39,
72                                  "0100" when 40 to
73                                      49,
74                                  "0101" when 50 to
75                                      59,
76                                  "1111" when others;
77
78      -- Prima cifra delle ore
79      with to_integer(unsigned(ore_in)) select
80          ore_temp (3 downto 0) <= "0000" when
81              0|10|20,
82                                  "0001" when 1|11|21,
83                                  "0010" when 2|12|22,
84                                  "0011" when 3|13|23,
85                                  "0100" when 4|14,
```

```

86
87    -- Seconda cifra delle ore
88
89    with to_integer(unsigned(ore_in)) select
90
91        ore_temp (7 downto 4) <= "0000" when 0 to
92
93            9,
94
95            "0001" when 10 to
96
97            19,
98
99            "0010" when 20 to
100
101            23,
102
103            "1111" when others;
104
105
106 end Dataflow;

```

### 5.2.2 Sintesi su board

Nel processo di sintesi su scheda, abbiamo definito nei vincoli fisici che i primi sei switch fungono da input per impostare i secondi/minuti/ora. Abbiamo assegnato il pin N17 come pulsante di reset, il P17 come pulsante di impostazione del cronometro e M17 come pulsante di caricamento dei valori dagli switch. In assenza di interazioni con i pulsanti di impostazione o reset, il cronometro prosegue ininterrottamente nel conteggio. Abbiamo inoltre verificato la funzionalità di reset automatico, che si attiva quando il cronometro raggiunge il valore di 23:59:59, ripristinando l'orario a 00:00:00.

```

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

```

```
3
4 entity Cronometro_onBoard is
5   port (
6     clk: in std_logic;
7     reset: in std_logic;
8     switches: in std_logic_vector(5 downto 0);
9     btn_load: in std_logic;
10    btn_set: in std_logic;
11    catodi: out std_logic_vector(7 downto 0);
12    anodi: out std_logic_vector(7 downto 0)
13  );
14 end Cronometro_onBoard;
15
16 architecture Structural of Cronometro_onBoard is
17
18   component Blocco_acquisizione
19     port (
20       clk: in std_logic;
21       reset: in std_logic;
22       switches: in std_logic_vector(5 downto 0);
23       load: in std_logic;
24       sec_out: out std_logic_vector(5 downto 0);
25       min_out: out std_logic_vector(5 downto 0);
26       ore_out: out std_logic_vector(4 downto 0)
27     );
28   end component;
29
30   component cronometro
```

```

31      port (
32          clk: in std_logic; -- Segnale di clock in ingresso.
33          rst: in std_logic; -- Segnale di reset in ingresso.
34          set: in std_logic; -- Segnale per impostare i valori iniziali.
35          in_sec: in std_logic_vector(5 downto 0); -- Valore iniziale per i secondi.
36          in_min: in std_logic_vector(5 downto 0); -- Valore iniziale per i minuti.
37          in_ore: in std_logic_vector(4 downto 0); -- Valore iniziale per le ore.
38          out_sec: out std_logic_vector(5 downto 0); -- Output dei secondi.
39          out_min: out std_logic_vector(5 downto 0); -- Output dei minuti.
40          out_ore: out std_logic_vector(4 downto 0) -- Output delle ore.
41      );
42  end component;
43
44 component manager_visualizzazione
45  port (
46      clk: in std_logic;
47      reset: in std_logic;
48      sec_in: in std_logic_vector(5 downto 0);
49      min_in: in std_logic_vector(5 downto 0);

```

```

50      ore_in: in std_logic_vector(4 downto 0);
51      anodi: out std_logic_vector(7 downto 0);
52      catodi: out std_logic_vector(7 downto 0)
53  );
54 end component;
55
56 signal sec_in_temp: std_logic_vector(5 downto 0) :=
57   (others => '0');
58 signal min_in_temp: std_logic_vector(5 downto 0) :=
59   (others => '0');
60 signal ore_in_temp: std_logic_vector(4 downto 0) :=
61   (others => '0');
62 signal sec_out_temp: std_logic_vector(5 downto 0)
63   := (others => '0');
64 signal min_out_temp: std_logic_vector(5 downto 0)
65   := (others => '0');
66 signal ore_out_temp: std_logic_vector(4 downto 0)
67   := (others => '0');
68
69 begin
70
71   ba: Blocco_acquisizione port map(
72     clk => clk,
73     reset => reset,
74     switches => switches,
75     load => btn_load,
76     sec_out => sec_in_temp,
77     min_out => min_in_temp,

```

```
72         ore_out => ore_in_temp  
73     );  
74  
75 cron: cronometro port map(  
76     clk => clk,  
77     rst => reset,  
78     set => btn_set,  
79     in_sec => sec_in_temp,  
80     in_min => min_in_temp,  
81     in_ore => ore_in_temp,  
82     out_sec => sec_out_temp,  
83     out_min => min_out_temp,  
84     out_ore => ore_out_temp  
85 );  
86  
87 mv: manager_visualizzazione port map(  
88     clk => clk,  
89     reset => reset,  
90     sec_in => sec_out_temp,  
91     min_in => min_out_temp,  
92     ore_in => ore_out_temp,  
93     anodi => anodi,  
94     catodi => catodi  
95 );  
96  
97  
98 end Structural;
```

```
1 ## Clock signal
```

```

2 set_property -dict { PACKAGE_PIN E3      IOSTANDARD
3   LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35
4   Sch=clk100mhz
5
6 create_clock -add -name sys_clk_pin -period 10.00
7   -waveform {0 5} [get_ports {clk}];
8
9 ##Switches
10
11 set_property -dict { PACKAGE_PIN J15     IOSTANDARD
12   LVCMOS33 } [get_ports { switches[0] }];
13   #IO_L24N_T3_RS0_15 Sch=sw[0]
14
15 set_property -dict { PACKAGE_PIN L16     IOSTANDARD
16   LVCMOS33 } [get_ports { switches[1] }];
17   #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
18
19 set_property -dict { PACKAGE_PIN M13     IOSTANDARD
20   LVCMOS33 } [get_ports { switches[2] }];
21   #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
22
23 set_property -dict { PACKAGE_PIN R15     IOSTANDARD
24   LVCMOS33 } [get_ports { switches[3] }];
25   #IO_L13N_T2_MRCC_14 Sch=sw[3]
26
27 set_property -dict { PACKAGE_PIN R17     IOSTANDARD
28   LVCMOS33 } [get_ports { switches[4] }];
29   #IO_L12N_T1_MRCC_14 Sch=sw[4]
30
31 set_property -dict { PACKAGE_PIN T18     IOSTANDARD
32   LVCMOS33 } [get_ports { switches[5] }];
33   #IO_L7N_T1_D10_14 Sch=sw[5]
34
35
36 ##7 segment display
37
38 set_property -dict { PACKAGE_PIN T10     IOSTANDARD
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
636
636
637
638
638
639
639
640
641
642
643
644
644
645
646
646
647
647
648
648
649
649
650
651
652
653
654
654
655
656
656
657
657
658
658
659
659
660
661
662
663
664
664
665
666
666
667
667
668
668
669
669
670
671
672
673
673
674
675
675
676
676
677
677
678
678
679
679
680
681
682
683
683
684
685
685
686
686
687
687
688
688
689
689
690
691
692
693
693
694
695
695
696
696
697
697
698
698
699
699
700
701
702
703
703
704
705
705
706
706
707
707
708
708
709
709
710
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591

```

```

    LVCMOS33 } [get_ports { catodi[0] }];
    #IO_L24N_T3_A00_D16_14 Sch=ca
15 set_property -dict { PACKAGE_PIN R10      IOSTANDARD
    LVCMOS33 } [get_ports { catodi[1] }]; #IO_25_14
    Sch=cb
16 set_property -dict { PACKAGE_PIN K16      IOSTANDARD
    LVCMOS33 } [get_ports { catodi[2] }]; #IO_25_15
    Sch=cc
17 set_property -dict { PACKAGE_PIN K13      IOSTANDARD
    LVCMOS33 } [get_ports { catodi[3] }];
    #IO_L17P_T2_A26_15 Sch=cd
18 set_property -dict { PACKAGE_PIN P15      IOSTANDARD
    LVCMOS33 } [get_ports { catodi[4] }];
    #IO_L13P_T2_MRCC_14 Sch=ce
19 set_property -dict { PACKAGE_PIN T11      IOSTANDARD
    LVCMOS33 } [get_ports { catodi[5] }];
    #IO_L19P_T3_A10_D26_14 Sch=cf
20 set_property -dict { PACKAGE_PIN L18      IOSTANDARD
    LVCMOS33 } [get_ports { catodi[6] }];
    #IO_L4P_T0_D04_14 Sch=cg
21 set_property -dict { PACKAGE_PIN H15      IOSTANDARD
    LVCMOS33 } [get_ports { catodi[7] }];
    #IO_L19N_T3_A21_VREF_15 Sch=dp
22 set_property -dict { PACKAGE_PIN J17      IOSTANDARD
    LVCMOS33 } [get_ports { anodi[0] }];
    #IO_L23P_T3_FOE_B_15 Sch=an[0]
23 set_property -dict { PACKAGE_PIN J18      IOSTANDARD
    LVCMOS33 } [get_ports { anodi[1] }];

```

```

#IO_L23N_T3_FWE_B_15 Sch=an[1]

24 set_property -dict { PACKAGE_PIN T9      IOSTANDARD
                      LVCMOS33 } [get_ports { anodi[2] }];
#IO_L24P_T3_A01_D17_14 Sch=an[2]

25 set_property -dict { PACKAGE_PIN J14      IOSTANDARD
                      LVCMOS33 } [get_ports { anodi[3] }];
#IO_L19P_T3_A22_15 Sch=an[3]

26 set_property -dict { PACKAGE_PIN P14      IOSTANDARD
                      LVCMOS33 } [get_ports { anodi[4] }];
#IO_L8N_T1_D12_14 Sch=an[4]

27 set_property -dict { PACKAGE_PIN T14      IOSTANDARD
                      LVCMOS33 } [get_ports { anodi[5] }];
#IO_L14P_T2_SRCC_14 Sch=an[5]

28 set_property -dict { PACKAGE_PIN K2      IOSTANDARD
                      LVCMOS33 } [get_ports { anodi[6] }]; #IO_L23P_T3_35
Sch=an[6]

29 set_property -dict { PACKAGE_PIN U13      IOSTANDARD
                      LVCMOS33 } [get_ports { anodi[7] }];
#IO_L23N_T3_A02_D18_14 Sch=an[7]

30

31 ##Buttons

32 #set_property -dict { PACKAGE_PIN C12      IOSTANDARD
                      LVCMOS33 } [get_ports { CPU_RESETN }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

33 set_property -dict { PACKAGE_PIN N17      IOSTANDARD
                      LVCMOS33 } [get_ports { reset }]; #IO_L9P_T1_DQS_14
Sch=btnc

34 #set_property -dict { PACKAGE_PIN M18      IOSTANDARD

```

```

    LVCMOS33 } [get_ports { BTNU }]; #IO_L4N_T0_D05_14
    Sch=btnu
35 set_property -dict { PACKAGE_PIN P17      IOSTANDARD
    LVCMOS33 } [get_ports { btn_set }];
    #IO_L12P_T1_MRCC_14 Sch=btln
36 set_property -dict { PACKAGE_PIN M17      IOSTANDARD
    LVCMOS33 } [get_ports { btn_load }];
    #IO_L10N_T1_D15_14 Sch=btnr
37 #set_property -dict { PACKAGE_PIN P18      IOSTANDARD
    LVCMOS33 } [get_ports { BTND }];
    #IO_L9N_T1_DQS_D13_14 Sch=btnd

```

## 5.3 Traccia 5.3

- Estendere il componente sviluppato ai punti precedenti in modo che sia in grado di acquisire e memorizzare internamente fino ad un numero N di intertempi in corrispondenza di un ingresso di stop. Opzionalmente, il componente può prevedere una modalità di visualizzazione in cui, alla pressione di un bottone, vengano visualizzati sui display gli intertempi memorizzati (uno per ogni pressione).

### 5.3.1 Progetto e architettura

Per implementare queste modifiche, abbiamo apportato alcune modifiche al blocco centrale, ora denominato ‘Tempo Recorder’. Questo

blocco incorpora al suo interno il cronometro e un ‘Manager Intertempi’, responsabile della memorizzazione e dell’output dei vari intervalli di tempo. Questa funzionalità è resa possibile grazie all’utilizzo di due segnali distinti: ‘SAVE’, che consente di salvare un intervallo di tempo, e ‘SHOW’, che permette di visualizzare i vari intervalli di tempo registrati, uno per ogni pressione del pulsante. Questa struttura consente una gestione efficiente e precisa dei tempi, garantendo un’interazione fluida e intuitiva per l’utente

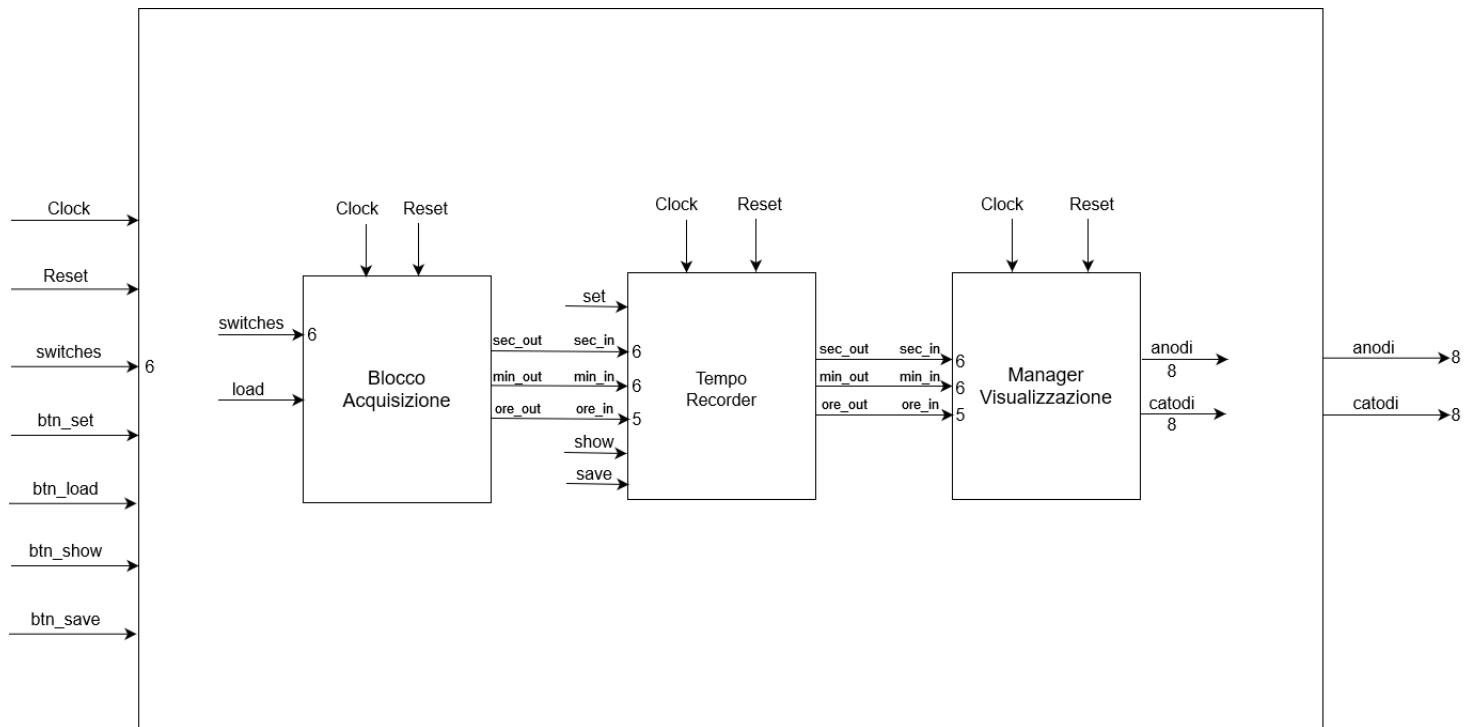
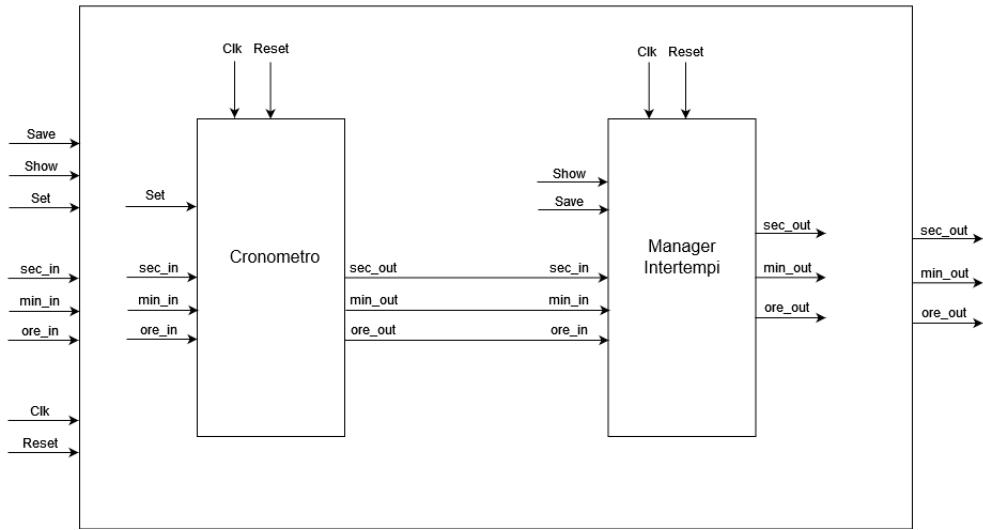


Figure 5.5: Architettura complessiva (Cronometro<sub>onBoard</sub>)

Figure 5.6: *TempoRecorder*

**Manager Intertempi** Il blocco ‘Manager Intertempi’ è progettato per gestire la memorizzazione e la visualizzazione degli intervalli di tempo. Questo blocco utilizza due debouncer per gestire i segnali di ‘save’ e ‘show’.

Il segnale ‘show’ funge da segnale di abilitazione per il contatore. Quando ‘show’ è attivo, il contatore incrementa il suo valore, permettendo di selezionare in sequenza i diversi intervalli di tempo memorizzati. Questo avviene perché il valore del contatore viene utilizzato come segnale di selezione per i registri di memorizzazione.

D’altra parte, il segnale ‘save’ funge da segnale di abilitazione per i registri di memorizzazione. Quando ‘save’ è attivo, i registri memorizzano i valori in ingresso corrispondenti al momento dell’attivazione.

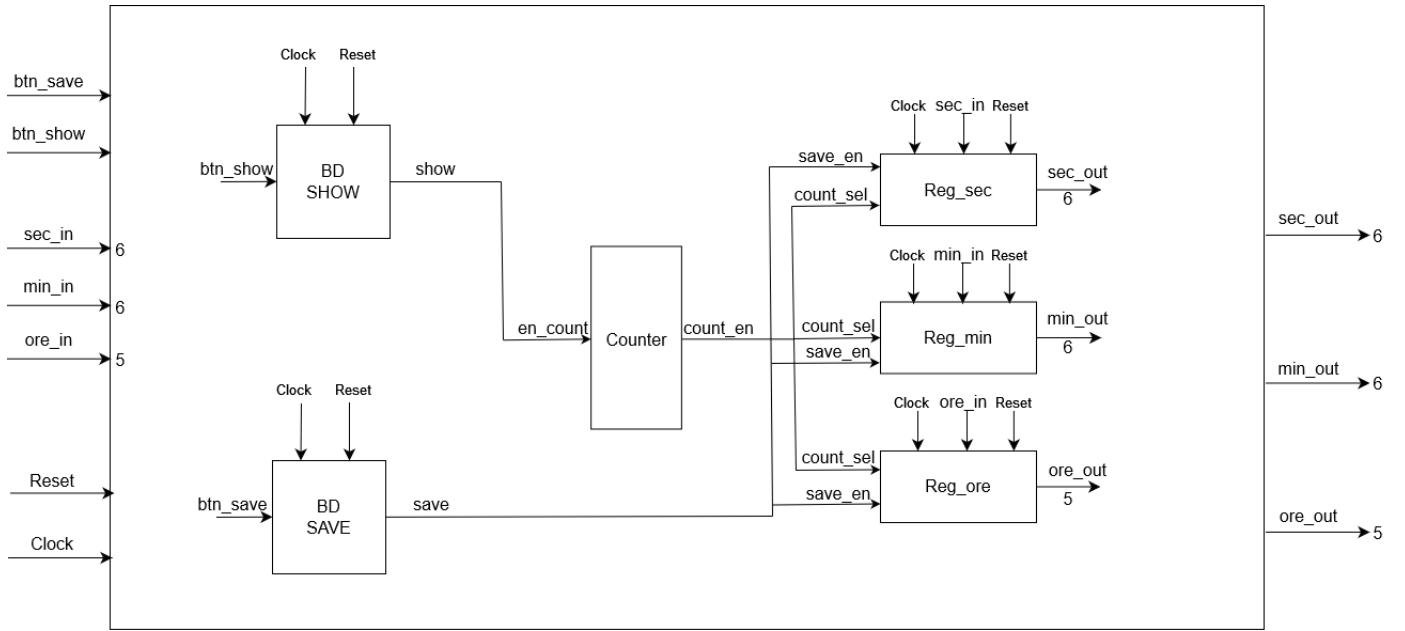


Figure 5.7: Manager<sub>intertempi</sub>

Il componente registro<sub>\_intertempi</sub> è progettato per memorizzare diversi valori di tempo (come i secondi) e fornire in uscita un valore specifico in base al segnale di selezione (sel). Questo viene realizzato utilizzando una combinazione di registri a scorrimento e un multiplexer.

Ogni bit del segnale di ingresso data<sub>\_in</sub> viene inviato a un registro a scorrimento (shift\_register) distinto. Quando il segnale di abilitazione (en) è alto, ogni registro a scorrimento memorizza il bit corrispondente di data<sub>\_in</sub> ad ogni ciclo di clock.

Il segnale di uscita di ogni registro a scorrimento viene poi inviato a un multiplexer (mux<sub>\_8\_1</sub>) corrispondente. Ogni multiplexer ha un segnale di selezione (sel) comune, il che significa che tutti i multiplexer selezionano e forniscono in uscita lo stesso bit dei loro ingressi.

In questo modo, il componente registro\_intertempi può memorizzare fino a N diversi valori di tempo e fornire in uscita un valore specifico in base al segnale di selezione. Ad esempio, se sel è impostato su 3, allora data\_out sarà il valore memorizzato dal quarto registro a scorrimento (dato che l'indicizzazione inizia da 0). Se sel viene cambiato in 5, allora data\_out sarà il valore memorizzato dal sesto registro a scorrimento, e così via.

### 5.3.2 Implementazione

#### Manager Intertempi

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity manager_intertempi is
5     port (
6         clk: in std_logic;
7         reset: in std_logic;
8         save: in std_logic;
9         show: in std_logic;
10        sec_in: in std_logic_vector(5 downto 0);
11        min_in: in std_logic_vector(5 downto 0);
12        ore_in: in std_logic_vector(4 downto 0);
13        sec_out: out std_logic_vector(5 downto 0);
14        min_out: out std_logic_vector(5 downto 0);
15        ore_out: out std_logic_vector(4 downto 0)
16    );

```

```
17  end manager_intertempi;  
18  
19  architecture Structural of manager_intertempi is  
20  
21  component registro_intertempi  
22      generic(  
23          N: integer := 8 --numero di registri  
24      );  
25      port(  
26          clk: in std_logic;  
27          reset: in std_logic;  
28          en: in std_logic;  
29          sel: in std_logic_vector(2 downto 0);  
30          data_in: in std_logic_vector(N-1 downto 0);  
31          data_out: out std_logic_vector(N-1 downto 0)  
32      );  
33  end component;  
34  
35  component counter_mod8  
36      Port (  
37          clock : in STD_LOGIC;  
38          reset : in STD_LOGIC;  
39          enable : in STD_LOGIC;  
40          counter : out STD_LOGIC_VECTOR (2 downto 0)  
41      );  
42  end component;  
43  
44  component ButtonDebouncer
```

```
45      generic (
46          CLK_period: integer := 10; -- periodo del
47              clock della board (in nanosecondi)
48          btn_noise_time: integer := 10000000 -- 
49              durata stimata dell'oscillazione del
50              bottone
51                  -- il
52                  valore
53                  di
54                  default
55                  [U+FFF]D
56                  10
57                  millisecondi
58
59      );
60
61      port (
62          RST: in std_logic;
63          CLK: in std_logic;
64          BTN: in std_logic;
65          CLEARED_BTN: out std_logic -- segnale di
66              output ripulito
67      );
68
69      end component;
```

```
70
71      signal show_en_temp: std_logic := '0';
72      signal save_en_temp: std_logic := '0';
73      signal sel_temp: std_logic_vector(2 downto 0) :=
74          (others => '0');
```

```
62 begin
63
64     deb_show: ButtonDebouncer port map(
65         RST => reset,
66         CLK => clk,
67         BTN => show,
68         CLEARED_BTN => show_en_temp
69     );
70
71     deb_save: ButtonDebouncer port map(
72         RST => reset,
73         CLK => clk,
74         BTN => save,
75         CLEARED_BTN => save_en_temp
76     );
77
78     counter: counter_mod8 port map(
79         clock => clk,
80         reset => reset,
81         enable => show_en_temp,
82         counter => sel_temp
83     );
84
85     reg_s: registro_intertempi --secondi
86         generic map(6)
87         port map(
88             clk => clk,
89             reset => reset,
```

```
90      en => save_en_temp,
91      sel => sel_temp,
92      data_in => sec_in,
93      data_out => sec_out
94  );
95
96  reg_m: registro_intertempi --minuti
97  generic map(6)
98  port map(
99      clk => clk,
100     reset => reset,
101     en => save_en_temp,
102     sel => sel_temp,
103     data_in => min_in,
104     data_out => min_out
105 );
106
107 reg_o: registro_intertempi --ore
108 generic map(5)
109 port map(
110     clk => clk,
111     reset => reset,
112     en => save_en_temp,
113     sel => sel_temp,
114     data_in => ore_in,
115     data_out => ore_out
116 );
117
```

```
118
119 end Structural;
```

## Registro intertempi

```

1      library IEEE;
2      use IEEE.STD_LOGIC_1164.ALL;
3
4      entity registro_intertempi is
5          generic(
6              N: integer := 8 --numero di registri
7          );
8          port(
9              clk: in std_logic;
10             reset: in std_logic;
11             en: in std_logic;
12             sel: in std_logic_vector(2 downto 0);
13             data_in: in std_logic_vector(N-1 downto 0);
14             data_out: out std_logic_vector(N-1 downto 0)
15         );
16     end registro_intertempi;
17
18     architecture Behavioral of registro_intertempi is
19
20         component shift_register
21             generic (
22                 n      : integer := 8
23             );
24             port (
```

```
25      clk, reset : in std_logic;
26
27      enable : in std_logic;
28
29      input : in std_logic;
30
31      Y : out std_logic_vector(n-1 downto 0)
32
33  );
34
35 end component;
36
37
38 component mux_8_1
39
40  port(
41
42      data : in STD_LOGIC_VECTOR (7 downto 0);
43
44      sel : in STD_LOGIC_VECTOR (2 downto 0);
45
46      output : out STD_LOGIC
47
48  );
49
50 end component;
51
52
53 type out_array is array (N-1 downto 0) of
54
55     std_logic_vector(6 downto 0);
56
57 signal output_temp: out_array;
58
59
60 begin
61
62
63 sr_n1_to_0: for i in N-1 downto 0 generate
64
65
66     sr_i: shift_register
67
68     generic map(7)
69
70     port map(
71
72         clk => clk,
73
74         reset => reset,
```

```

52         enable => en,
53
54         input => data_in(i),
55
56         Y => output_temp(i)
57     );
58
59
60     mux_i: mux_8_1 port map(
61
62         data => output_temp(i) & data_in(i),
63
64         sel => sel,
65
66         output => data_out(i)
67     );
68
69
70 end generate;
71
72
73 end Behavioral;

```

### 5.3.3 Sintesi su Board

Nel corso del processo di sintesi sulla scheda, abbiamo introdotto due nuovi pulsanti e li abbiamo associati a specifici pin. Il pulsante ‘show’, che è responsabile della visualizzazione degli intertempi memorizzati, è stato assegnato al pin M18. Ogni pressione di questo pulsante consente di visualizzare un intervallo di tempo diverso. D’altra parte, il pulsante ‘save’, che permette di salvare l’intervallo di tempo corrente, è stato assegnato al pin P18.

```

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

```

```
3
4 entity Cronometro_onBoard is
5   port (
6     clk: in std_logic;
7     reset: in std_logic;
8     switches: in std_logic_vector(5 downto 0);
9     btn_load: in std_logic;
10    btn_set: in std_logic;
11    btn_show: in std_logic;
12    btn_save: in std_logic;
13    catodi: out std_logic_vector(7 downto 0);
14    anodi: out std_logic_vector(7 downto 0)
15  );
16 end Cronometro_onBoard;
17
18 architecture Structural of Cronometro_onBoard is
19
20   component Blocco_acquisizione
21     port (
22       clk: in std_logic;
23       reset: in std_logic;
24       switches: in std_logic_vector(5 downto 0);
25       load: in std_logic;
26       sec_out: out std_logic_vector(5 downto 0);
27       min_out: out std_logic_vector(5 downto 0);
28       ore_out: out std_logic_vector(4 downto 0)
29     );
30   end component;
```

```
31
32 component tempo_recorder
33
34     port (
35         clk: in std_logic;
36         reset: in std_logic;
37         set: in std_logic;
38         save: in std_logic;
39         show: in std_logic;
40         sec_in: in std_logic_vector(5 downto 0);
41         min_in: in std_logic_vector(5 downto 0);
42         ore_in: in std_logic_vector(4 downto 0);
43         sec_out: out std_logic_vector(5 downto 0);
44         min_out: out std_logic_vector(5 downto 0);
45         ore_out: out std_logic_vector(4 downto 0)
46     );
47
48 end component;
49
50
51 component manager_visualizzazione
52
53     port (
54         clk: in std_logic;
55         reset: in std_logic;
56         sec_in: in std_logic_vector(5 downto 0);
57         min_in: in std_logic_vector(5 downto 0);
58         ore_in: in std_logic_vector(4 downto 0);
59         anodi: out std_logic_vector(7 downto 0);
60         catodi: out std_logic_vector(7 downto 0)
61     );
62
63 end component;
```

```
59
60     signal sec_in_temp: std_logic_vector(5 downto 0) :=
61         (others => '0');
62     signal min_in_temp: std_logic_vector(5 downto 0) :=
63         (others => '0');
64     signal ore_in_temp: std_logic_vector(4 downto 0) :=
65         (others => '0');
66
67 begin
68
69     ba: Blocco_acquisizione port map(
70         clk => clk,
71         reset => reset,
72         switches => switches,
73         load => btn_load,
74         sec_out => sec_in_temp,
75         min_out => min_in_temp,
76         ore_out => ore_in_temp
77     );
78
79     tr: tempo_recorder port map(
80         clk => clk,
```

```

81     reset => reset,
82
83     set => btn_set,
84
85     save => btn_save,
86
87     show => btn_show,
88
89     sec_in => sec_in_temp,
90
91     min_in => min_in_temp,
92
93     ore_in => ore_in_temp,
94
95     sec_out => sec_out_temp,
96
97     min_out => min_out_temp,
98
99     ore_out => ore_out_temp
100
101 );
102
103
104 mv: manager_visualizzazione port map(
105
106     clk => clk,
107
108     reset => reset,
109
110     sec_in => sec_out_temp,
111
112     min_in => min_out_temp,
113
114     ore_in => ore_out_temp,
115
116     anodi => anodi,
117
118     catodi => catodi
119
120 );
121
122
123
124 end Structural;

```

```

1  ##Buttons
2 #set_property -dict { PACKAGE_PIN C12      IO_STANDARD
3                         LVCMOS33 } [get_ports { CPU_RESETN }];
4 #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

```

```
3 set_property -dict { PACKAGE_PIN N17      IOSTANDARD
                      LVCMOS33 } [get_ports { reset }]; #IO_L9P_T1_DQS_14
                      Sch=btnc
4 set_property -dict { PACKAGE_PIN M18      IOSTANDARD
                      LVCMOS33 } [get_ports { btn_show }];
                      #IO_L4N_T0_D05_14 Sch=btnu
5 set_property -dict { PACKAGE_PIN P17      IOSTANDARD
                      LVCMOS33 } [get_ports { btn_load }];
                      #IO_L12P_T1_MRCC_14 Sch=btnl
6 set_property -dict { PACKAGE_PIN M17      IOSTANDARD
                      LVCMOS33 } [get_ports { btn_set }];
                      #IO_L10N_T1_D15_14 Sch=btnr
7 set_property -dict { PACKAGE_PIN P18      IOSTANDARD
                      LVCMOS33 } [get_ports { btn_save }];
                      #IO_L9N_T1_DQS_D13_14 Sch=btnd
```

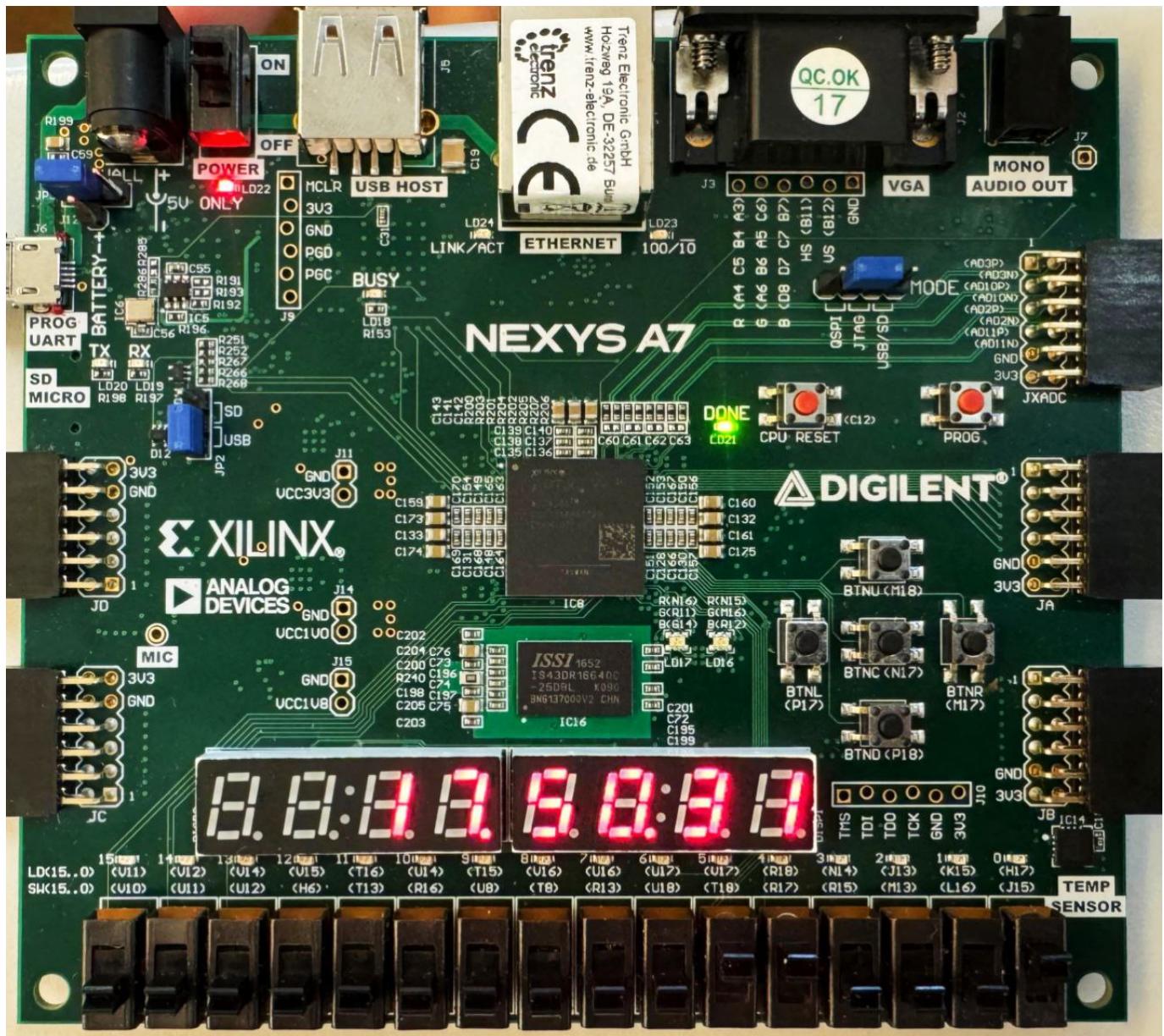


Figure 5.8: Implementazione su board

# Chapter 6

## Sistema PO\\_PC

### 6.1 Traccia

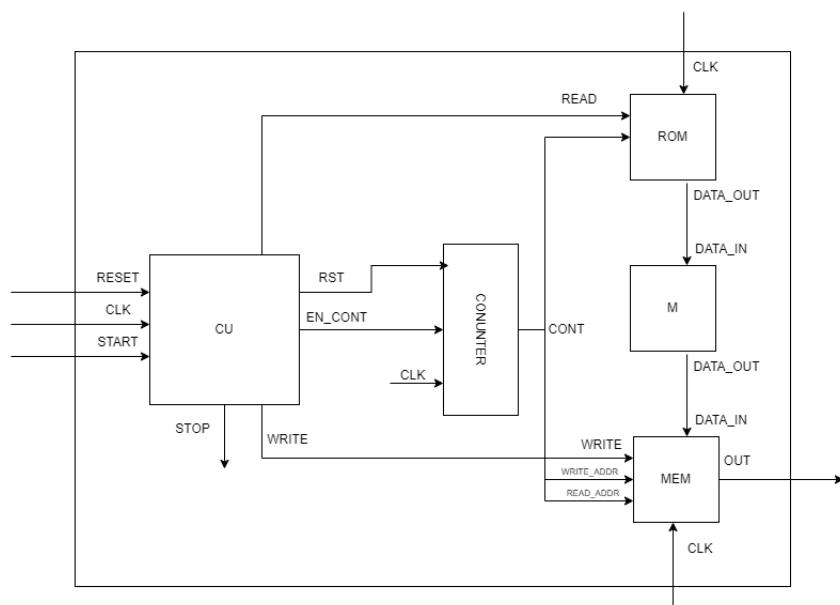
- Progettare, implementare in VHDL e verificare mediante simulazione un sistema dotato di una memoria ROM di  $N$  locazioni da 8 bit ciascuna, una macchina combinatoria  $M$  in grado di trasformare (secondo una funzione a scelta dello studente) la stringa di 8 bit letta dalla ROM in una stringa di 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.

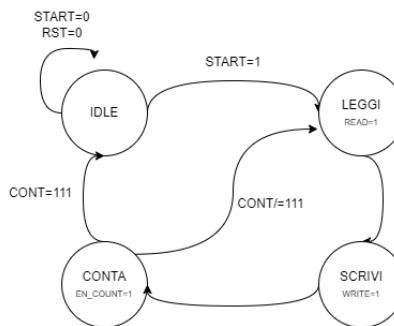
- Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando due bottoni per i segnali di read e reset rispettivamente e i led per la visualizzazione delle uscite della macchina istante per istante.

## 6.2 Progetto e architettura

Il design è stato costruito a partire dai componenti elementari quali: memoria ROM (8 locazioni 1 byte ciascuna), una memoria r/w, un contatore modulo 8 per scandire le locazioni delle memorie e una macchina combinatoria costruita appositamente e con la funzione di effettuare una negazione ed uno shift a destra di quattro posizioni della stringa in ingresso così da trasformare una stringa iniziale di 8 bit in un output finale di 4 bit.



La Control Unit descrive il comportamento del sistema e racchiude nel corpo del process la rappresentazione dell' automa a stati finiti. Il sistema parte in uno stato "IDLE" in cui i segnali "start" e "rst" sono inizializzati a '0'. Non appena arriva il segnale di start dall'esterno, e quindi il valore logico diventa alto (start=1), il sistema transita nello stato "LEGGI" in cui viene posto pari ad '1' il segnale "read". La stringa viene quindi letta e prelevata dalle memoria ROM, trasferita alla macchina combinatoria che effettua la propria elaborazione e rilascia in output il dato da 4 bit elaborato. A questo punto il sistema si trova in stato "SCRIVI" in cui il segnale "write" viene posto al valore logico alto, il dato viene prelevato dal buffer tra la macchina combinatoria e la memoria r/w e memorizzato. Nel quarto stato inizia il conteggio da parte del contatore (EN\_COUNT=1) e fintanto che il valore di conteggio non raggiunge il valore massimo (111), lo stato successivo per il sistema risulterà essere "LEGGI". Nel momento in cui il valore massimo di conteggio viene raggiunto, il sistema transita nello stato "IDLE" dal quale partito, i segnali vengono resettati ed il sistema è pronto per un nuovo ciclo di esecuzione all' arrivo di un nuovo segnale start.



## 6.3 Implementazione

I codici di implementazione per memoria ROM, mem r/w e contatore sono consultabili in appendice.

### 6.3.1 Control Unit

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CU is
5     port (
6         clk: in std_logic;
7         reset: in std_logic;
8         start: in std_logic;
9         count: in std_logic_vector(2 downto 0);
10
11        en_cont: out std_logic;
12        rst_cont: out std_logic;
13        read: out std_logic;
14        write: out std_logic
15    );
16 end CU;
17
18 architecture Behavioral of CU is
19
20 type stato is (IDLE,READ_STATE,WRITE_STATE,COUNT_STATE);
21
```

```
22 signal stato_corrente: stato:=IDLE;
23 signal stato_prossimo: stato:=IDLE;
24
25
26 begin
27
28 funzione_Stato_uscita: process(stato_corrente, start)
29 begin
30
31 read <= '0';
32 write <= '0';
33 en_cont <= '0';
34
35 case stato_corrente is
36 when IDLE=>
37     if start = '0' then
38         stato_prossimo<=IDLE;
39     else
40         stato_prossimo<=READ_STATE;
41     end if;
42
43 when READ_STATE=>
44     read<='1';
45     stato_prossimo<=WRITE_STATE;
46
47 when WRITE_STATE=>
48     write<='1';
49     stato_prossimo<=COUNT_STATE;
```

```
50
51      when COUNT_STATE=>
52          en_cont<='1';
53          if (count="111") then
54              stato_prossimo<=IDLE;
55          else
56              stato_prossimo<=READ_STATE;
57          end if;
58
59      end case;
60  end process;
61
62 mem: process(clk)
63 begin
64     if rising_edge(clk) then
65         if reset = '1' then
66             rst_cont <= '0';
67             stato_corrente <= IDLE;
68         else
69             stato_corrente <= stato_prossimo;
70         end if;
71     end if;
72  end process;
73
74
75
76 end Behavioral;
```

### 6.3.2 SistemaPO\_PC

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity sistemaPO_PC is
5     port(
6         clk: in std_logic;
7         reset: in std_logic;
8         start: in std_logic;
9         output: out std_logic_vector(3 downto 0);
10        stop: out std_logic
11    );
12 end sistemaPO_PC;
13
14 architecture Structural of sistemaPO_PC is
15
16 signal en_cont_temp: std_logic;
17 signal rst_cont_temp: std_logic;
18 signal read_temp: std_logic;
19 signal write_temp: std_logic;
20 signal count_temp: std_logic_vector(2 downto 0);
21 signal in_m_temp: std_logic_vector(7 downto 0);
22 signal out_m_temp: std_logic_vector(3 downto 0);
23 signal stop_temp: std_logic;
24
25 begin
26
27     unita_controllo: entity work.CU_sintesi port map(
```



```

56    );
57
58 MEM: entity work.mem port map(
59     clk => clk,
60     input => out_m_temp,
61     write_addr => count_temp,
62     read_addr => count_temp,
63     write_en => write_temp,
64     output => output
65 );
66
67 end Structural;

```

### 6.3.3 Macchina Combinatoria

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity macchinaM is
5   port (
6     data_in : in STD_LOGIC_VECTOR (7 downto 0); --
7       Stringa di 8 bit in ingresso
8     data_out : out STD_LOGIC_VECTOR (3 downto 0)
9       -- Stringa di 4 bit in uscita
10    );
11 end macchinaM;
12
13 architecture Dataflow of macchinaM is

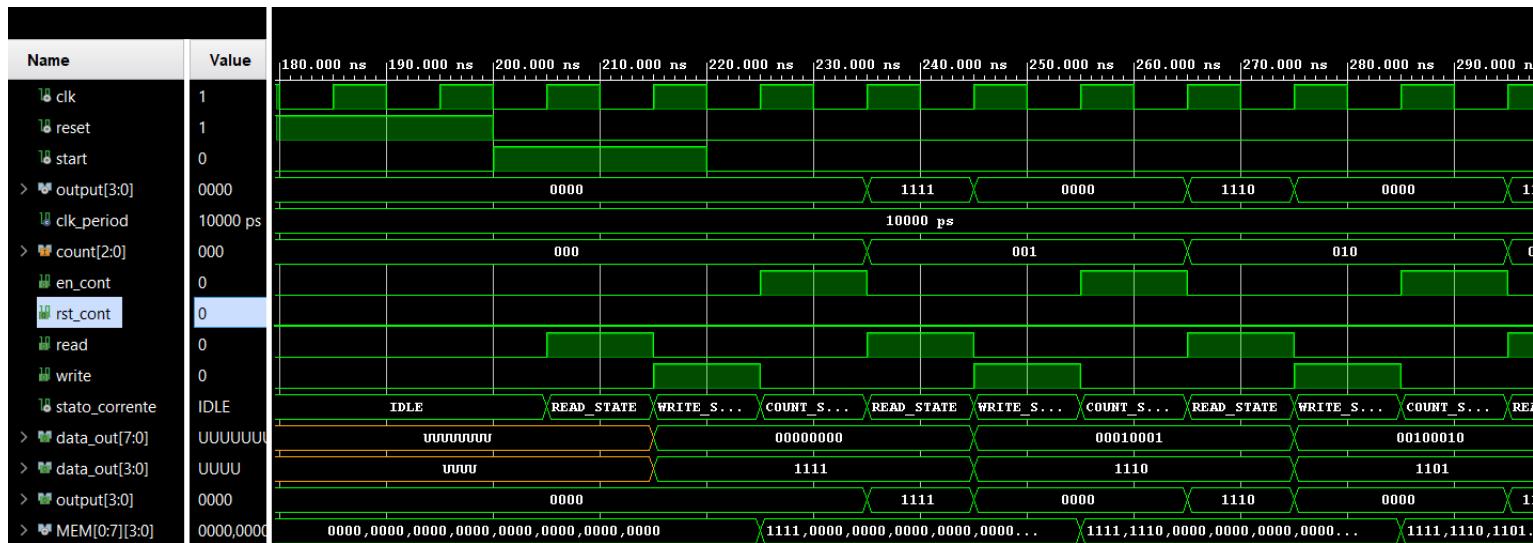
```

```

12
13 begin
14
15     -- Effettua il negato bit a bit della stringa
16     -- di input
17     -- e applica uno shift right di 4 posizioni
18     data_out <= not data_in(7 downto 4);
19
20 end Dataflow;

```

## 6.4 Simulazione



## 6.5 Implementazione su Board

Il codice di implementazione del button debouncer è consultabile in appendice. Per quanto concerne la fase di sintesi su board si è deciso di creare una nuova CU in modo da rispettare fedelmente quanto

richiesto dalla traccia. In questa control unit si fa in modo che il sistema vada avanti nell'acquisizione dei dati dalla memoria ROM solo nel momento in cui un segnale start arriva dall'esterno. Il controllo (tramite bottone) avviene quindi istante per istante.

### 6.5.1 CU\_sintesi

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CU_sintesi is
5   port (
6     clk: in std_logic;
7     reset: in std_logic;
8     start: in std_logic;
9     count: in std_logic_vector(2 downto 0);
10
11    en_cont: out std_logic;
12    rst_cont: out std_logic;
13    read: out std_logic;
14    write: out std_logic;
15    stop: out std_logic
16  );
17 end CU_sintesi;
18
19 architecture Behavioral of CU_sintesi is
20
21 type stato is (IDLE,READ_STATE,WRITE_STATE,COUNT_STATE);

```

```
22
23 signal stato_corrente: stato:=IDLE;
24 signal stato_prossimo: stato:=IDLE;
25
26
27 begin
28
29 funzione_Stato_uscita: process(stato_corrente, start)
30 begin
31
32 read <= '0';
33 write <= '0';
34 en_cont <= '0';
35
36 case stato_corrente is
37 when IDLE=>
38     if start = '0' then
39         stato_prossimo<=IDLE;
40     else
41         stato_prossimo<=READ_STATE;
42     end if;
43
44 when READ_STATE=>
45     read<='1';
46     stato_prossimo<=WRITE_STATE;
47
48 when WRITE_STATE=>
49     write<='1';
```

```
50      stato_prossimo<=COUNT_STATE;
51
52  when COUNT_STATE=>
53      en_cont<='1';
54      if (count="111") then
55          stop <= '1';
56      else
57          stop <= '0';
58      end if;
59      stato_prossimo<=IDLE;
60
61  end case;
62 end process;
63
64 mem: process(clk)
65 begin
66     if rising_edge(clk) then
67         if reset = '1' then
68             rst_cont <= '0';
69             stato_corrente <= IDLE;
70         else
71             stato_corrente <= stato_prossimo;
72         end if;
73     end if;
74 end process;
75
76
77
```

78      **end Behavioral;**

### 6.5.2 SistemaOnBoard

Nell'implementazione sulla scheda, abbiamo utilizzato un button debouncer per filtrare e pulire il segnale di avvio. Questo assicura che il segnale di avvio sia privo di rumore indesiderato, migliorando l'affidabilità del sistema.

Per quanto riguarda le indicazioni visive, abbiamo associato l'uscita del sistema ai primi quattro LED. Questo è stato possibile grazie alla specifica dei constraint fisici della scheda. Inoltre, abbiamo utilizzato l'ultimo LED per indicare il segnale di stop, fornendo un feedback visivo immediato all'utente.

Per quanto riguarda l'interfaccia utente, abbiamo mappato il segnale di reset al pin N17 e il segnale di avvio al pin P17.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Sistema_onBoard is
5   port (
6     clock: in std_logic;
7     reset: in std_logic;
8     start: in std_logic;
9     Y: out std_logic_vector(3 downto 0);
10    stop: out std_logic

```

```
11      );
12 end Sistema_onBoard;
13
14 architecture Structural of Sistema_onBoard is
15
16 component ButtonDebouncer is
17
18     generic(
19         CLK_period: integer := 10;
20         btn_noise_time: integer := 10000000
21     );
22
23     port(
24         RST: in std_logic;
25         CLK: in std_logic;
26         BTN: in std_logic;
27         CLEARED_BTN: out std_logic -- segnale di
28                         output ripulito
29     );
30
31 end component;
32
33
34 component sistemaPO_PC is
35
36     port(
37         clk: in std_logic;
38         reset: in std_logic;
39         start: in std_logic;
40         output: out std_logic_vector(3 downto 0);
41         stop: out std_logic
42     );
43
44 end component;
```

```

38
39     signal cleared_start: std_logic;
40
41 begin
42
43     deb_s: ButtonDebouncer generic map(
44         CLK_period => 10,
45         btn_noise_time => 10000000
46     )
47
48     port map(
49         RST => reset,
50         CLK => clock,
51         BTN => start,
52         CLEARED_BTN => cleared_start
53     );
54
55     sys: sistemaPO_PC port map(
56         clk => clock,
57         reset => reset,
58         start => cleared_start,
59         output => Y,
60         stop => stop
61     );
62 end Structural;

```

```

1 ## Clock signal
2 set_property -dict { PACKAGE_PIN E3      IO_STANDARD
3                         LVCMOS33 } [get_ports { clock }];

```

```
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
3 create_clock -add -name sys_clk_pin -period 10.00
   -waveform {0 5} [get_ports {clock}];
4
5 ## LEDs
6 set_property -dict { PACKAGE_PIN H17      IOSTANDARD
   LVCMOS33 } [get_ports { Y[0] }]; #IO_L18P_T2_A24_15
   Sch=led[0]
7 set_property -dict { PACKAGE_PIN K15      IOSTANDARD
   LVCMOS33 } [get_ports { Y[1] }]; #IO_L24P_T3_RS1_15
   Sch=led[1]
8 set_property -dict { PACKAGE_PIN J13      IOSTANDARD
   LVCMOS33 } [get_ports { Y[2] }]; #IO_L17N_T2_A25_15
   Sch=led[2]
9 set_property -dict { PACKAGE_PIN N14      IOSTANDARD
   LVCMOS33 } [get_ports { Y[3] }]; #IO_L8P_T1_D11_14
   Sch=led[3]
10 #set_property -dict { PACKAGE_PIN R18      IOSTANDARD
   LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14
   Sch=led[4]
11 #set_property -dict { PACKAGE_PIN V17      IOSTANDARD
   LVCMOS33 } [get_ports { LED[5] }];
   #IO_L18N_T2_A11_D27_14 Sch=led[5]
12 #set_property -dict { PACKAGE_PIN U17      IOSTANDARD
   LVCMOS33 } [get_ports { LED[6] }];
   #IO_L17P_T2_A14_D30_14 Sch=led[6]
13 #set_property -dict { PACKAGE_PIN U16      IOSTANDARD
   LVCMOS33 } [get_ports { LED[7] }];
```

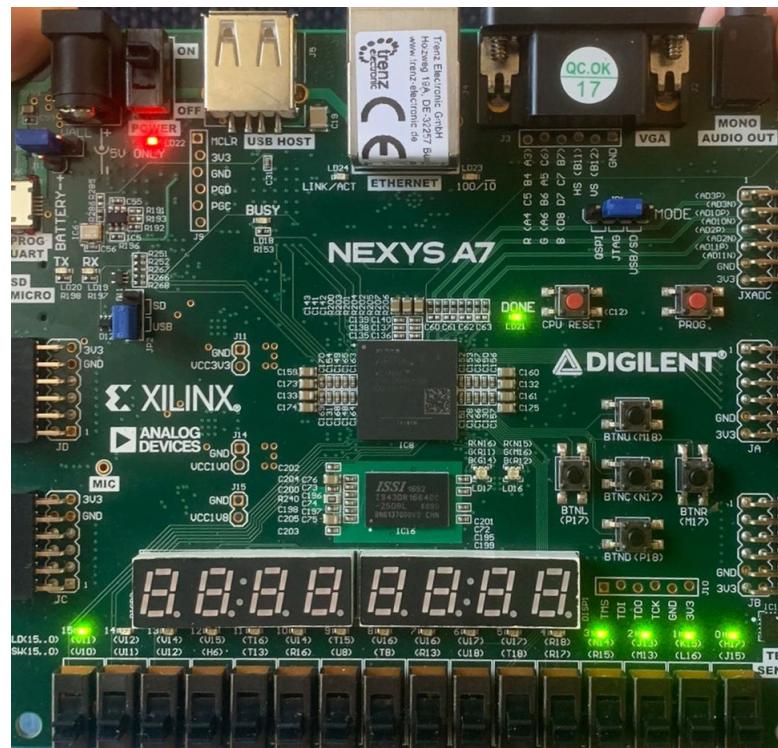
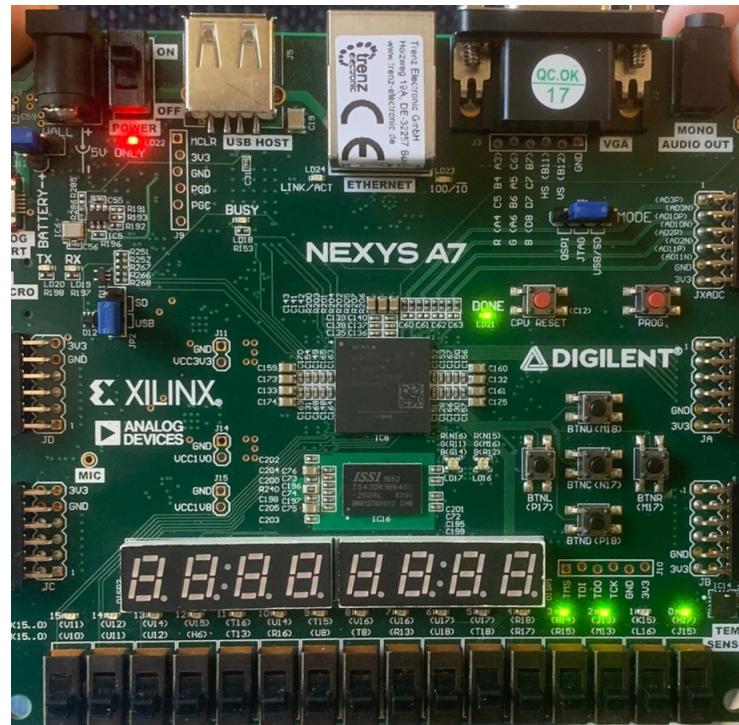
```

    #IO_L18P_T2_A12_D28_14 Sch=led[7]
14 #set_property -dict { PACKAGE_PIN V16      IOSTANDARD
    LVCMOS33 } [get_ports { LED[8] }];
    #IO_L16N_T2_A15_D31_14 Sch=led[8]
15 #set_property -dict { PACKAGE_PIN T15      IOSTANDARD
    LVCMOS33 } [get_ports { LED[9] }];
    #IO_L14N_T2_SRCC_14 Sch=led[9]
16 #set_property -dict { PACKAGE_PIN U14      IOSTANDARD
    LVCMOS33 } [get_ports { LED[10] }];
    #IO_L22P_T3_A05_D21_14 Sch=led[10]
17 #set_property -dict { PACKAGE_PIN T16      IOSTANDARD
    LVCMOS33 } [get_ports { LED[11] }];
    #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
18 #set_property -dict { PACKAGE_PIN V15      IOSTANDARD
    LVCMOS33 } [get_ports { LED[12] }];
    #IO_L16P_T2_CSI_B_14 Sch=led[12]
19 #set_property -dict { PACKAGE_PIN V14      IOSTANDARD
    LVCMOS33 } [get_ports { LED[13] }];
    #IO_L22N_T3_A04_D20_14 Sch=led[13]
20 #set_property -dict { PACKAGE_PIN V12      IOSTANDARD
    LVCMOS33 } [get_ports { LED[14] }];
    #IO_L20N_T3_A07_D23_14 Sch=led[14]
21 set_property -dict { PACKAGE_PIN V11      IOSTANDARD
    LVCMOS33 } [get_ports { stop }];
    #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
22
23 ##Buttons
24 #set_property -dict { PACKAGE_PIN C12      IOSTANDARD

```

```
    LVCMOS33 } [get_ports { CPU_RESETN }];

    #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
25 set_property -dict { PACKAGE_PIN N17      IOSTANDARD
    LVCMOS33 } [get_ports { reset }]; #IO_L9P_T1_DQS_14
    Sch=btnc
26 #set_property -dict { PACKAGE_PIN M18      IOSTANDARD
    LVCMOS33 } [get_ports { reset }]; #IO_L4N_T0_D05_14
    Sch=btnu
27 set_property -dict { PACKAGE_PIN P17      IOSTANDARD
    LVCMOS33 } [get_ports { start }];
    #IO_L12P_T1_MRCC_14 Sch=btnl
28 #set_property -dict { PACKAGE_PIN M17      IOSTANDARD
    LVCMOS33 } [get_ports { BTNR }]; #IO_L10N_T1_D15_14
    Sch=btnr
29 #set_property -dict { PACKAGE_PIN P18      IOSTANDARD
    LVCMOS33 } [get_ports { BTND }];
    #IO_L9N_T1_DQS_D13_14 Sch=btnd
```



# Chapter 7

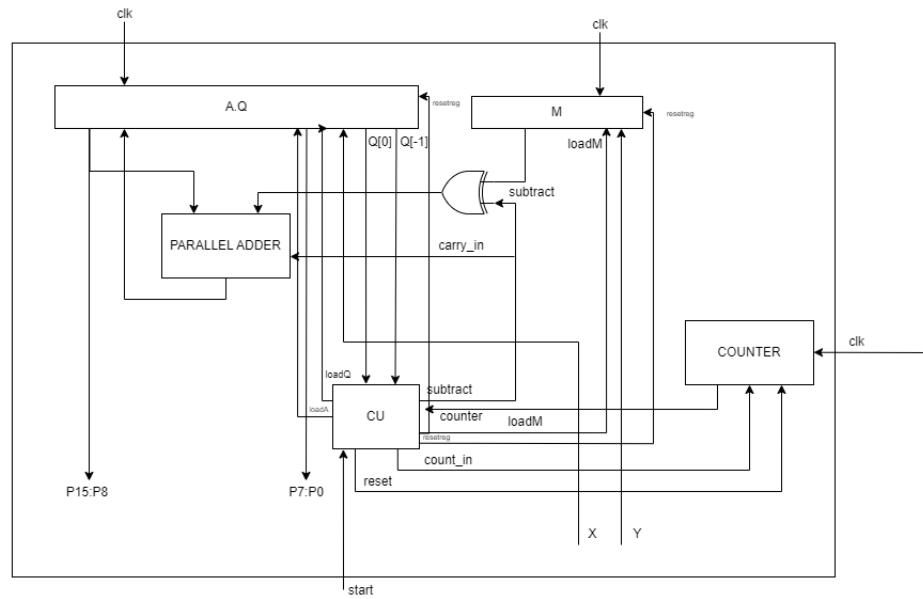
## Moltiplicatore Booth

### 7.1 Traccia

- Progettare, implementare in VHDL e simulare una macchina moltiplicatore di Booth in grado di effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna.
- 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 Progetto e Architettura

Il design del moltiplicatore di Booth è organizzato in due principali componenti funzionali: un'unità operativa e un'unità di controllo. Queste due unità collaborano per eseguire l'operazione di moltiplicazione binaria. L'unità operativa include: Contatore Modulo 8: Questo contatore è utilizzato per generare i segnali di controllo ciclico che guidano l'esecuzione degli step di moltiplicazione.. Shift Register da 17 bit: Questo registro funge da deposito principale per i dati durante il processo di moltiplicazione. È organizzato per contenere i fattori coinvolti nella moltiplicazione, tra cui il moltiplicatore ( $Q$ ), l'accumulatore ( $A$ ) e un bit di estensione ( $Q[-1]$ ) che viene utilizzato per gestire i casi durante l'esecuzione dell'algoritmo di Booth. Ripple Carry Adder: Questo componente esegue l'addizione parziale dei prodotti parziali generati durante il processo di moltiplicazione. È responsabile di sommare o sottrarre i valori accumulati nell'accumulatore e del moltiplicatore. Registro M da 8 bit: Questo registro contiene il fattore  $Y$ , che rappresenta il moltiplicatore originale. Durante il processo di moltiplicazione, il registro M fornisce il moltiplicatore corrente, che viene utilizzato per generare i prodotti parziali. L'unità di controllo, d'altra parte, supervisiona e coordina l'intera esecuzione del processo di moltiplicazione. Si occupa di generare i segnali di controllo appropriati per sincronizzare le operazioni.

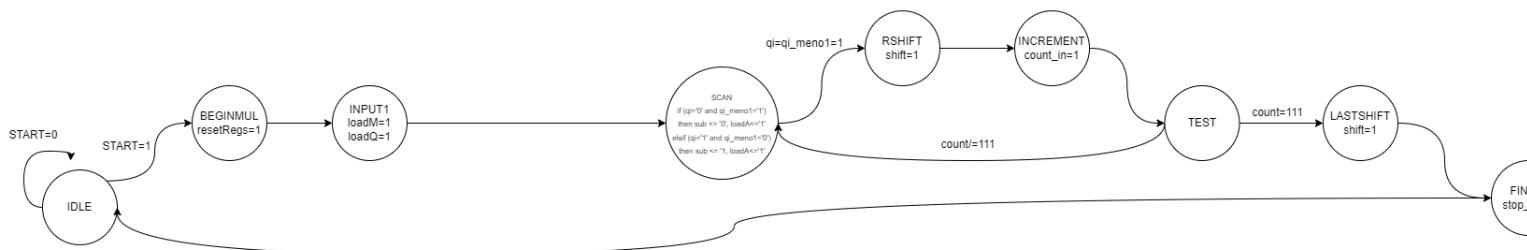


Il sistema del moltiplicatore di Booth segue un preciso flusso di esecuzione, guidato da una sequenza di stati che definiscono le varie fasi dell'operazione di moltiplicazione binaria.

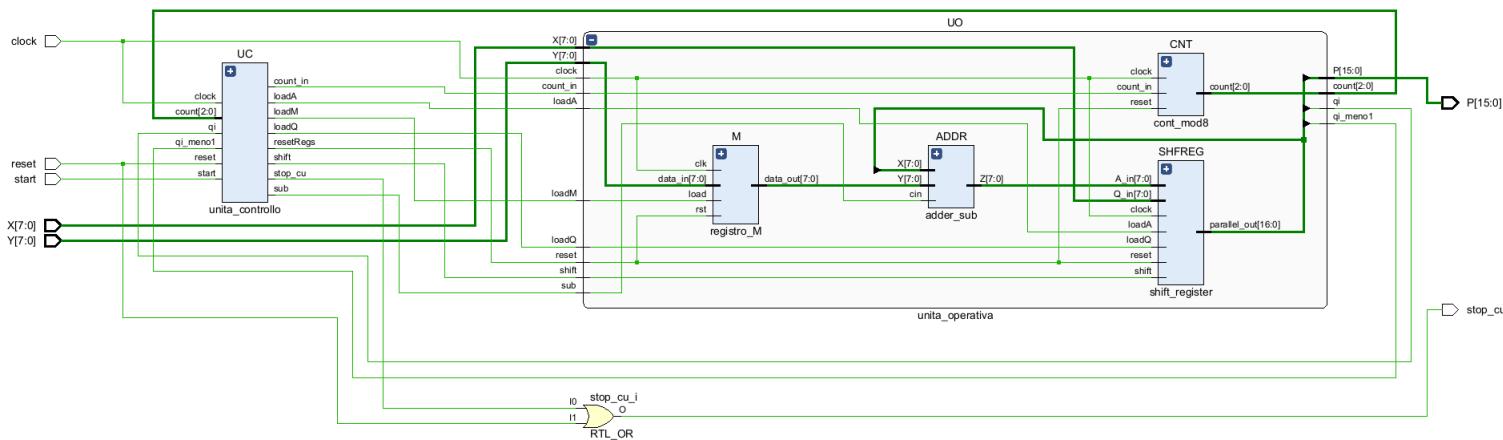
1. Stato **IDLE**: Questo è lo stato iniziale del sistema, in cui il moltiplicatore si trova in attesa del segnale di start per avviare l'operazione di moltiplicazione.
2. Stato **beginMul**: Quando il segnale di start viene rilevato, il sistema passa a questo stato. Qui, vengono azzerati lo shift register e il contatore. In questo modo, la parte A dello shift register, che corrisponde ai bit 16 downto 9, è già popolata con zeri, pronta per l'inizio dell'operazione di moltiplicazione.
3. Stato di **input**: In questo stato, vengono caricati gli operandi nel registro Q e nel registro M. Il registro Q contiene il moltiplicando X e il registro M contiene il moltiplicatore Y.

catore, mentre il registro M contiene il multiplicando.

4. Stato di **scan**: Durante questo stato, il sistema esamina gli ultimi due bit dello shift register per decidere se effettuare un'operazione di somma, differenza o nessuna operazione tra i registri A e M. Questa decisione guida il processo di generazione dei prodotti parziali.
5. Stato di **rshift**: Qui, il sistema esegue uno shift right dell'intero shift register. Questo movimento è essenziale per il corretto funzionamento dell'algoritmo di moltiplicazione di Booth.
6. Stato di **increment**: Durante questo stato, il sistema incrementa il contatore e testa il valore del contatore. Se il valore di conteggio raggiunge "111", il sistema procede all'ultimo shift (lastshf), altrimenti torna allo stato di scan.
7. Stato **lastshf**: In questo stato, il sistema esegue l'ultimo shift del registro. Questo passo è necessario per completare il processo di moltiplicazione.
8. Stato di **finish**: Una volta completato il processo di moltiplicazione, il sistema torna allo stato IDLE per attendere ulteriori operazioni.



## 7.3 Implementazione



### 7.3.1 Booth Multiplier

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5
6
7 entity molt_booth is
8 port (
9     X, Y: in std_logic_vector (7 downto 0);

```

```
10      clock, reset, start: in std_logic;
11      P: out std_logic_vector (15 downto 0);
12      stop_cu: out std_logic
13  );
14 end molt_booth;
15
16 architecture Structural of molt_booth is
17
18     component unita_controllo is
19         port(
20             qi,qi_menol: in std_logic;
21             clock, reset, start: in std_logic;
22             count: in std_logic_vector(2 downto 0);
23             loadA, loadM, loadQ: out std_logic;
24             resetRegs: out std_logic;
25             shift: out std_logic;
26             sub: out std_logic;
27             count_in: out std_logic;
28             stop_cu: out std_logic
29         );
30     end component;
31
32     component unita_operativa is
33         port(
34             X, Y: in std_logic_vector (7 downto 0);
35             clock, reset: in std_logic;
36             loadQ, loadA, loadM: in std_logic;
37             shift: in std_logic;
```



63      **end** Structural;

### 7.3.2 Unità di Controllo

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity unita_controllo is
6      port(
7          qi,qi_meno1: in std_logic;
8          clock, reset, start: in std_logic;
9          count: in std_logic_vector(2 downto 0);
10         loadA, loadM, loadQ: out std_logic;
11         resetRegs: out std_logic;
12         shift: out std_logic;
13         sub: out std_logic;
14         count_in: out std_logic;
15         stop_cu: out std_logic
16     );
17 end unita_controllo;
18
19 architecture Behavioral of unita_controllo is
20
21 type state is
22     (idle,beginMul,input,scan,rshift,lastshf,increment,test,fini
23
24     signal current_state, next_state: state;
```

```
24 begin
25
26 reg_stato: process(clock)
27 begin
28     if(clock'event and clock='1') then
29         if(reset='1') then
30             current_state <= idle;
31         else
32             current_state <= next_state;
33         end if;
34     end if;
35 end process;
36
37 comb: process(current_state,start,count)
38 begin
39     loadA <= '0';
40     loadM <= '0';
41     loadQ <= '0';
42     shift <= '0';
43     sub <= '0';
44     count_in <= '0';
45     stop_cu <= '0';
46     resetRegs <= '0';
47
48 CASE current_state is
49     WHEN idle =>
50         if(start='1') then
51             next_state <= beginMul;
```

```
52         else
53             next_state <= idle;
54         end if;
55
56         WHEN beginMul => -- Azzero i registri
57             della UO
58             resetRegs <= '1';
59             next_state <= input;
60
61
62         WHEN input =>
63             loadM <= '1'; -- carico gli operandi
64             loadQ <= '1';
65             next_state <= scan;
66
67
68         WHEN scan =>
69             if(qi=qi_menol) then
70                 next_state <= rshift;
71             else -- scelgo cosa fare e carico A
72                 con il risultato
73
74                 if(qi='0' and qi_menol='1') then
75                     sub <= '0';
76                     loadA<='1';
77
78                 elsif (qi='1' and qi_menol='0')
79                     then
80                         sub <= '1';
81                         loadA<='1';
82
83                     end if;
84
85                     next_state <= rshift;
86
87         end if;
```

```
77      WHEN rshift =>
78          shift <= '1';
79          next_state <= increment;
80
81      WHEN increment =>
82          count_in<='1';
83          next_state <= test;
84
85      WHEN test => -- testo il conteggio
86          if(count="111") then
87              next_state<=lastshf;
88          else
89              next_state <= scan;
90          end if;
91
92      WHEN lastshf =>
93          shift <='1';
94          next_state <= finish;
95
96      WHEN finish =>
97
98          stop_cu <= '1';
99          next_state <= idle;
100
101
102      end CASE;
103
104
105  end process;
106
107
108 end Behavioral;
```

### 7.3.3 Unità Operativa

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity unita_operativa is
6     port(
7         X, Y: in std_logic_vector(7 downto 0);
8         clock, reset: in std_logic;
9         loadQ, loadA, loadM: in std_logic;
10        shift: in std_logic;
11        sub: in std_logic;
12        count_in: in std_logic;
13        count: out std_logic_vector(2 downto 0);
14        P: out std_logic_vector(15 downto 0);
15        qi, qi_menol: out std_logic
16
17    );
18
19 end unita_operativa;
20
21 architecture Structural of unita_operativa is
22
23     component adder_sub is
24         port(
25             X, Y: in std_logic_vector(7 downto 0);
26             cin: in std_logic;
27             Z: out std_logic_vector(7 downto 0);
```

```
28         cout: out std_logic
29     );
30 end component;
31
32 component cont_mod8 is
33 port(
34     clock, reset: in std_logic;
35     count_in: in std_logic; --segnali di
36     count: out std_logic_vector(2 downto 0)
37     --uscita 3 bit che rappresenta il valore
38     del contatore
39 );
40 end component;
41
42 component shift_register is
43 port(
44     A_in: in std_logic_vector(7 downto 0);
45     Q_in: in std_logic_vector(7 downto 0);
46     clock, reset, loadQ, loadA, shift: in
47     std_logic; -- Segnali di clock, reset,
48     caricamento e shift.
49 parallel_out: out std_logic_vector(16
50     downto 0)
51 );
52 end component;
53
54 component registro_M is
```

```

50      port (
51          clk : in std_logic;
52          rst : in std_logic;
53          load : in std_logic;
54          data_in : in std_logic_vector(7 downto 0);
55          data_out : out std_logic_vector(7 downto 0)
56      );
57
58  end component;
59
60
61 --signal adder_to_acc: std_logic_vector (7 downto
62   0);
63
64 signal tmp_A: std_logic_vector (7 downto 0);
65 signal out_M: std_logic_vector (7 downto 0);
66 signal tmp_cout: std_logic;
67 signal tmp_Z: std_logic_vector (7 downto 0);
68 signal tmp_P: std_logic_vector (16 downto 0);
69
70 begin
71
72
73     P <= tmp_P(16 downto 1);
74     qi <= tmp_P(1);
75     qi_menol <= tmp_P(0);
76
77
78     M: registro_M port map(clock,reset,loadM,Y,out_M);
79         -- OK
80
81
82     CNT: cont_mod8 port
83         map(clock,reset,count_in,count); -- OK
84

```

```

75      tmp_A <= tmp_P(16 downto 9);

76

77      ADDR: adder_sub port
78          map(tmp_A,out_M,sub,tmp_Z,tmp_cout);

79      SHFREG: shift_Register port
80          map(tmp_Z,X,clock,reset,loadQ,loadA,shift,tmp_P);

81

82

83  end Structural;

```

### 7.3.4 Shift Register

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity unita_operativa is
6     port (
7         X, Y: in std_logic_vector (7 downto 0);
8         clock, reset: in std_logic;
9         loadQ, loadA, loadM: in std_logic;
10        shift: in std_logic;
11        sub: in std_logic;
12        count_in: in std_logic;
13        count: out std_logic_vector (2 downto 0);
14        P: out std_logic_vector(15 downto 0);

```

```
15      qi, qi_menol: out std_logic
16
17
18  );
19 end unita_operativa;
20
21 architecture Structural of unita_operativa is
22
23 component adder_sub is
24   port(
25     X, Y: in std_logic_vector(7 downto 0);
26     cin: in std_logic;
27     Z: out std_logic_vector(7 downto 0);
28     cout: out std_logic
29   );
30 end component;
31
32 component cont_mod8 is
33   port(
34     clock, reset: in std_logic;
35     count_in: in std_logic; --segnali di
36     abilitazione per il contatore
37     count: out std_logic_vector(2 downto 0)
38     --uscita 3 bit che rappresenta il valore
39     del contatore
40   );
41 end component;
```

```

40 component shift_register is
41     port (
42         A_in: in std_logic_vector(7 downto 0);
43         Q_in: in std_logic_vector(7 downto 0);
44         clock, reset, loadQ, loadA , shift: in
45             std_logic; -- Segnali di clock, reset,
46             caricamento e shift.
47
48         parallel_out: out std_logic_vector(16
49             downto 0)
50     );
51
52 end component;
53
54
55 component registro_M is
56     port (
57         clk : in std_logic;
58         rst : in std_logic;
59         load : in std_logic;
60         data_in : in std_logic_vector(7 downto 0);
61         data_out : out std_logic_vector(7 downto 0)
62     );
63
64 end component;
65
66
67 --signal adder_to_acc: std_logic_vector (7 downto
68     0);
69
70 signal tmp_A: std_logic_vector (7 downto 0);
71
72 signal out_M: std_logic_vector (7 downto 0);
73
74 signal tmp_cout: std_logic;
75
76 signal tmp_Z: std_logic_vector (7 downto 0);

```

```

64      signal tmp_P: std_logic_vector (16 downto 0);
65 begin
66
67      P <= tmp_P(16 downto 1);
68      qi <= tmp_P(1);
69      qi_menol <= tmp_P(0);
70
71      M: registro_M port map(clock,reset,loadM,Y,out_M);
72          -- OK
73
74      CNT: cont_mod8 port
75          map(clock,reset,count_in,count); -- OK
76
77      tmp_A <= tmp_P(16 downto 9);
78
79      ADDR: adder_sub port
80          map(tmp_A,out_M,sub,tmp_Z,tmp_cout);
81
82      SHFREG: shift_Register port
83          map(tmp_Z,X,clock,reset,loadQ,loadA,shift,tmp_P);
84
85
86
87
88
89
90
91
92
93 end Structural;

```

### 7.3.5 Registro M

library ieee;

```
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity registro_M is
6      port (
7          clk : in std_logic;
8          rst : in std_logic;
9          load : in std_logic;
10         data_in : in std_logic_vector(7 downto 0);
11         data_out : out std_logic_vector(7 downto 0)
12     );
13 end entity registro_M;
14
15 architecture Behavioral of registro_M is
16     signal m_reg : std_logic_vector(7 downto 0);
17 begin
18     process(clk)
19     begin
20         if (clk'event and clk='1') then
21             if rst = '1' then
22                 m_reg <= (others => '0');
23             elsif load = '1' then
24                 m_reg <= data_in;
25             end if;
26         end if;
27     end process;
28     data_out <= m_reg;
29
```

30      **end architecture;**

### 7.3.6 Addizionatore - Sottrattore

```

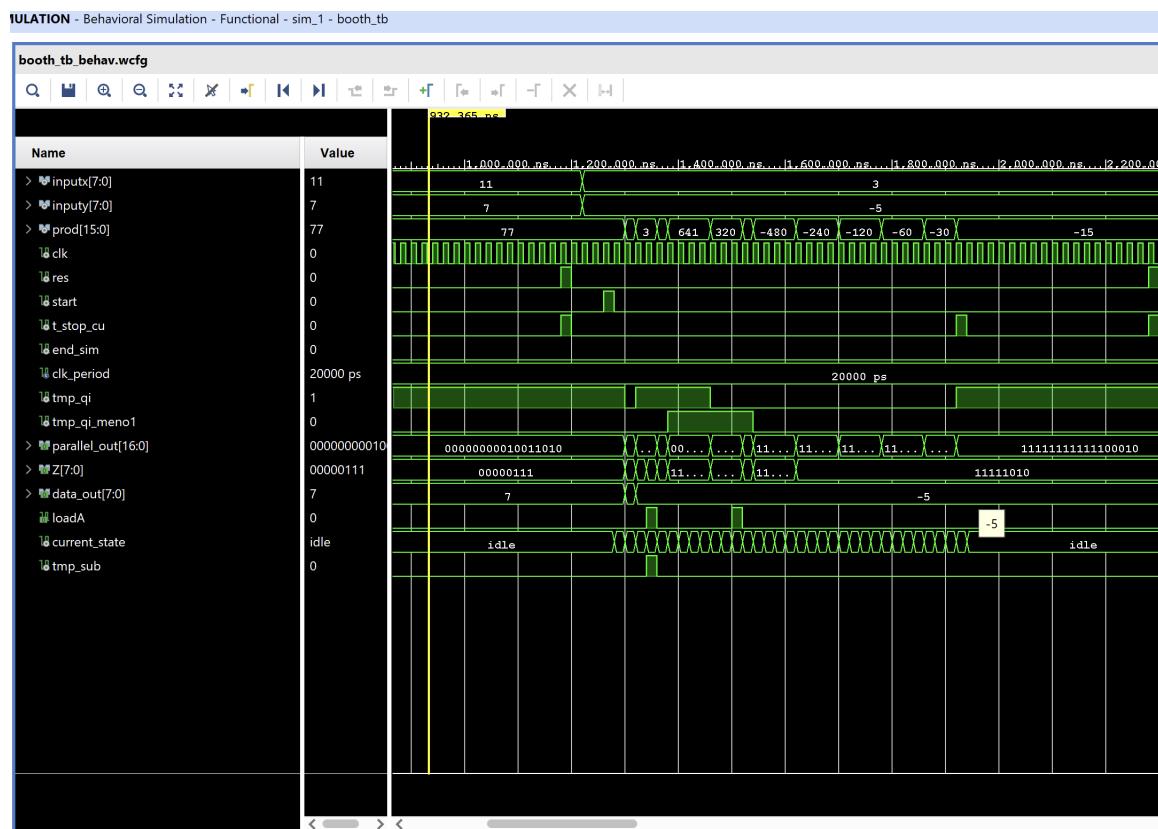
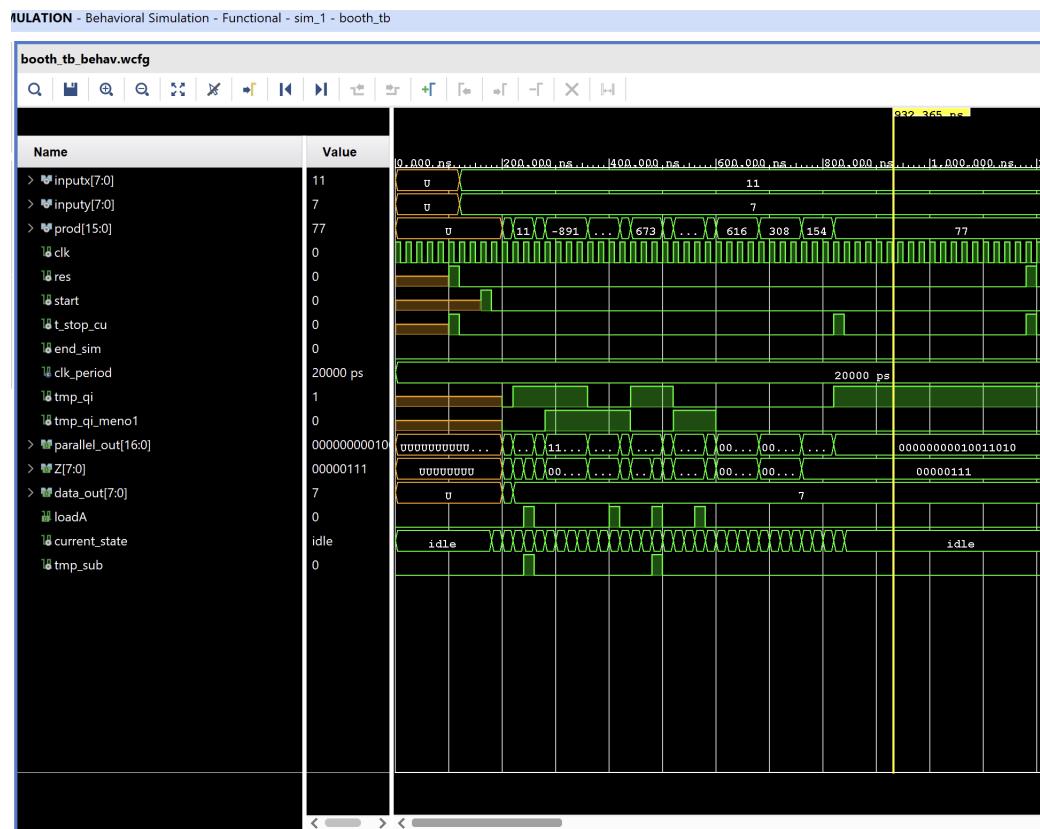
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  -- Entity adder_sub rappresenta un modulo che puo'
5  -- eseguire sia addizione che sottrazione
6  entity adder_sub is
7    port(
8      X, Y: in std_logic_vector(7 downto 0); -- 
9      -- Ingressi: due operandi a 8 bit.
10     cin: in std_logic; -- Ingresso: segnale di
11     -- controllo per selezionare addizione ('0') o
12     -- sottrazione ('1').
13     Z: out std_logic_vector(7 downto 0); -- Uscita:
14     -- risultato dell'addizione o sottrazione.
15     cout: out std_logic -- Uscita: riporto in
16     -- uscita per la sottrazione.
17   );
18 end adder_sub;

19
20 architecture Structural of adder_sub is
21
22 component ripple_carry is
23   port(
24     X, Y: in std_logic_vector(7 downto 0);

```

```
19      c_in: in std_logic;
20      c_out: out std_logic;
21      Z: out std_logic_vector(7 downto 0)
22  );
23
24 end component;
25
26 -- Segnale interno usato per mantenere la versione
27 -- complementata di Y.
28 signal complementoy: std_logic_vector(7 downto 0);
29
30 begin
31
32 -- Generazione del complemento di Y in base al
33 -- valore di cin.
34 -- Se cin = '0' (addizione), Y rimane invariato.
35 -- Se cin = '1' (sottrazione), Y viene
36 -- complementato.
37
38 complemento_y: FOR i IN 0 TO 7 GENERATE
39
40     complementoy(i) <= Y(i) xor cin; -- Operazione
41
42         xor per complementare Y se necessario.
43
44 END GENERATE;
45
46
47 -- Istanza del componente ripple_carry
48 RA: ripple_carry port map(X, complementoy, cin,
49
50     cout, Z);
51
52 end Structural;
```

## CHAPTER 7. MOLTIPLICATORE BOOTH



## 7.4 Implementazione su Board

Nell'implementazione sulla scheda del moltiplicatore, abbiamo introdotto un button debouncer per il segnale di start. Questo componente assicura che il segnale di avvio sia privo di rumore indesiderato, migliorando così l'affidabilità del sistema.

Per quanto riguarda l'acquisizione dell'input, abbiamo mappato i primi 8 switch all'input X e gli ultimi 8 switch all'input Y. Questa configurazione consente di inserire facilmente i valori di X e Y direttamente attraverso l'interfaccia utente della scheda.

Il prodotto P del moltiplicatore viene visualizzato sui LED della scheda. Questo fornisce un feedback visivo immediato del risultato dell'operazione di moltiplicazione.

Infine, abbiamo associato il segnale di start al pin P17 e il segnale di reset al pin M17. Questa configurazione consente un controllo intuitivo e diretto del moltiplicatore attraverso l'interfaccia utente della scheda.

```
1 ## Clock signal
2 set_property -dict { PACKAGE_PIN E3      IOSTANDARD
3                         LVCMOS33 } [get_ports { clock }];
4                         #IO_L12P_T1_MRCC_35 Sch=clk100mhz
5
6 create_clock -add -name sys_clk_pin -period 10.00
7                         -waveform {0 5} [get_ports {clock}];
8
9
10##Switches
11set_property -dict { PACKAGE_PIN J15      IOSTANDARD
```

```

        LVCMOS33 } [get_ports { X[0] }]; #IO_L24N_T3_RS0_15
        Sch=sw[0]
8 set_property -dict { PACKAGE_PIN L16      IOSTANDARD
        LVCMOS33 } [get_ports { X[1] }];
        #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
9 set_property -dict { PACKAGE_PIN M13      IOSTANDARD
        LVCMOS33 } [get_ports { X[2] }];
        #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
10 set_property -dict { PACKAGE_PIN R15     IOSTANDARD
        LVCMOS33 } [get_ports { X[3] }]; #IO_L13N_T2_MRCC_14
        Sch=sw[3]
11 set_property -dict { PACKAGE_PIN R17     IOSTANDARD
        LVCMOS33 } [get_ports { X[4] }]; #IO_L12N_T1_MRCC_14
        Sch=sw[4]
12 set_property -dict { PACKAGE_PIN T18     IOSTANDARD
        LVCMOS33 } [get_ports { X[5] }]; #IO_L7N_T1_D10_14
        Sch=sw[5]
13 set_property -dict { PACKAGE_PIN U18     IOSTANDARD
        LVCMOS33 } [get_ports { X[6] }];
        #IO_L17N_T2_A13_D29_14 Sch=sw[6]
14 set_property -dict { PACKAGE_PIN R13     IOSTANDARD
        LVCMOS33 } [get_ports { X[7] }]; #IO_L5N_T0_D07_14
        Sch=sw[7]
15 set_property -dict { PACKAGE_PIN T8      IOSTANDARD
        LVCMOS18 } [get_ports { Y[0] }]; #IO_L24N_T3_34
        Sch=sw[8]
16 set_property -dict { PACKAGE_PIN U8      IOSTANDARD
        LVCMOS18 } [get_ports { Y[1] }]; #IO_25_34 Sch=sw[9]

```

```

17 set_property -dict { PACKAGE_PIN R16      IOSTANDARD
                      LVCMOS33 } [get_ports { Y[2] }];
                      #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
18 set_property -dict { PACKAGE_PIN T13      IOSTANDARD
                      LVCMOS33 } [get_ports { Y[3] }];
                      #IO_L23P_T3_A03_D19_14 Sch=sw[11]
19 set_property -dict { PACKAGE_PIN H6       IOSTANDARD
                      LVCMOS33 } [get_ports { Y[4] }]; #IO_L24P_T3_35
                      Sch=sw[12]
20 set_property -dict { PACKAGE_PIN U12      IOSTANDARD
                      LVCMOS33 } [get_ports { Y[5] }];
                      #IO_L20P_T3_A08_D24_14 Sch=sw[13]
21 set_property -dict { PACKAGE_PIN U11      IOSTANDARD
                      LVCMOS33 } [get_ports { Y[6] }];
                      #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
22 set_property -dict { PACKAGE_PIN V10      IOSTANDARD
                      LVCMOS33 } [get_ports { Y[7] }]; #IO_L21P_T3_DQS_14
                      Sch=sw[15]
23
24 ## LEDs
25 set_property -dict { PACKAGE_PIN H17      IOSTANDARD
                      LVCMOS33 } [get_ports { P[0] }];
                      #IO_L18P_T2_A24_15
                      Sch=led[0]
26 set_property -dict { PACKAGE_PIN K15      IOSTANDARD
                      LVCMOS33 } [get_ports { P[1] }];
                      #IO_L24P_T3_RS1_15
                      Sch=led[1]
27 set_property -dict { PACKAGE_PIN J13      IOSTANDARD
                      LVCMOS33 } [get_ports { P[2] }];
                      #IO_L17N_T2_A25_15

```

```
Sch=led[2]
28 set_property -dict { PACKAGE_PIN N14      IOSTANDARD
                      LVCMOS33 } [get_ports { P[3] }]; #IO_L8P_T1_D11_14
Sch=led[3]
29 set_property -dict { PACKAGE_PIN R18      IOSTANDARD
                      LVCMOS33 } [get_ports { P[4] }]; #IO_L7P_T1_D09_14
Sch=led[4]
30 set_property -dict { PACKAGE_PIN V17      IOSTANDARD
                      LVCMOS33 } [get_ports { P[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
31 set_property -dict { PACKAGE_PIN U17      IOSTANDARD
                      LVCMOS33 } [get_ports { P[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
32 set_property -dict { PACKAGE_PIN U16      IOSTANDARD
                      LVCMOS33 } [get_ports { P[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
33 set_property -dict { PACKAGE_PIN V16      IOSTANDARD
                      LVCMOS33 } [get_ports { P[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
34 set_property -dict { PACKAGE_PIN T15      IOSTANDARD
                      LVCMOS33 } [get_ports { P[9] }]; #IO_L14N_T2_SRCC_14
Sch=led[9]
35 set_property -dict { PACKAGE_PIN U14      IOSTANDARD
                      LVCMOS33 } [get_ports { P[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
36 set_property -dict { PACKAGE_PIN T16      IOSTANDARD
                      LVCMOS33 } [get_ports { P[11] }];
#IO_L15N_T2_DQS_DOUT_CS0_B_14 Sch=led[11]
```

```

37 set_property -dict { PACKAGE_PIN V15      IOSTANDARD
                      LVCMOS33 } [get_ports { P[12] }];
                      #IO_L16P_T2_CSI_B_14 Sch=led[12]
38 set_property -dict { PACKAGE_PIN V14      IOSTANDARD
                      LVCMOS33 } [get_ports { P[13] }];
                      #IO_L22N_T3_A04_D20_14 Sch=led[13]
39 set_property -dict { PACKAGE_PIN V12      IOSTANDARD
                      LVCMOS33 } [get_ports { P[14] }];
                      #IO_L20N_T3_A07_D23_14 Sch=led[14]
40 set_property -dict { PACKAGE_PIN V11      IOSTANDARD
                      LVCMOS33 } [get_ports { P[15] }];
                      #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
41
42 ##Buttons
43 #set_property -dict { PACKAGE_PIN C12      IOSTANDARD
                      LVCMOS33 } [get_ports { CPU_RESETN }];
                      #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
44 #set_property -dict { PACKAGE_PIN N17      IOSTANDARD
                      LVCMOS33 } [get_ports { BTNC }];
                      #IO_L9P_T1_DQS_14
                      Sch=btnc
45 #set_property -dict { PACKAGE_PIN M18      IOSTANDARD
                      LVCMOS33 } [get_ports { BTNU }];
                      #IO_L4N_T0_D05_14
                      Sch=btnu
46 set_property -dict { PACKAGE_PIN P17      IOSTANDARD
                      LVCMOS33 } [get_ports { start }];
                      #IO_L12P_T1_MRCC_14 Sch=btnl
47 set_property -dict { PACKAGE_PIN M17      IOSTANDARD
                      LVCMOS33 } [get_ports { reset }];
                      #IO_L10N_T1_D15_14

```

Sch=bt(nr)

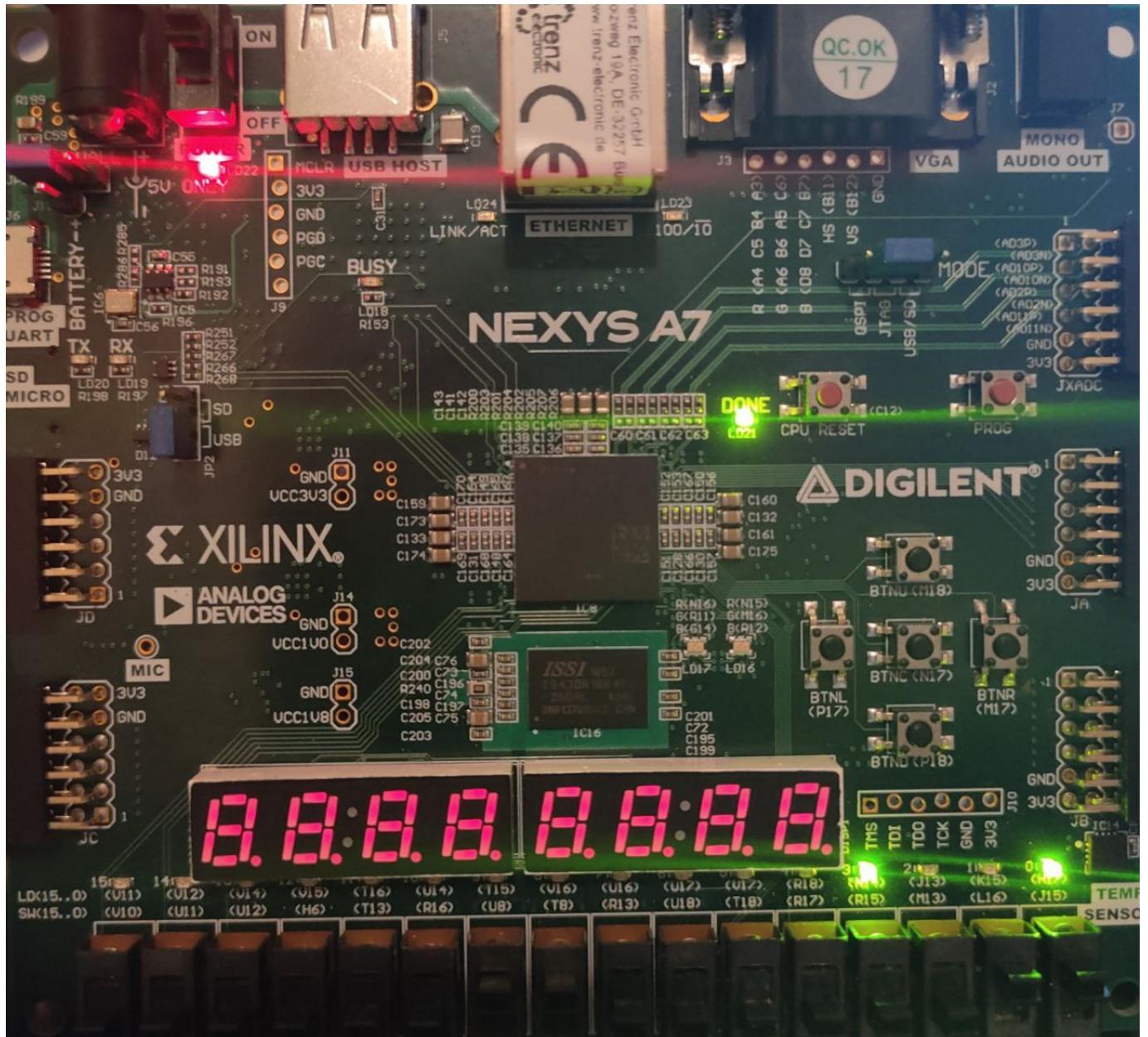


Figure 7.1: Esempio di moltiplicazione ( $3 \times 3$ )

# Chapter 8

## Handshaking

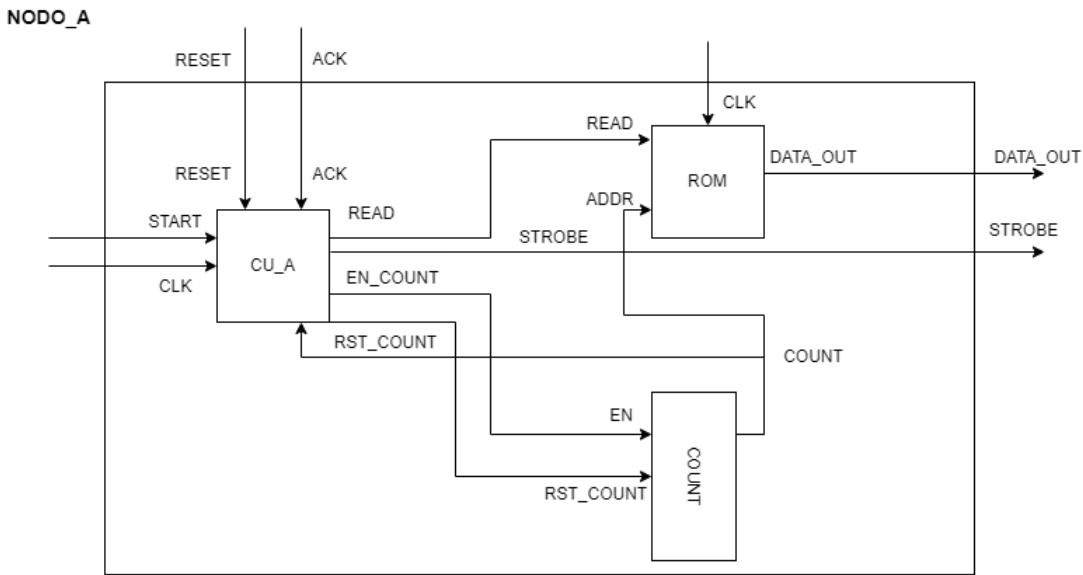
### 8.1 Traccia

- 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.

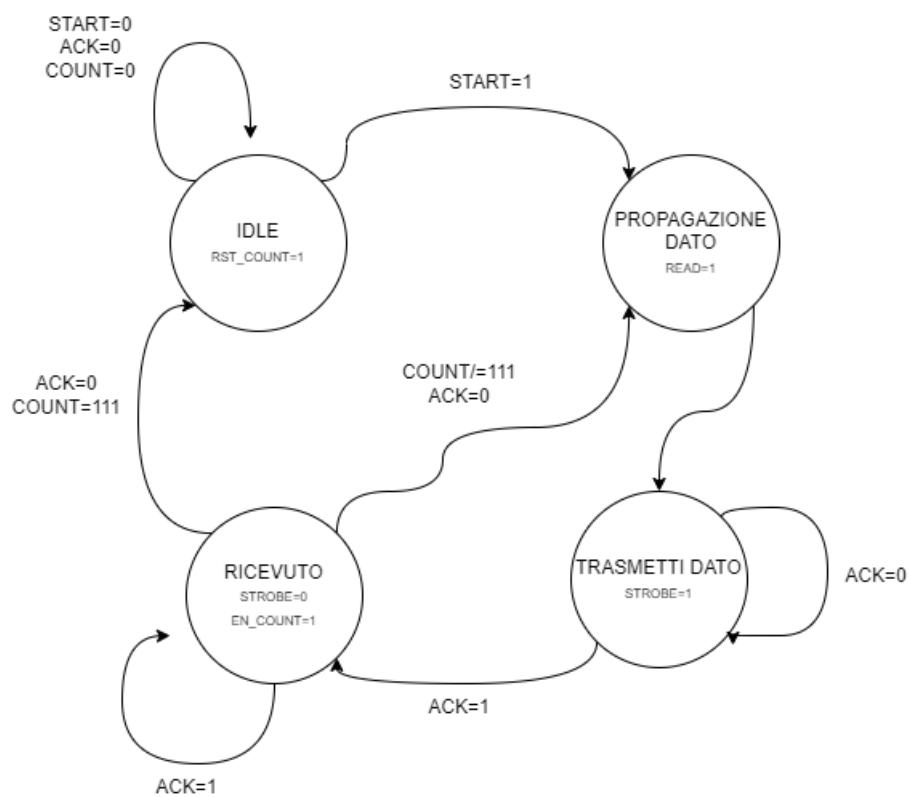
## 8.2 Progetto e Architettura

Si è progettato il design del sistema richiesto applicando un protocollo di handshaking a 2 vie per regolare la comunicazione tra i nodi A e B. Entrambi i nodi hanno una memoria interna in cui sono memorizzate N stringhe di M bit, denotate come  $X(i)$  per il nodo A e  $Y(i)$  per il nodo B, dove  $i$  varia da 0 a  $N-1$ . Il sistema si propone di trasmettere una stringa dal nodo A al nodo B che, una volta ricevuta, calcola la somma  $S(i)=X(i)+Y(i)$  e la memorizza in una specifica locazione della sua memoria interna. I due nodi del sistema complessivo operano a frequenze differenti (differente segnale di temporizzazione "CLK") Il nodo A è responsabile della trasmissione delle stringhe  $X(i)$  al nodo B. È stato progettato un componente VHDL che gestisce la trasmissione di una stringa alla volta, utilizzando un protocollo di handshaking. È stato implementato con un approccio structural, istanziando al suo interno i diversi componenti che costituiscono il sistema A: contatore modulo 8, memoria ROM da 8 locazioni da 1 byte ciascuna ed una control unit che descrive in modo behavioral l' automa a stati finiti per il nodo A.



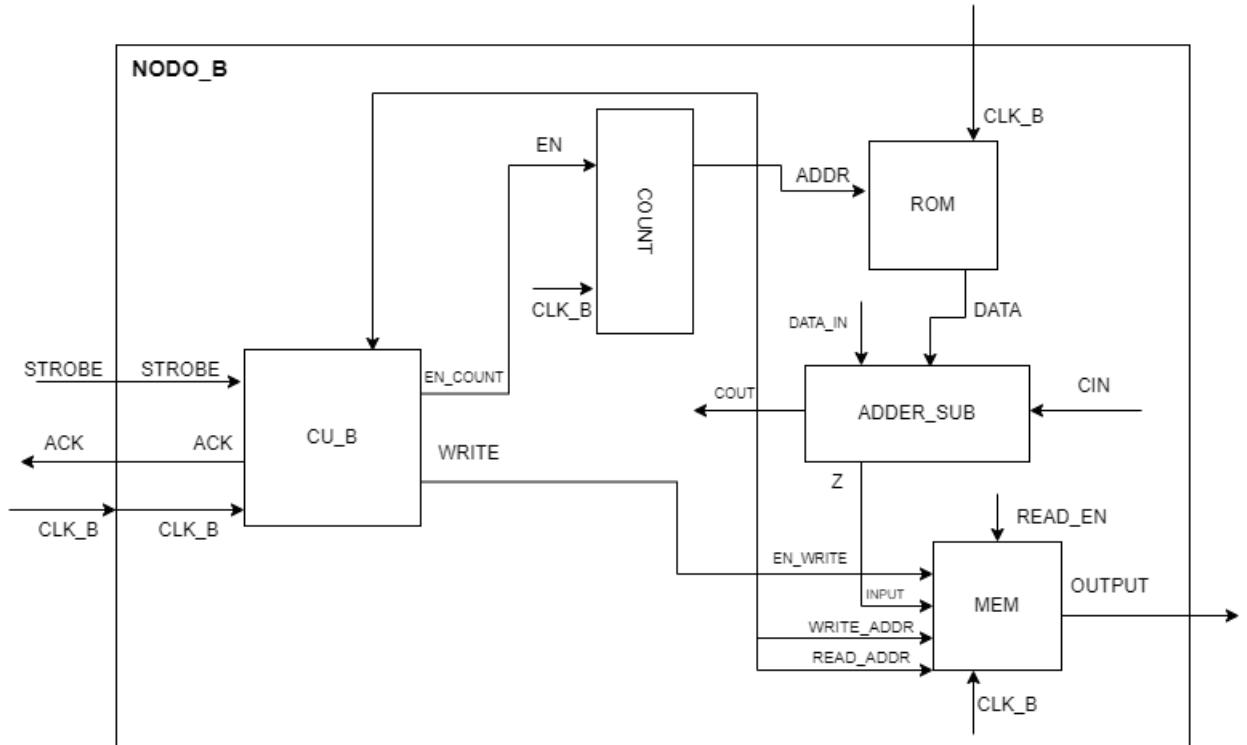
La control unit del nodo descrive il comportamento della macchina basandosi dell' automa a stati finiti. Inizialmente, fino a quando non arriva dall' esterno un segnale di START, il sistema permane nello stato "IDLE" ed il contatore resta resettato ( $RST\_COUNT=1$ ). Quando START diventa '1' il sistema passa allo stato "PROPAGAZIONE DATO" in cui il segnale di read viene posto pari a '1' ed avviene l' effettivo caricamento del dato sul bus dati in uscita. Lo stato successivo è "TRASMETTI DATO" in cui il segnale di "strobe" viene posto pari a '1' in modo da avvisare il sistema ricevente sull' avvenuto caricamento del dato in uscita sul bus dati. In tale stato si permane fintanto che il segnale di "ack" resta pari a '0'. Nel momento in cui ACK diventa alto il sistema entra nello stato "RICEVUTO" in cui i segnali di "strobe" e "en\_count" vengono posti rispettivamente pari a '0' e '1'. In questo stato il sistema permane fintanto che il segnale "ack" resta alto. Nel momento in cui il conteggio uscente dal conta-

tore non ha ancora raggiunto il valore massimo, lo stato successivo del sistema sarà di nuovo "PROPAGAZIONE DATO", viceversa nel caso in cui il conteggio raggiunge il valore massimo, il sistema torna allo stato "IDLE". In entrambi i casi il segnale "ack" viene posto pari a '0'.

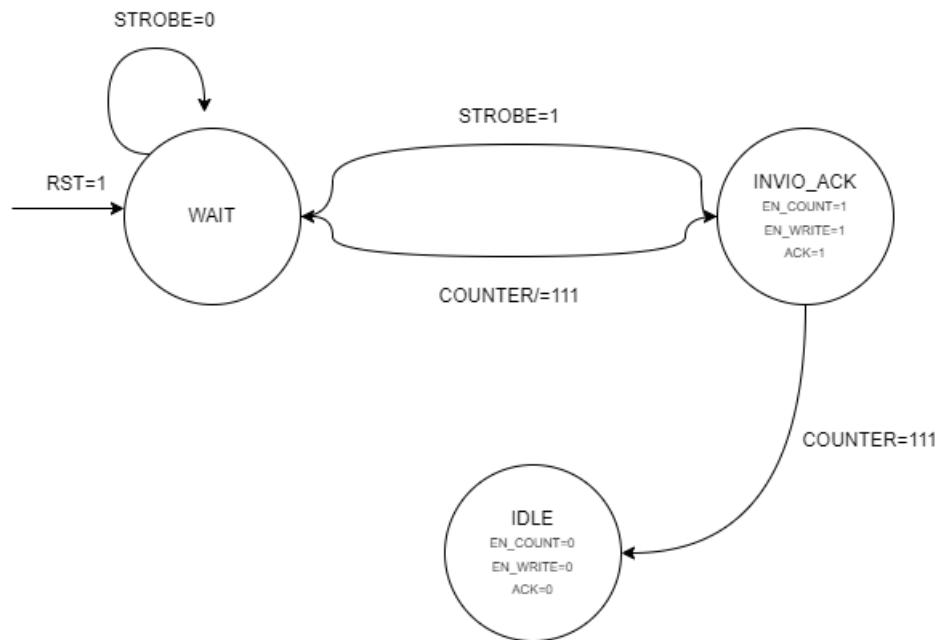
**CU\_A**

Il nodo B riceve le stringhe  $X(i)$  da A e calcola la somma  $S(i)=X(i)+Y(i)$ . Si è progettato un nodo B in cui sono stati istanziati i componenti: una ROM, una memoria r/w, un contatore modulo 8, una unità di controllo dedicata ed un sommatore/sottrattore RCA (implementato per costruzione di elementi elementari partendo da celle full-adder, consultabili in appendice). L' unità di controllo del nodo B descrive il

comportamento dell' automa a stati finiti.



Il sistema parte dallo stato "WAIT" in cui permane fintato che il segnale di "strobe" resta basso, cioè fino a quando il sistema A non comunica l' avvenuto caricamento di un dato sul bus condiviso. Nel momento in cui il segnale "strobe" diventa pari ad '1' il sistema transita nello stato "INVIO\_ACK" in cui i segnali "en\_counter", "en\_write", "ack" vengono posti pari ad '1'. A questo punto, nel mentre che il conteggio del contatore aumenta, fintanto che il conteggio non raggiunge il valore massimo, il sistema ritorna allo stato "WAIT". Nel momento in cui il valore massimo di conteggio è raggiunto il sistema transita nello stato "IDLE" in cui i segnali precedentemente posti pari ad '1' vengono azzerati.

**CU\_B**

## 8.3 Implementazione

### 8.3.1 Nodo A

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity nodo_A is
5     port (
6         clk: in std_logic;
7         reset: in std_logic;
8         start: in std_logic;
9         ack: in std_logic;
10        strobe: out std_logic;
11        data_out: out std_logic_vector(7 downto 0)

```

```
12      );
13 end nodo_A;
14
15 architecture Structural of nodo_A is
16
17 signal en_count_temp: std_logic;
18 signal count_temp: std_logic_vector(2 downto 0);
19 signal rst_count_temp: std_logic;
20 signal read_temp: std_logic;
21
22 begin
23
24     unita_controllo: entity work.CU_A port map(
25         clk => clk,
26         reset => reset,
27         start => start,
28         ack => ack,
29         en_count => en_count_temp,
30         rst_count => rst_count_temp,
31         count => count_temp,
32         strobe => strobe,
33         read => read_temp
34     );
35
36     contatore: entity work.counter_mod8 port map(
37         clk => clk,
38         reset => rst_count_temp,
39         enable => en_count_temp,
```

```

40      count => count_temp
41  );
42
43 ROM: entity work.rom port map(
44   clk => clk,
45   read => read_temp,
46   addr => count_temp,
47   data_out => data_out
48 );
49
50 end Structural;

```

### 8.3.2 CU A

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CU_A is
5   port (
6     clk: in std_logic;
7     reset: in std_logic;
8     start: in std_logic;
9     ack: in std_logic;
10    count: in std_logic_vector(2 downto 0);
11    en_count: out std_logic;
12    rst_count: out std_logic;
13    strobe: out std_logic;
14    read: out std_logic

```

```
15      );
16 end CU_A;
17
18 architecture Behavioral of CU_A is
19
20 type stato is (idle, preparazione_dato, trasmetti_dato,
21   ricevuto);
22 signal curr_state: stato := idle;
23 signal next_state: stato := idle;
24
25 begin
26   stato_uscita: process(curr_state, start, ack)
27   begin
28
29     rst_count <= '0';
30     read <= '0';
31     strobe <= '0';
32     en_count <= '0';
33
34     case curr_state is
35       when idle =>
36         if (start = '1') then
37           rst_count <= '1';
38           next_state <= preparazione_dato;
39         else
40           next_state <= idle;
41         end if;
```

```

42      when preparazione_dato =>
43          read <= '1';
44          next_state <= trasmetti_dato;
45      when trasmetti_dato =>
46          strobe <= '1';
47          if (ack = '1') then
48              next_state <= ricevuto;
49          else
50              next_state <= trasmetti_dato;
51      end if;
52      when ricevuto =>
53          if (ack = '0') then
54              en_count <= '1';
55              if(count = "111") then
56                  next_state <= idle;
57              else
58                  next_state <= preparazione_dato;
59              end if;
60          else
61              next_state <= ricevuto;
62          end if;
63      end case;
64
65  end process;
66
67 mem: process(clk)
68 begin
69     if rising_edge(clk) then

```

```
70      if reset = '1' then
71          curr_state <= idle;
72      else
73          curr_state <= next_state;
74      end if;
75  end if;
76 end process;
77
78
79 end Behavioral;
```

### 8.3.3 Nodo B

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity nodo_B is
5     port(
6         clk: in std_logic;
7         reset: in std_logic;
8         strobe: in std_logic;
9         data_in: in std_logic_vector(7 downto 0);
10        ack: out std_logic
11    );
12 end nodo_B;
13
14 architecture Structural of nodo_B is
15
```

```
16 signal en_count_temp: std_logic;
17 signal en_write_temp: std_logic;
18 signal count_temp: std_logic_vector(2 downto 0);
19 signal data_Y: std_logic_vector(7 downto 0);
20 signal res_Z: std_logic_vector(7 downto 0);
21 signal cout_temp: std_logic;
22 signal input_temp: std_logic_vector(8 downto 0);
23 signal out_temp: std_logic_vector(8 downto 0);
24
25 begin
26
27     unita_controllo: entity work.CU_B port map(
28         clk => clk,
29         reset => reset,
30         strobe => strobe,
31         count => count_temp,
32         en_count => en_count_temp,
33         en_write => en_write_temp,
34         ack => ack
35     );
36
37     ROM: entity work.rom port map(
38         clk => clk,
39         read => '1',
40         addr => count_temp,
41         data_out => data_Y
42     );
43
```

```
44     contatore: entity work.counter_mod8 port map(
45         clk => clk,
46         reset => reset,
47         enable => en_count_temp,
48         count => count_temp
49     );
50
51     adder: entity work.adder_sub port map(
52         X => data_in,
53         Y => data_Y,
54         cin => '0',
55         Z => res_Z,
56         cout => cout_temp
57     );
58
59     MEM: entity work.mem port map(
60         clk => clk,
61         reset => reset,
62         input => cout_temp & res_Z,
63         write_addr => count_temp,
64         read_addr => count_temp,
65         write_en => en_write_temp,
66         output => out_temp
67     );
68
69
70
71
```

72      **end** Structural;

### 8.3.4 CU B

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity CU_B is
5      port (
6          clk: in std_logic;
7          reset: in std_logic;
8          strobe: in std_logic;
9          count: in std_logic_vector(2 downto 0);
10         en_count: out std_logic;
11         en_write: out std_logic;
12         ack: out std_logic
13     );
14 end CU_B;
15
16 architecture Behavioral of CU_B is
17
18 type stato is (idle, waiting, invio_ack);
19 signal curr_state: stato := waiting;
20 signal next_state: stato := waiting;
21
22 begin
23
24     stato_uscita: process(curr_state, strobe)
```

```
25      begin
26
27          en_count <= '0';
28          en_write <= '0';
29          ack <= '0';
30
31      case curr_state is
32
33          when idle =>
34              next_state <= idle;
35
36          when waiting =>
37              if strobe='1' then
38                  next_state <= invio_ack;
39
40          else
41              next_state <= waiting;
42
43          end if;
44
45          when invio_ack =>
46              en_write <= '1';
47              en_count <= '1';
48              ack <= '1';
49
50              if(count = "111") then
51                  next_state <= idle;
52
53          else
54              next_state <= waiting;
55
56          end if;
57
58      end case;
59
60
61      end process;
```

```
53  
54     mem: process(clk)  
55     begin  
56         if rising_edge(clk) then  
57             if reset='1' then  
58                 curr_state <= waiting;  
59             else  
60                 curr_state <= next_state;  
61             end if;  
62         end if;  
63     end process;  
64  
65 end Behavioral;
```

## 8.4 Simulazione

La simulazione della comunicazione tra due nodi è stata condotta fornendo a ciascuno dei nodi comunicanti clock di diversa frequenza, al fine di dimostrare la corretta operatività del sistema. In particolare, al nodo A è stato assegnato un clock con un periodo di 10 ns, mentre al nodo B è stato attribuito un clock con un periodo di 40 ns. Questa configurazione ha permesso di simulare una situazione realistica in cui i due nodi operano a velocità diverse.

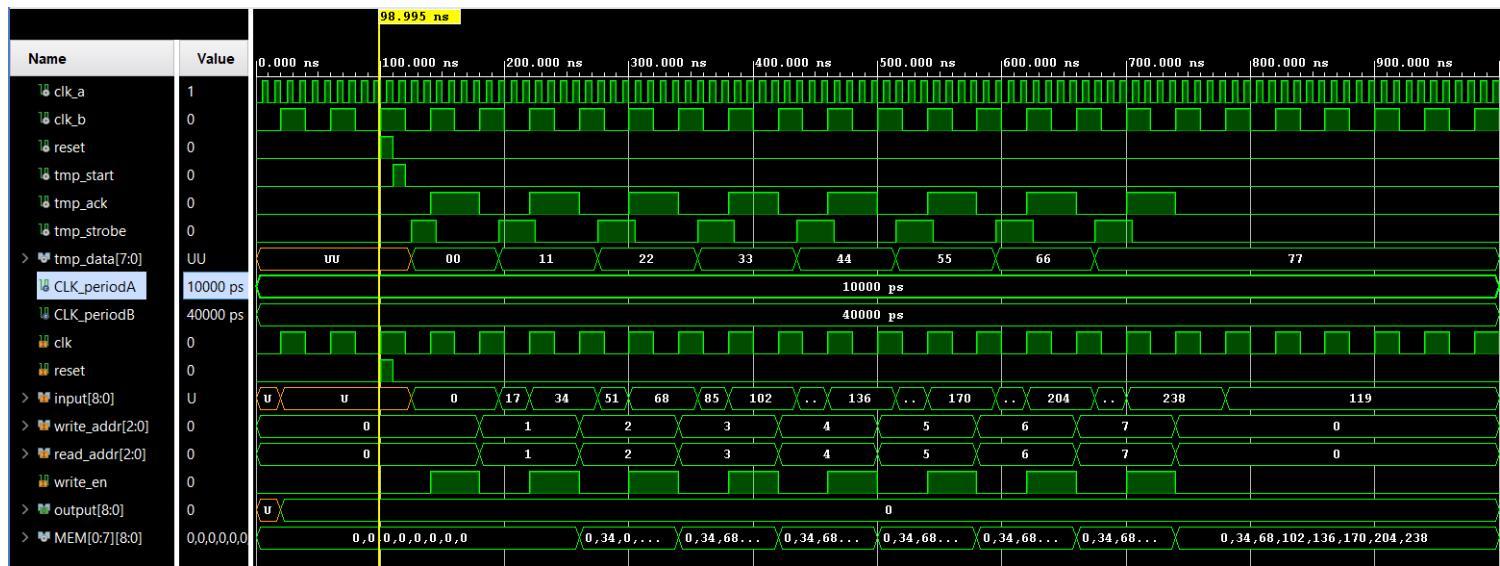


Figure 8.1: Simulazione con clock differenti

# Chapter 9

## Processore MIC-1

### 9.1 Traccia

A partire dall’implementazione fornita del processore operante secondo il modello IJVM,

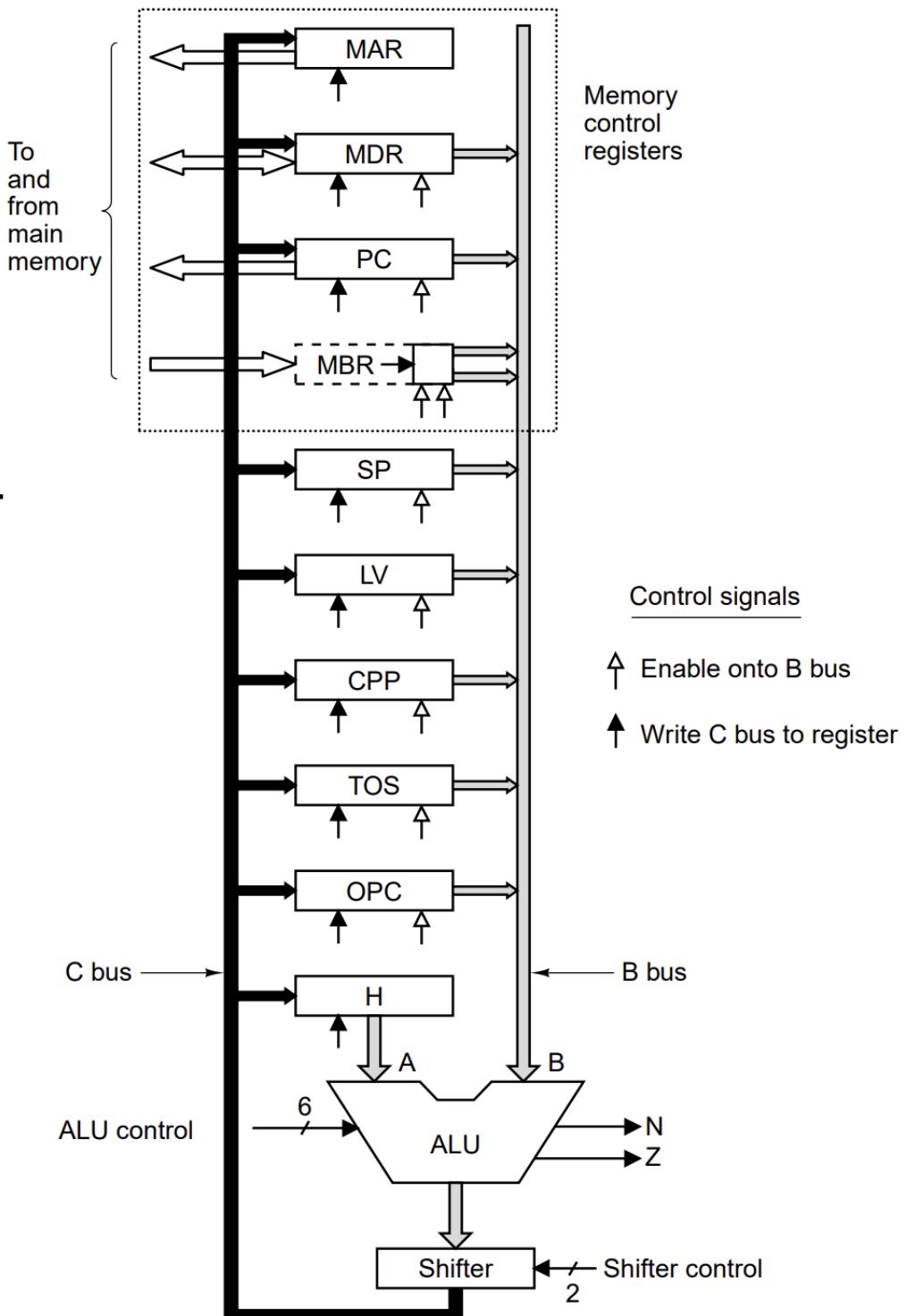
- si proceda all’analisi dell’architettura mediante simulazione e si approfondisca lo studio del suo funzionamento per due istruzioni a scelta,
- si modifichi un codice operativo a scelta, documentando tutte le modifiche effettuate

### 9.2 Architettura del processore

Il processore MIC-1, è un esempio semplice ma completo di un processore che affronta tutte le sfide e le complessità della progettazione di

sistemi avanzati. Questo processore implementa un sottoinsieme delle istruzioni della Java Virtual Machine, rendendolo un componente fondamentale nell’ambito dell’informatica. Una caratteristica distintiva del MIC-1 è la sua architettura a stack. A differenza di altri processori che utilizzano registri generali, il MIC-1 preleva gli operandi esclusivamente da uno stack. Questo approccio semplifica notevolmente l’insieme di istruzioni del processore, poiché supporta solo operazioni con operandi impliciti. Tuttavia, questo comporta anche una maggiore inefficienza dovuta ai continui accessi in memoria necessari per recuperare gli operandi. Il MIC-1 è dotato di una serie di registri interni, ognuno con un compito specifico. Questi registri, visibili solo al programmatore, sono in realtà solo una piccola parte di quelli presenti nella microarchitettura del processore. Ogni istruzione assembler è un punto di ingresso a una microsequenza che inizia con una microistruzione prelevata dal control store della microarchitettura e prosegue fino a quando la funzione dell’istruzione non è stata completamente eseguita. L’implementazione del processore MIC-1 utilizzata nell’esercizio prevede un’unità di controllo realizzata in logica micropogrammata. Ogni istruzione IJVM è implementata come una sequenza di microistruzioni, o microprocedura. Un insieme di microistruzioni forma il micropogramma, che è tipicamente memorizzato in una micro-ROM interna al processore.

### 9.2.1 Datapath



Il datapath del processore MIC-1, che consente l'implementazione delle istruzioni IJVM, è un sistema complesso e ben organizzato. È costituito da una serie di componenti chiave, tra cui l'Unità di Elaborazione Aritmetica (ALU), una serie di registri a 32 bit accessibili solo a livello microarchitetturale, e due bus principali: il BUS B per la lettura dai registri e il BUS C per la scrittura sui registri. L'ALU, il cuore del datapath, ha 6 linee di controllo. Queste linee di controllo, identificate come F0, F1, ENA, ENB, INVA e INC, permettono una vasta gamma di operazioni. Le linee F0 e F1 determinano l'operazione da eseguire, mentre ENA e ENB abilitano gli ingressi A e B. INVA inverte l'ingresso A, e INC incrementa l'ingresso di 1. Questa configurazione offre una flessibilità significativa, permettendo all'ALU di eseguire una vasta gamma di operazioni aritmetiche e logiche. L'ALU ha anche due segnali di uscita, N e Z, che indicano rispettivamente se il risultato di un'operazione è negativo o nullo. Quando l'ALU deve eseguire operazioni su due operandi, il primo operando viene caricato da uno dei registri, messo sul bus B e poi copiato dal bus C nel registro H. Il secondo operando viene poi connesso direttamente al bus B. Inoltre, il datapath include uno shift register con due linee di controllo aggiuntive: SLL8 per lo shift logico a sinistra di un byte, e SRA1 per lo shift aritmetico a destra di un bit. Per quanto riguarda la comunicazione con la memoria nel processore MIC-1 è orchestrata attraverso l'uso di coppie di registri, creando due interfacce distinte verso la memoria.

Queste coppie sono MAR (Memory Address Register)-MDR(Memory Data Register) e PC(Program Counter)-MBR(Memory Byte Register).

La coppia MAR-MDR è dedicata alla gestione dei dati, mentre la coppia PC-MBR si occupa delle istruzioni. L’interfaccia MAR-MDR offre la possibilità di specificare l’indirizzo in memoria da cui leggere o scrivere, grazie al collegamento bidirezionale dell’MDR con la memoria. D’altra parte, PC e MBR sono utilizzati per le istruzioni, con l’MBR che può solo leggere dalla memoria, dato che le istruzioni possono essere solo lette. Oltre all’ALU e ai registri di memoria, il processore MIC-1 dispone di una serie di altri registri, ciascuno con un ruolo specifico. Il registro H, ad esempio, è noto come il registro “holding”. Questo registro ha il compito di mantenere il primo operando dell’ALU, fungendo da deposito temporaneo per i dati in attesa di essere elaborati. Gli altri registri, pur essendo funzionalmente equivalenti, sono distinti sulla base dell’uso specifico che se ne fa nel microprogramma.

Questi includono:

- SP, o Stack Pointer, che tiene traccia della posizione corrente all’interno dello stack di memoria del programma.
- LV, o Local Variables, un registro dedicato alla memorizzazione delle variabili locali all’interno di una funzione o di un blocco di codice.
- CPP, o Constant Pool Pointer, che punta alla posizione nella memoria dove sono memorizzate le costanti utilizzate nel pro-

gramma.

- TOS, o Top of Stack, che tiene traccia dell'elemento più recente aggiunto allo stack.
- OPC, un registro “scratch” o temporaneo, utilizzato per una varietà di scopi a seconda delle necessità del programma.

### 9.2.2 Control Unit

Il datapath del processore MIC-1 è governato da un insieme di 29 segnali distinti. Tuttavia, grazie all'esclusività mutua di alcuni di questi segnali, il numero effettivo può essere ridotto a 24. Questi segnali costituiscono l'ossatura delle control word. Le control word, tuttavia, non si limitano a questi 24 segnali. Per assicurare un controllo microprogrammato efficace, è indispensabile includere anche l'indirizzo della prossima microistruzione da eseguire. Di conseguenza, la IJVM introduce due segnali di controllo supplementari, portando il totale a 36 bit. Questi bit sono raggruppati in cinque categorie: Addr, JAM, ALU, C, Mem e B. Questi segnali di controllo permettono di gestire i salti nel ciclo delle istruzioni, controllare l'ALU, selezionare quali registri vengono scritti dal bus C, gestire le operazioni da e verso la memoria, e determinare quale registro viene letto dal bus B.

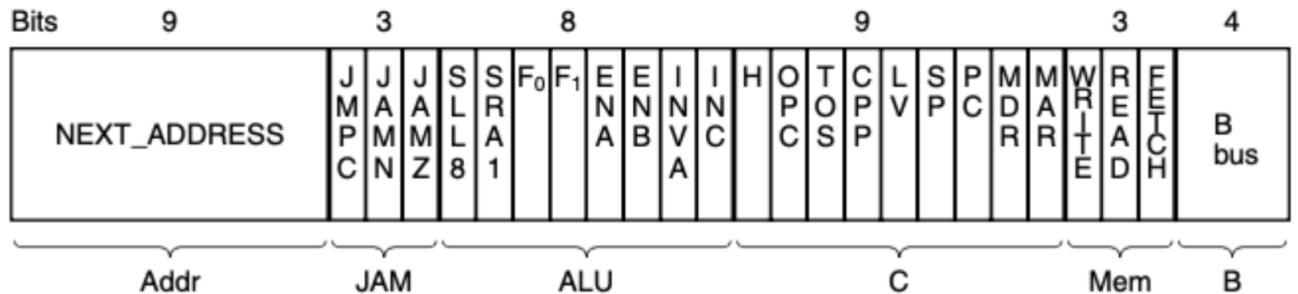


Figure 9.1: Struttura di una micro-istruzione MIC-1.

Le control word, o microistruzioni, sono memorizzate all'interno di una ROM 512x36 bit, nota come control store. La control store utilizza due componenti chiave: il Micro Program Counter (MPC), che indica quale microistruzione leggere ad ogni passo di controllo, e il Micro Instruction Register (MIR), che contiene la microistruzione letta all'indirizzo puntato dal MPC. L'indirizzo nella control store della prima microistruzione di una istruzione ISA corrisponde al valore binario dell'opcode dell'istruzione stessa. Il set di microistruzioni implementato dal MIC-1 è un sottoinsieme di quello della Java Virtual Machine, noto come IJVM, in quanto opera esclusivamente su interi. Più microistruzioni formano una sequenza di controllo. Per facilitare la programmazione del processore, viene utilizzato un linguaggio chiamato MAL (Micro Assembly Language). Un microassemblatore si occupa poi di tradurre dal linguaggio MAL al formato della microistruzione. Questo processo consente di gestire in modo efficiente e preciso il complesso sistema di controllo del processore MIC-1.

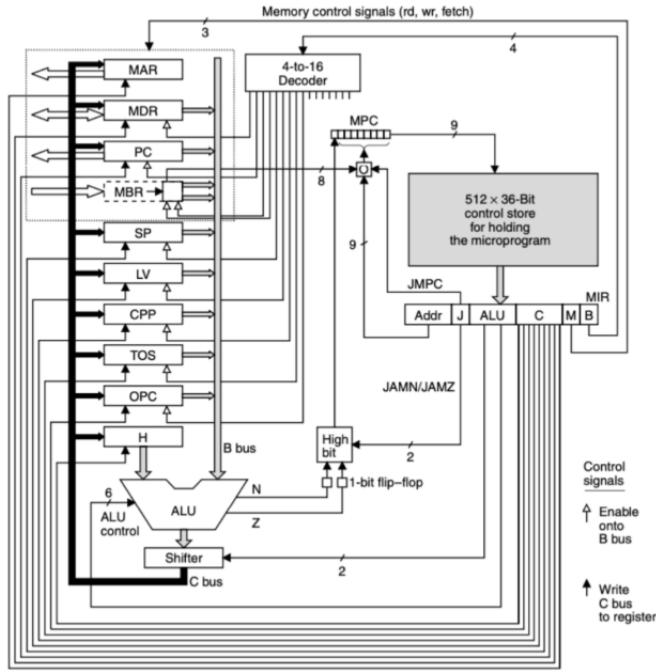


Figure 9.2: Architettura del processore MIC-1

## 9.3 Istruzioni Analizzate

### 9.3.1 BIPUSH

L'istruzione BIPUSH permette di caricare un byte, specificato alla chiamata del codice, sullo stack. Questa è un'istruzione fondamentale in un processore basato sul modello a stack come il MIC-1. Un esempio di chiamata di questa istruzione in un microprogramma potrebbe essere BIPUSH 0xA, che carica il valore 0xA sullo stack. L'istruzione BI-PUSH ha una lunghezza di due byte. Un byte è dedicato a mantenere il valore da caricare sullo stack, mentre l'altro byte specifica l'indirizzo, riferito al control store, del codice operativo BIPUSH. Quest'ultimo è

detto anche “entry point”, perché è il punto in cui il microprogramma entra nel control store per eseguire le microistruzioni della BIPUSH. Le microistruzioni, in linguaggio MAL, che descrivono il funzionamento del BIPUSH sono:

```

1 bipush = 0x10:
2
3     SP = MAR = SP + 1
4
5     PC = PC + 1; fetch
6
7     MDR = TOS = MBR; wr; goto main

```

L’indirizzo di questa istruzione nel control store è 0x10. Questo è l’entry point della BIPUSH nel microprogramma. Prima dell’esecuzione della BIPUSH, si esegue il microprogramma “main” che si occupa di fare il fetch della prossima istruzione. Quindi, al momento dell’esecuzione della prima microistruzione della BIPUSH, il valore da caricare sullo stack sarà già memorizzato all’interno del Memory Byte Register (MBR).

- $SP = MAR = SP + 1$ : Questa microistruzione incrementa il registro Stack Pointer (SP) e assegna il nuovo valore sia al Memory Address Register (MAR) che allo Stack Pointer stesso. Questo prepara la scrittura del byte in memoria e mantiene coerente il valore di SP nel processore.
- $PC = PC + 1$ ; fetch: Questa microistruzione incrementa il Program Counter (PC) e fa il fetch dell’istruzione successiva. Quando si ritorna a “main”, la prossima istruzione sarà già caricata nel Memory Data Register (MDR).

- MDR = TOS = MBR; wr; goto main: Questa microistruzione sposta il byte (già salvato in MBR) nel registro Top of Stack (TOS) per mantenerlo coerente con lo stato attuale dello stack e lo inserisce anche nel registro MDR. Poi richiama l'operazione di scrittura (wr) per scrivere il contenuto di MDR all'indirizzo specificato in MAR, completando così l'esecuzione dell'istruzione. Infine, esegue un salto a “main” per consentire il fetch della prossima istruzione e proseguire con l'esecuzione del microprogramma.

Nella RAM abbiamo analizzato un programma che esegue una BI-PUSH, come vediamo in figura.

Figure 9.3: In arancione troviamo l'opcode 0x10 e in azzurro il valore 0x0A da caricare sullo stack. Questi sono i 2 byte dell'istruzione BIPUSH.

Attraverso l’uso della simulazione VIVADO, abbiamo avuto l’opportunità di esaminare in dettaglio lo stato dei registri del datapath durante l’esecuzione dell’istruzione BIPUSH. I risultati ottenuti hanno confermato le nostre aspettative. In particolare, abbiamo osservato come il valore del registro Stack Pointer (SP) venisse incrementato correttamente.

mente durante l'esecuzione dell'istruzione. Inoltre, abbiamo notato che il valore del registro Top of Stack (TOS) corrispondeva esattamente al valore desiderato, ovvero 0x0A.

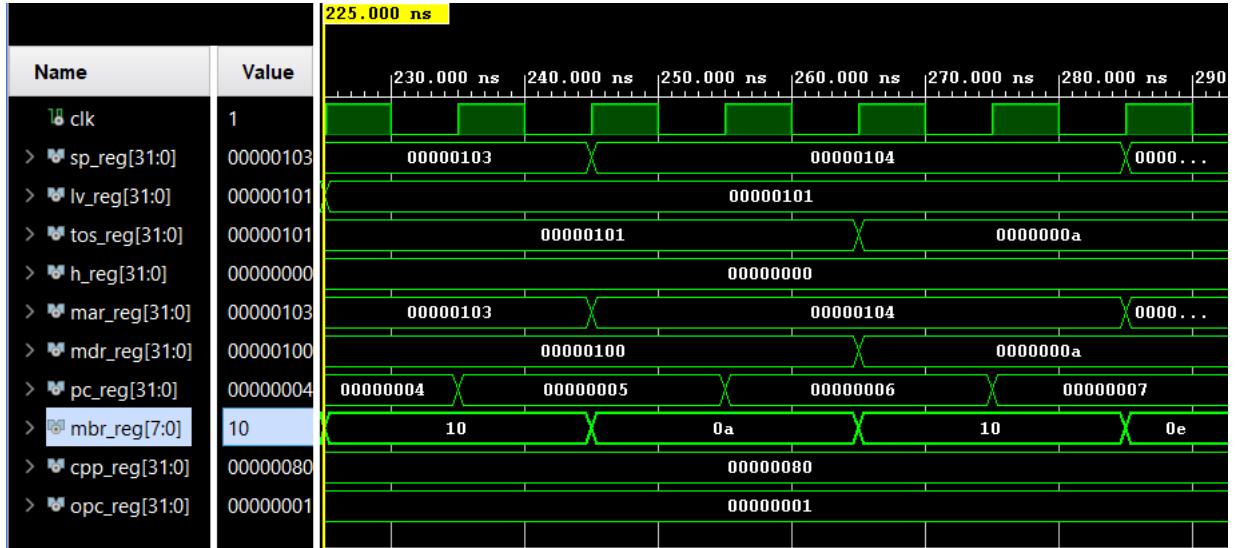


Figure 9.4: Stato dei registri del datapath durante l'esecuzione della BIPUSH.

### 9.3.2 IOR

La IOR è un'istruzione che esegue l'OR logico tra due valori sullo stack.

Per questo motivo questa istruzione ha significato solo quando ci sono almeno due valori caricati sullo stack. Questa istruzione è lunga 1 solo byte, che viene utilizzato per specificare l'entry point nella control store, che è 0xB6, per il codice operativo della IOR. Le microistruzioni, in linguaggio MAL, che descrivono il funzionamento della IOR sono:

```

1 ior = 0xB6:
2
3     MAR = SP = SP - 1; rd
4
5     H = TOS

```

4

```
MDR = TOS = MDR OR H; wr; goto main
```

Prima dell'esecuzione dell'istruzione IOR, si esegue il microprogramma “main” che si occupa di fare il fetch della prossima istruzione. Quindi, al momento dell'esecuzione della prima microistruzione dell'istruzione IOR, il primo operando sarà già memorizzato nel registro TOS.

- $\text{MAR} = \text{SP} = \text{SP} - 1$ ; rd: Questa microistruzione decrementa il registro Stack Pointer (SP) in modo che SP punti al secondo operando della OR. Questo valore viene anche assegnato al Memory Address Register (MAR) e viene richiamata l'operazione di lettura (rd) per leggere il secondo operando e averlo disponibile sul Memory Data Register (MDR).
- $H = \text{TOS}$ : Questa microistruzione posiziona il primo operando, il cui valore è memorizzato in TOS, nel registro di holding dell'ALU (H).
- $\text{MDR} = \text{TOS} = \text{MDR OR H}; \text{wr}; \text{goto main}$ : Questa microistruzione esegue effettivamente l'operazione di OR tra il registro H e il registro MDR. Il risultato viene salvato in TOS e in MDR. Poi viene richiamata l'operazione di scrittura (wr) per scrivere il risultato (che è in MDR) nello stack all'indirizzo specificato in MAR, che corrisponde a quello dell'SP, così da avere un risultato in cima allo stack.

Infine, esegue un salto a “main” per consentire il fetch della prossima

istruzione. Nella RAM abbiamo analizzato un programma che chiama due BIPUSH e successivamente una IOR, come vediamo in figura.

```
64 |      -- RAM content
65 |      signal mem : dp_ar_ram_type := (
66 |      --BEGIN_WORDS_ENTRY
67 |      128 => "00000000000000000000000000000000",
68 |      0 => "00000001000000000000000100000000",
69 |      1 => "000011000010000000101000010000",
70 |      2 => "00000000000000001010011110110110",  
    |      others => (others => '0')
72 |      --END_WORDS_ENTRY
```

Figure 9.5: In rosso c'è l'opcode della IOR (0xB6). La riga 1 invece contiene le due istruzioni di BIPUSH con i rispettivi valori 0xA e 0xE.

Attraverso l'uso della simulazione VIVADO, abbiamo avuto l'opportunità di esaminare in dettaglio lo stato dei registri del datapath durante l'esecuzione dell'istruzione IOR. Anche in questo caso i risultati ottenuti hanno confermato le nostre aspettative. In particolare, abbiamo osservato come il valore del registro Stack Pointer (SP) venisse decrementato correttamente durante l'esecuzione dell'istruzione. Inoltre, abbiamo notato che il valore del registro Top of Stack (TOS) e Memory Data Register (MDR) corrispondeva esattamente al risultato della OR logica tra i due operandi 0x0A e 0x0E, ovvero 0x0E.

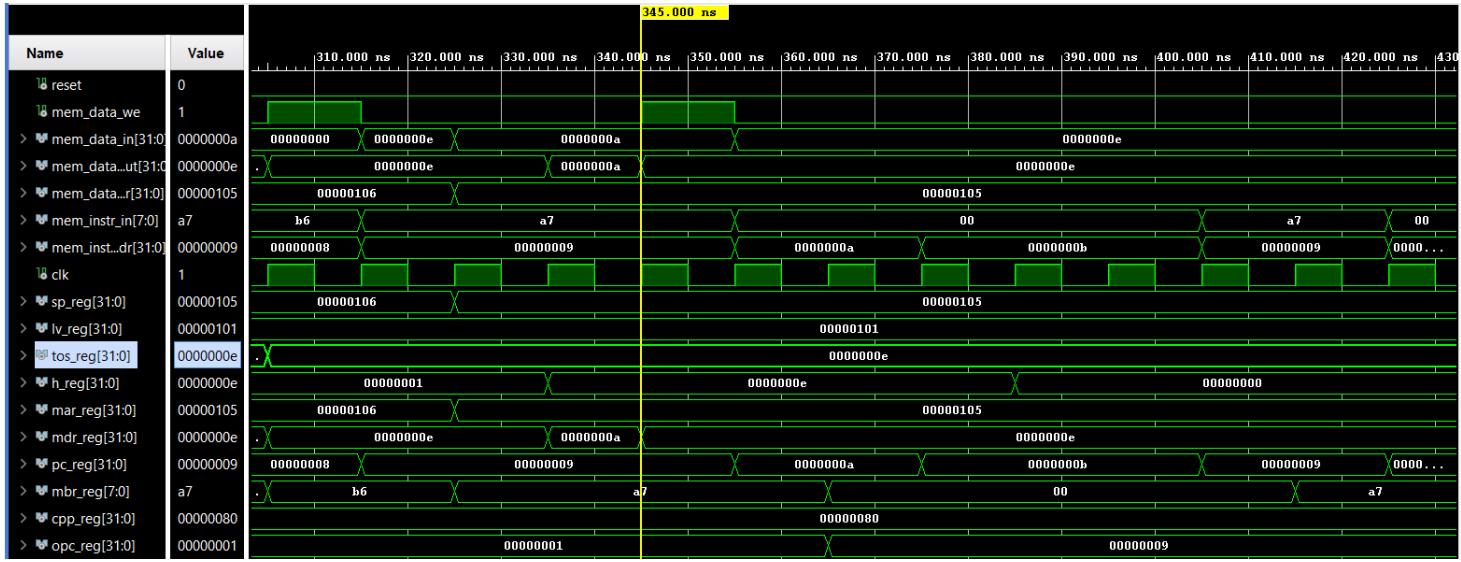


Figure 9.6: Stato dei registri del datapath durante l'esecuzione della IOR.

Possiamo osservare che l'esecuzione completa del codice operativo avviene in un intervallo di tempo che va da 325 ns a 345 ns. Durante questo periodo, si verificano tre cicli di clock. In ogni ciclo di clock, viene eseguita una delle tre microistruzioni che abbiamo analizzato precedentemente

### 9.3.3 Istruzione modificata: IADD3OP

L'istruzione IADD3OP è una versione modificata dell'istruzione IADD che permette di sommare 3 operandi presi dallo stack. La ricerca di spazio nella micro-rom è un passaggio fondamentale per l'aggiunta di una nuova istruzione. Prima di poter implementare l'istruzione IADD3OP, era necessario trovare dello spazio libero nella micro-rom che potesse contenere il nuovo microprogramma associato all'istruzione.

Questo spazio è stato individuato nel range di indirizzi che va dall'indirizzo 106 all'indirizzo 112.

Figure 9.7: Il microprogramma inizia alla locazione 0x6A (106 in decimale).

L'opcode utilizzato per la codifica dell'istruzione IADD3OP è 0x6A.

La descrizione dell'istruzione:

```
1 iadd3op = 0x6A:  
2     MAR = SP = SP - 1; rd  
3     H = TOS  
4     OPC = MDR  
5     MAR = SP = SP - 1; rd  
6     H = H + OPC  
7     MDR = TOS = MDR + H; wr; goto main
```

Il microprogramma deve prelevare i tre operandi dallo stack, sommarli e poi memorizzare il risultato nello stack. Ecco come funziona:

- Il valore del terzo operando (OP3) è già presente nel registro TOS.
  - Lo stack pointer viene decrementato e il nuovo valore dello SP viene caricato nel MAR. Viene, dunque, inizializzata l'operazione di lettura dell'operando due (OP2).

- Il valore di OP3 viene portato verso il registro di holding in ingresso all'ALU (H).
- OP2, a questo punto caricato in MDR, viene salvato nel registro temporaneo (OPC).
- Si effettua un ulteriore decremento dello SP e una conseguente lettura per caricare in MDR l'operando OP1.
- Si effettua la somma tra OP3 ed OP2, posizionati nei registri H ed OPC.
- Si procede poi ad effettuare la somma tra il risultato precedente ed OP1, ottenendo la somma complessiva.
- Questa somma complessiva viene caricata in TOS e nel MDR affinché possa, tramite il comando wr (write), essere inserita nello stack.

Infine, esegue un salto a “main” per consentire il fetch della prossima istruzione. Nella RAM abbiamo analizzato un programma che esegue tre BIPUSH e successivamente una IADD3OP, come vediamo in figura.

```

44      -- RAM content
45      signal mem : dp_ar_ram_type := (
46      --BEGIN_WORDS_ENTRY
47      128 => "000000000000000000000000000000000000000000000000000000000000000",
48      0 => "000000010000000000000000001000000000000000000000000000000000000",
49      1 => "0000001000010000000000000011100010000",  

50      2 => "1010011101101010000001100010000",  

51      3 => "000000000000000000000000000000000000000000000000000000000000000",
52      others => (others => '0')
53      --END_WORDS_ENTRY
54    );

```

Figure 9.8: Codice del microprogramma con tre bipush, in giallo, ed una iadd3op, in viola.

Grazie all'utilizzo della simulazione VIVADO, abbiamo avuto l'opportunità di analizzare minuziosamente lo stato dei registri del datapath durante l'esecuzione dell'istruzione IADD3OP. I risultati ottenuti hanno confermato le nostre previsioni. In particolare, abbiamo osservato che il valore del registro Stack Pointer (SP) veniva decrementato in modo appropriato durante l'esecuzione dell'istruzione. Abbiamo inoltre notato che gli operandi venivano caricati correttamente nei registri TOS, OPC e H. Abbiamo osservato che la somma parziale dei primi due operandi, conservata nel registro H, era 5, mentre il primo operando, di valore 7, era situato nel registro MDR. Infine, abbiamo constatato che il risultato finale, ovvero C, era correttamente posizionato nel registro TOS.

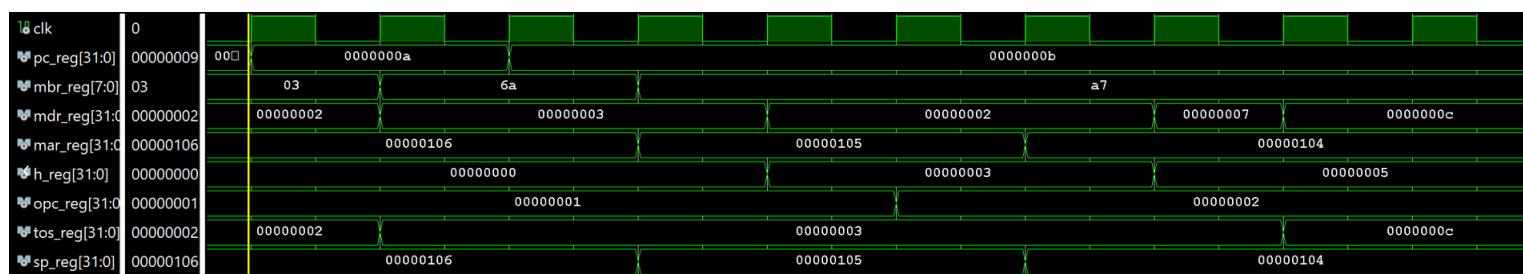


Figure 9.9: Stato dei registri del datapath durante l'esecuzione della IOR.

# Chapter 10

## Interfaccia Seriale

### 10.1 Traccia

- Partendo dall’implementazione fornita dalla Digilent di un dispositivo UART-RS232 (componente RS232RefComp.vhd), progettare, implementare e simulare in VHDL un sistema composto da 2 unità A e B che condividono lo stesso segnale di clock e comunicano tra loro mediante interfaccia seriale. Il sistema A contiene una ROM di 8 locazioni da 1 byte ciascuno, un contatore CONT\_A per scandire le locazioni della ROM e una UART\_A, mentre il sistema B contiene una memoria MEM di 8 locazioni da 1 byte ciascuno, un contatore CONT\_B per scandire le locazioni della MEM e una UART\_B. Quando un segnale WR viene asserito nell’unità A, viene prelevato un byte dalla ROM e inviato all’unità B, che dovrà riceverlo e salvarlo in MEM

## 10.2 Progetto e architettura

L’interfaccia seriale UART è un componente essenziale nei sistemi embedded e nelle comunicazioni seriale tra dispositivi. Questa tecnologia permette la trasmissione e la ricezione di dati in modo asincrono, consentendo la comunicazione tra dispositivi digitali attraverso la trasmissione di bit in serie su un singolo canale. L’UART opera inviando e ricevendo byte di dati in serie. Ogni trasmissione inizia con un bit di start, seguito dai dati stessi (generalmente 8 bit), opzionalmente un bit di parità e uno o più bit di stop. Questo formato di trama permette al ricevitore di sincronizzarsi con il flusso dei dati e di identificare l’inizio e la fine di ogni byte trasmesso. Ogni UART contiene un registro a scorrimento, che è il metodo fondamentale di conversione tra forme seriali e parallele. Esistono tre modalità di trasmissione:

- Simplex: la comunicazione è unidirezionale, per cui solo il trasmettitore comunica con il ricevitore;
- Half-Duplex: trasmettitore e ricevitore comunicano sullo stesso canale, ma solo uno alla volta può parlare;
- Full-Duplex: trasmettitore e ricevitore comunicano sullo stesso canale e possono parlare contemporaneamente.

La comunicazione viene regolata da parametri concordati precedentemente tra le due entità in trasmissione:

- BaudRate: velocità di trasmissione delle informazioni

- Lunghezza del frame: lunghezza della stringa di bit da trasmettere
- Endianness: ordine in cui i bit vengono trasmessi
- Sincronizzazione
- Controllo di errore

Ogni trasmissione è segnalata da un bit di start a livello logico basso e termina con uno o più bit di stop a livello logico alto, garantendo almeno due cambi di segnale tra i caratteri. Il ricevitore, sincronizzato con il flusso dei dati, campiona i bit trasmessi per identificare il loro inizio e fine, evitando errori di framing. I dispositivi devono concordare parametri come il baud rate, la lunghezza dei caratteri e il numero di bit di parità e di stop per garantire una comunicazione corretta. Il trasmettitore deposita i dati nel registro a scorrimento, generando bit di start, bit di dati, bit di parità (se necessario) e stop bit. Al termine della trasmissione, il trasmettitore segnala l'avvenuta trasmissione mentre il ricevitore indica la disponibilità dei dati ricevuti. L'UART è implementato attraverso due blocchi funzionali: uno per la ricezione e uno per la trasmissione dei dati seriali.

### 10.3 Trasmettitore TX - Architettura UART

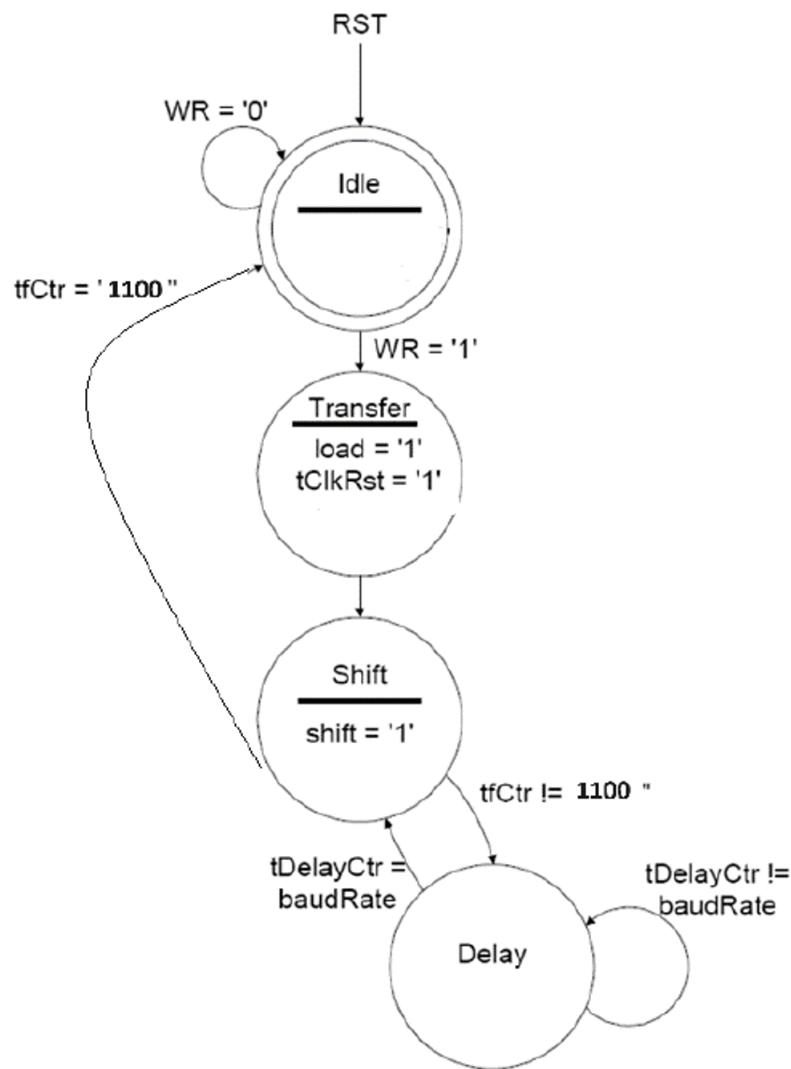
- Stato Idle: In questo stato la UART attende un segnale per iniziare la trasmissione. Rimanendo qui, la UART non compie

azioni finché non viene dato il via alla trasmissione di dati.

- La UART resta in Idle finché il segnale WR (Write Strobe) non viene attivato (cioè messo a '1'). Alternativamente, questo può corrispondere al segnale TBE (Transfer Bus Empty) che diventa basso, indicando che il bus di trasferimento è vuoto e pronto a ricevere dati per la trasmissione.
- Transizione a Transfer: Quando WR diventa alto:
  - La UART passa allo stato Transfer. In questo stato, viene inviato un segnale di load allo shift register, che significa che i dati da trasmettere vengono caricati nello shift register. Questi dati tipicamente consistono di un bit di start ('1'), il dato da trasmettere (DBIN), un eventuale bit di parità (parity\_bit), e un bit di stop ('0').
  - Inoltre, nello stato Transfer, il contatore fCtr, che tiene traccia del numero di bit trasmessi, viene resettato. Questo è necessario per poter contare i bit mentre vengono spostati fuori dallo shift register durante la trasmissione.
- Stato Shift: Dopo il caricamento dei dati:
  - La UART entra nello stato Shift, dove inizia effettivamente la trasmissione seriale dei bit. Viene attivato il segnale di

shift, permettendo la trasmissione di ciascun bit in uscita dallo shift register.

- Ogni bit è trasmesso in base alla velocità impostata dalla baud rate. Il passaggio attraverso lo stato Delay rappresenta questo intervallo di tempo, dove  $**tDelayCtr$  è il contatore che modula il ritardo tra la trasmissione di ogni bit per rispettare la baud rate impostata.\*\*
- Ritorno a Idle: Una volta che tutti i bit sono stati trasmessi:
  - \* Il contatore fCtr tiene traccia del numero di bit trasmessi. Quando fCtr raggiunge il valore di 12, che corrisponde alla lunghezza del frame di dati più un bit aggiuntivo (tipicamente usato per garantire che ci sia un intervallo tra la fine di un frame di dati e l'inizio del successivo), la macchina a stati ritorna nello stato Idle.
  - \* Questo significa che la trasmissione del frame di dati è completa, e la UART è pronta per iniziare la trasmissione di un nuovo byte di dati.

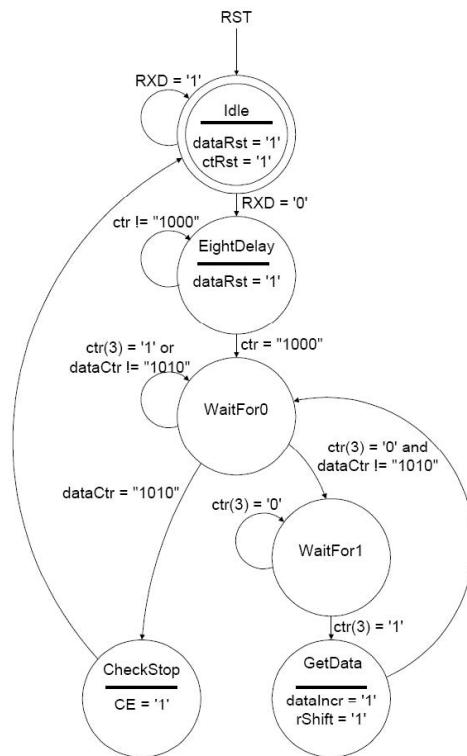


## 10.4 Ricevitore RX - Architettura UART

Quando è inattivo, il pin dei dati seriali in input (RXD) viene mantenuto alto

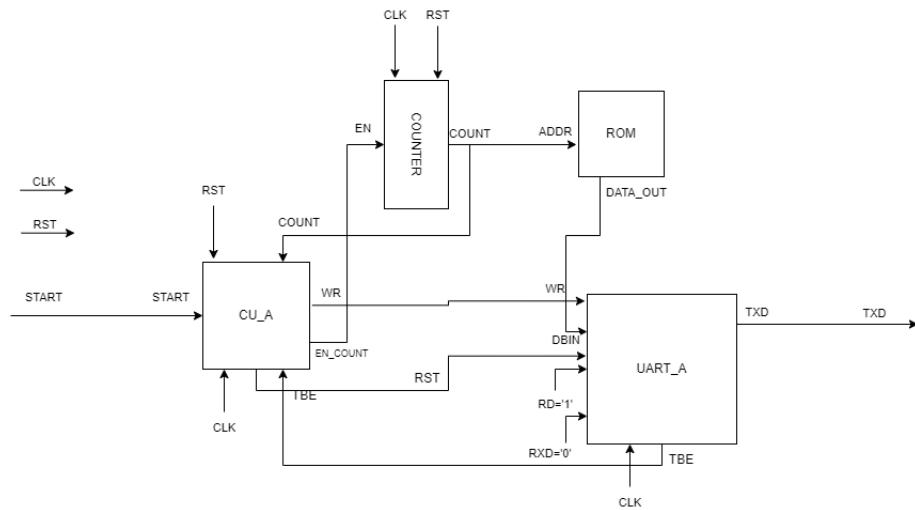
- Appena RXD diventa basso, si passa nello stato EightDelay, in cui si permane per 8 impulsi di conteggio in modo da posizionarsi a centro bit

- il contatore  $ctr$  avanza con una velocità 16 volte più alta rispetto a quella del trasmettitore
- Appena  $ctr=8$ , si passa nello stato `WaitFor0`, seguito dallo stato `WaitFor1`, che insieme assicurano che la macchina a stati venga ritardata esattamente per un tempo sufficiente a leggere il segnale RXD nel mezzo della sua trasmissione successiva (si attendono 8+8 impulsi di conteggio)
  - Questi due stati possono essere sostituiti da un semplice conteggio su  $ctr$  azzerando opportunamente il contatore quando serve
- Lo stato `GetData` incrementa il contatore `dataCtr` (dei bit di dato trasmessi) e fornisce il segnale di shift
  - NOTA: da `EightDelay` si può andare direttamente in `GetData` se viene inserita in tale stato l'attesa per i 16 impulsi di conteggio
- Appena  $dataCtr=10$  (8 bit di dati, 1 bit di parità e 1 bit di stop), viene attivato lo stato `CheckStop` che abilita il controller degli errori.



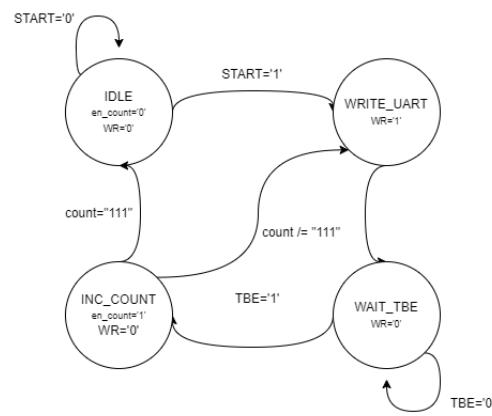
## 10.5 Implementazione progetto

### 10.5.1 Unità A

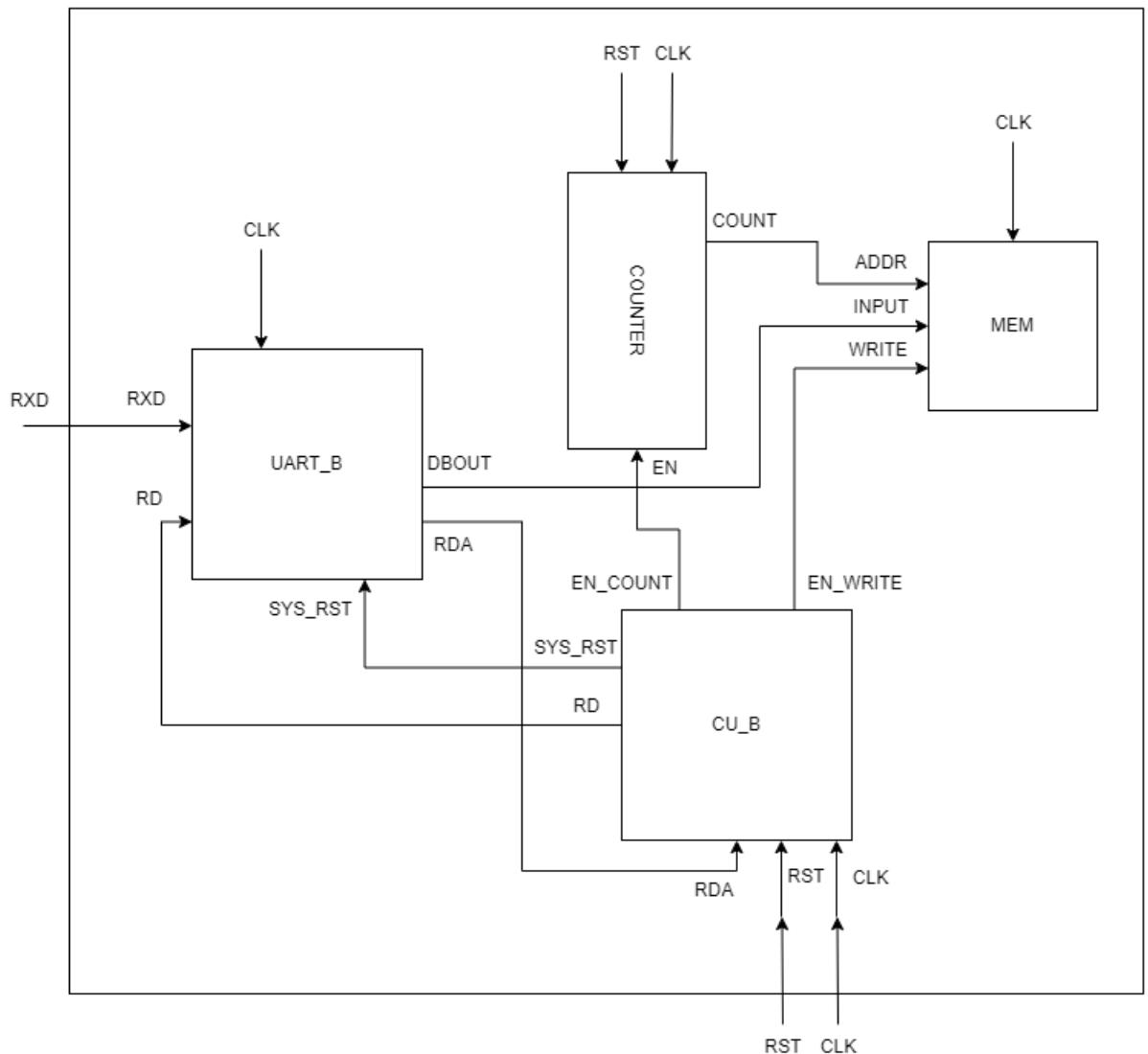


### 10.5.2 Automa a stati finiti - CU A

Inizialmente il sistema permane nello stato IDLE fino a quando non viene fornito dall'esterno un segnale di START che segnala l'inizio della trasmissione. All'avvenuta ricezione di START, il contatore, che ha il compito di scandire le locazioni della memoria ROM in cui sono memorizzati i byte da trasmettere, viene azzerato ed il segnale di WR (write) su UART viene abbassato. Si passa quindi allo stato WRITE\_UART in cui, ponendo WR='1', avviene l'effettiva scrittura del primo byte della memoria e scritto sul buffer in output del trasmettitore (DBIN del componente UART dell'unità A). A questo punto si attende che TBE diventi pari a '1' (stato WAIT\_TBE) per poi passare allo stato INC\_COUNT in cui viene incrementato il conteggio del contatore. A questo punto, se il valore di conteggio massimo non è ancora stato raggiunto, il sistema transita nello stato WRITE\_UART in cui verrà scritto su bus il valore contenuto nella locazione di memoria successiva. Viceversa in caso di valore massimo di conteggio raggiunto il sistema terminerà la propria esecuzione ritornando in IDLE e restando in attesa di un nuovo start.



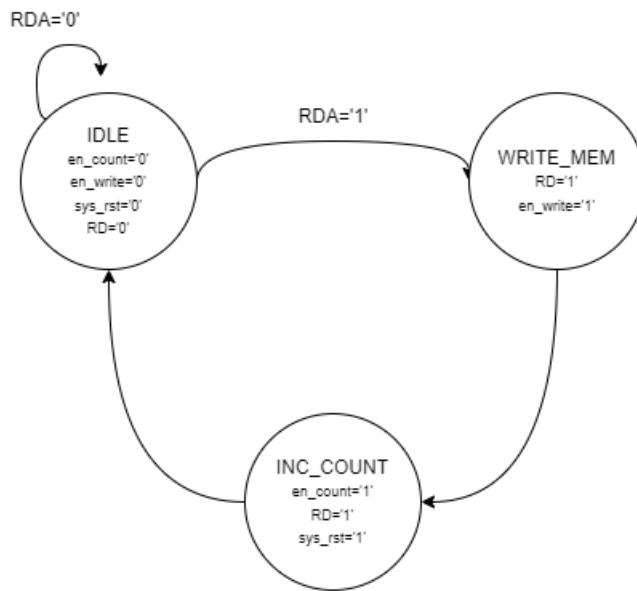
### 10.5.3 Unità B



### 10.5.4 Automa a stati finiti - CU B

L'unità ricevente inizia il proprio ciclo di esecuzione in uno stato di IDLE in cui permane fino a quando il segnale RDA non viene posto pari a '1'. Il segnale RDA (Receive Data Available) indica che sono disponibili dati ricevuti e pronti per essere letti dal ricevitore. Quando questo segnale è attivo, il ricevitore può leggere i dati dalla UART. In

questo stato il counter viene resettato cosí come l’interfaccia ricevente UART (insieme al segnale di RD read del bus in input). Se RDA=’1’ il sistema transita nello stato WRITE\_MEM in cui il segnale RD viene posto ad ’1’. Tale transizione segnala la volontá di leggere il valore in entrata dal bus (DBOUT) del componente UART. Si pone quindi il segnale en\_write=’1’ in modo da memorizzare il dato appena letto nella memoria mem interna. Si passa quindi al terzo stato in cui si incrementa il contatore e si resetta il componente UART del sistema ricevitore.



## 10.6 Implementazione

I componenti ROM (combinatoria), MEM e Counter\_Mod8 possono essere consultati in appendice

### 10.6.1 Comunicazione\_Seriale

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Comunicazione_Seriale is
5     port (
6         clk: in std_logic;
7         reset: in std_logic;
8         start: in std_logic
9     );
10 end Comunicazione_Seriale;
11
12 architecture Structural of Comunicazione_Seriale is
13
14     signal link_UARTS: std_logic;
15
16 begin
17
18     UA: entity work.Unita_A
19         port map(
20             clk => clk,
21             reset => reset,
22             start => start,
23             TXD => link_UARTS
24         );
25
26     UB: entity work.Unita_B
27         port map(
```

```

28      clk => clk,
29
30      reset => reset,
31
32      RXD => link_UARTS
33  );
34
35 end Structural;

```

## 10.6.2 CU A

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CU_A is
5
6     port (
7         start: in std_logic;
8         clk: in std_logic;
9         reset: in std_logic;
10        count: in std_logic_vector(2 downto 0);
11        TBE: in std_logic;
12
13        en_count: out std_logic;
14        WR: out std_logic
15    );
16
17 end CU_A;
18
19 architecture Behavioral of CU_A is
20
21 type state is (IDLE, WRITE_UART, WAIT_TBE, INC_COUNT);

```

```
20 signal stato_corrente: stato := IDLE;
21 signal stato_successivo: stato;
22
23 begin
24
25 stato_uscita: process(stato_corrente, start, TBE)
26 begin
27
28     en_count <= '0';
29     WR <= '0';
30
31     case stato_corrente is
32         when IDLE =>
33             en_count<='0';
34             WR<='0';
35             if (start = '1') then
36                 stato_successivo <= WRITE_UART;
37             else
38                 stato_successivo <= IDLE;
39             end if;
40
41         when WRITE_UART =>
42             WR <= '1';
43             stato_successivo <= WAIT_TBE;
44
45         when WAIT_TBE =>
46             WR<='0';
47             if (TBE = '0') then
```

```
48         stato_successivo <= WAIT_TBE;
49
50     else
51
52         stato_successivo <= INC_COUNT;
53
54     end if;
55
56
57     when INC_COUNT =>
58
59         en_count <= '1';
60
61         WR<='0';
62
63         if(count = "111") then
64
65             stato_successivo <= IDLE;
66
67         else
68
69             stato_successivo <= WRITE_UART;
70
71         end if;
72
73     end case;
74
75 end process;
```

```
64
65 mem: process(clk)
66 begin
67
68     if rising_edge(clk) then
69
70         if reset = '1' then
71
72             stato_corrente <= IDLE;
73
74         else
75
76             stato_corrente <= stato_successivo;
77
78         end if;
79
80     end if;
81
82 end process;
```

76      **end Behavioral;**

### 10.6.3 Unita A

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Unita_A is
5     port (
6         clk: in std_logic;
7         reset: in std_logic;
8         start: in std_logic;
9
10        TXD: out std_logic
11    );
12 end Unita_A;
13
14 architecture Structural of Unita_A is
15
16    signal en_count_temp: std_logic;
17    signal count_temp: std_logic_vector(2 downto 0);
18    signal WR_temp: std_logic;
19    signal rom_out_temp: std_logic_vector(7 downto 0);
20    signal TBE_temp: std_logic;
21
22 begin
23
24    CU: entity work.CU_A
```

```
25      port map(
26          reset => reset,
27          clk => clk,
28          start => start,
29          TBE => TBE_temp,
30          count => count_temp,
31          en_count => en_count_temp,
32          WR => WR_temp
33      );
34
35 cont: entity work.counter_mod8
36     port map(
37         clk => clk,
38         reset => reset,
39         enable => en_count_temp,
40         count => count_temp
41     );
42
43 ROM: entity work.rom
44     port map(
45         addr => count_temp,
46         data_out => rom_out_temp
47     );
48
49 UART_A: entity work.Rs232RefComp
50     port map(
51         RXD => '0', --non ci sono dati in arrivo
52         CLK => clk,
```

```

53      DBIN => rom_out_temp,
54      TBE => TBE_temp,
55      RD => '1',
56      WR => WR_temp,
57      RST => reset,
58      TXD => TXD
59  );
60
61 end Structural;

```

#### 10.6.4 CU B

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CU_B is
5   port(
6     clk: in std_logic;
7     reset: in std_logic;
8     RDA: in std_logic;
9
10    en_count: out std_logic;
11    en_write: out std_logic;
12    sys_RST: out std_logic;
13    RD: out std_logic
14  );
15 end CU_B;
16

```

```
17 architecture Behavioral of CU_B is
18
19 type stato is (IDLE, WRITE_MEM, INC_COUNT);
20 signal stato_corrente: stato := IDLE;
21 signal stato_successivo: stato;
22
23 begin
24
25 stato_uscita: process(stato_corrente, RDA)
26 begin
27
28     en_count <= '0';
29     en_write <= '0';
30     sys_RST<='0';
31
32     case stato_corrente is
33         when IDLE =>
34             en_count<='0';
35             en_write<='0';
36             sys_RST<='0';
37             RD<='0';
38             if (RDA = '1') then
39                 stato_successivo <= WRITE_MEM;
40             else
41                 stato_successivo <= IDLE;
42             end if;
43
44         when WRITE_MEM =>
```

```
45      RD <= '1';
46      en_write <= '1';
47      stato_successivo <= INC_COUNT;
48
49      when INC_COUNT =>
50          en_count <= '1';
51          RD <='1'; --
52          sys_rst<='1';
53          stato_successivo <= IDLE;
54
55      end case;
56
57 end process;
58
59 mem: process(clk)
60 begin
61     if rising_edge(clk) then
62         if reset = '1' then
63             stato_corrente <= IDLE;
64         else
65             stato_corrente <= stato_successivo;
66         end if;
67     end if;
68
69 end process;
70
71 end Behavioral;
```

### 10.6.5 Unita B

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Unita_B is
5     port (
6         clk: in std_logic;
7         reset: in std_logic;
8         RXD: in std_logic
9     );
10 end Unita_B;
11
12 architecture Structural of Unita_B is
13
14     signal RD_temp: std_logic;
15     signal rst_temp: std_logic;
16     signal RDA_temp: std_logic;
17     signal en_count_temp: std_logic;
18     signal count_temp: std_logic_vector(2 downto 0);
19     signal en_write_temp: std_logic;
20     signal data_mem_temp: std_logic_vector(7 downto 0);
21
22 begin
23
24     CU: entity work.CU_B
25         port map(
26             clk => clk,
27             reset => reset,
```

```

28          RDA => RDA_temp,
29
30          en_count => en_count_temp
31
32          en_write => en_write_temp
33
34          sys_RST => rst_temp,
35
36          RD => RD_temp
37
38      );
39
40
41      cont: entity work.counter_mod8
42
43          port map(
44
45              clk => clk,
46
47              reset => reset,
48
49              enable => en_count_temp,
50
51              count => count_temp
52
53      );
54
55
56      MEM: entity work.mem
57
58          port map(
59
60              clk => clk,
61
62              input => data_mem_temp,
63
64              write_addr => count_temp,
65
66              write_en => en_write_temp
67
68      );
69
70
71      UART_B: entity work.Rs232RefComp
72
73          port map(
74
75              RXD => RXD,
76
77              CLK => clk,
78
79              DBIN => (others => '0'),
80
81              TXD => TXD
82
83      );
84
85
86      end;

```

```

56      RD  => RD_temp,
57      WR  => '0',
58      RST  => rst_temp,
59      DBOUT => data_mem_temp,
60      RDA  => RDA_temp
61  );
62
63 end Structural;

```

## 10.7 Simulazione



# Chapter 11

## Switch Multistadio

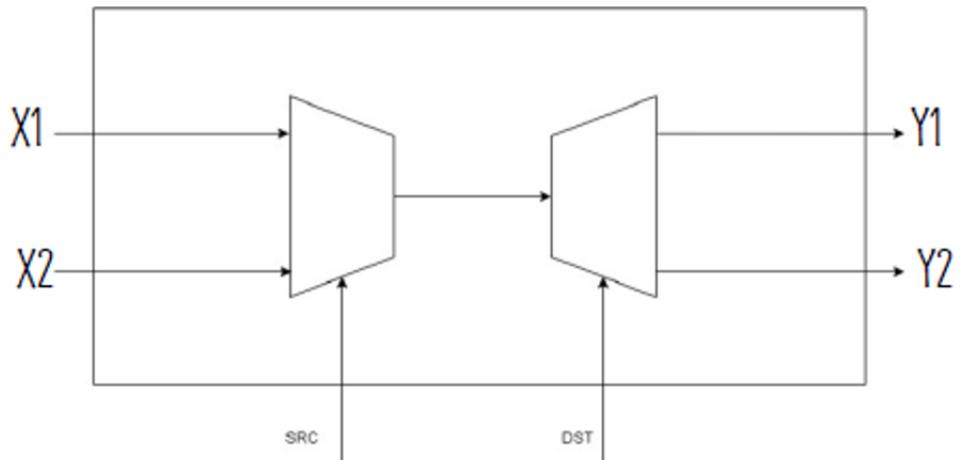
### 11.1 Traccia

- Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network. Lo switch deve consentire lo scambio di messaggi di 2 bit ciascuno da un nodo sorgente a un nodo destinazione in una rete con 4 nodi, implementando uno schema a priorità fissa fra i nodi (es. nodo 1 più prioritario, con priorità decrescenti fino al nodo 4).

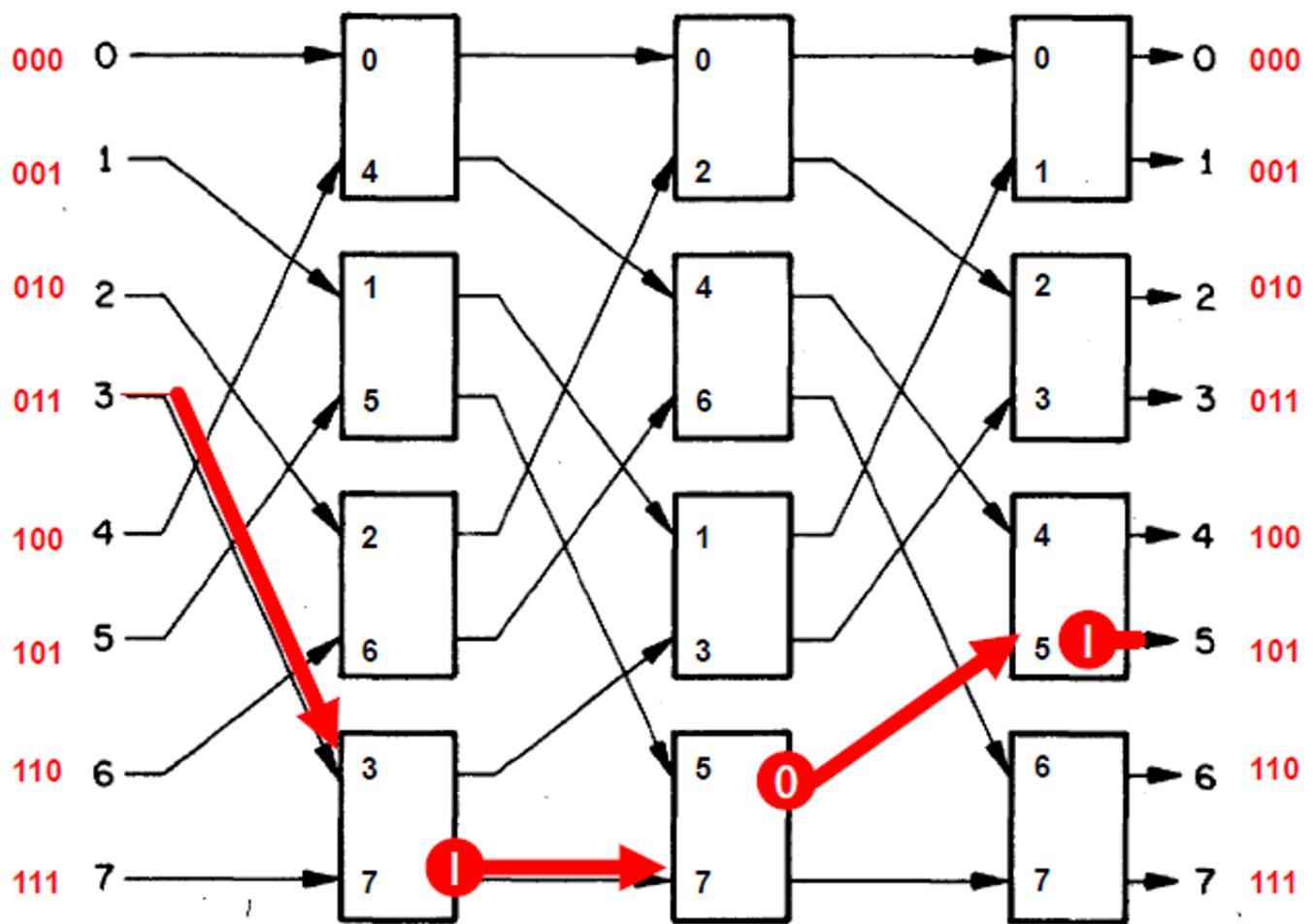
### 11.2 Cenni teorici

Uno switch multistadio in modello omega network è un esempio di rete di interconnessione con una topologia ben definita. Uno switch elementare è quello che interconnette due sorgenti X1 e X2 con due des-

tinazioni Y1 e Y2 utilizzando un mux2:1 e un demux. Utilizzando blocchi elementari come questo è possibile realizzare un'interconnessione fra tutte le coppie di nodi comunicanti adottando una specifica architettura di interconnessione.



La Omega Network é un'architettura di interconnessione (basata su switch) . Si propone di far comunicare tra loro N nodi tra loro, per cui per una matrice  $N \times N$  vi saranno  $\log_2 N$  stadi identici che sfruttano un'interconnessione fra i nodi basata sul perfect shuffling (algoritmo derivante dal mischiare le carte da gioco). Sfruttando questo algoritmo esiste un unico percorso fra una determinata coppia di nodi. Ciascuno switch usato in questa architettura é molto simile a quello elementare ma ha la capacità di assumere uno tra quattro stati possibili. L'instradamento dei messaggi attraverso la omega network viene stabilito sulla base dell'indirizzo di destinazione.

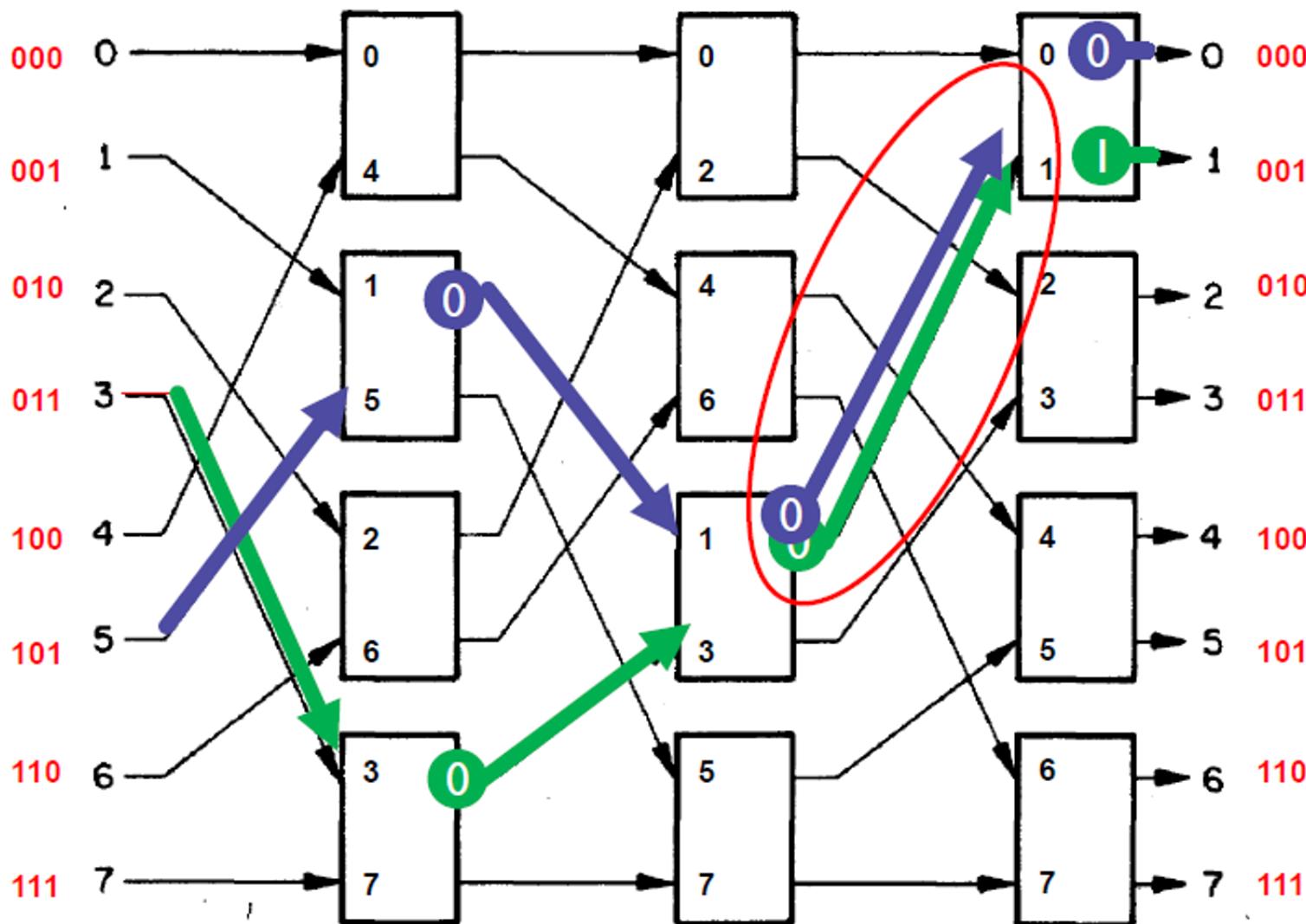


Si parte dal **bit più significativo** dell'indirizzo, che decide quale uscita prendere nello switch interessato. Ognuno dei bit è relativo ad uno stadio:

- Se il **Bit** è **0** → si attiva l'uscita superiore
- Se il **Bit** è **1** → si attiva l'uscita inferiore
  - Dunque stabilisce per ogni stadio quale percorso deve prendere il messaggio

Il conflitto ce l'ho quando due comunicazione simultanee vanno sullo

stesso oggetto quindi sullo stesso filo → però se ciò non si verifica posso avere più comunicazioni simultanee. Nel caso in cui vi siano più comunicazioni simultanee potrebbe verificarsi una collisione:

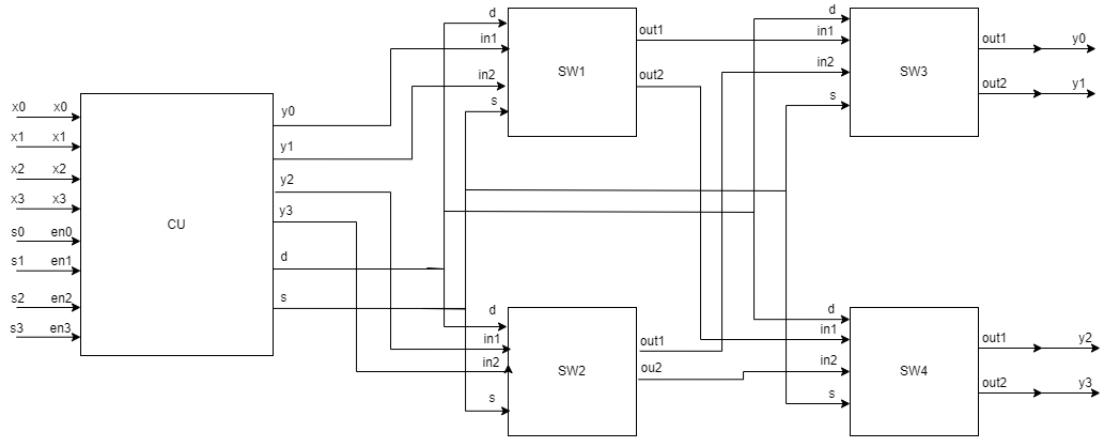


In questo caso i due percorsi provocano un conflitto perché hanno un percorso in comune all'uscita del secondo stadio. Nel caso in cui venga utilizzata una tecnica di instradamento dei pacchetti di tipo "Wormhole", sono 4 le tecniche che possono essere applicate per gestire i conflitti:

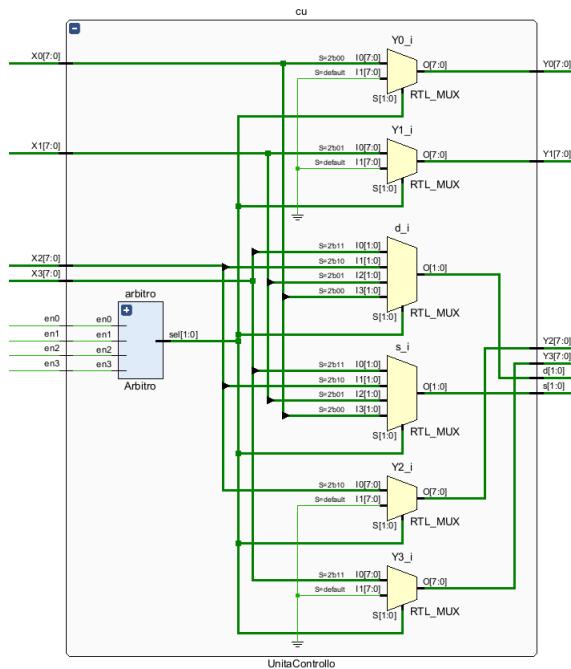
- Blocco: il nodo che non è in grado di propagare i segmenti interrompe l'avanzare del wormhole finché non si liberano le risorse necessarie (non riceve più segmenti)
- Perdita pacchetti: che non è in grado di propagare i segmenti si limita a distruggerli (rete con perdita di dati).
- Re-instradamento: si stabilisce un percorso alternativo a quello bloccato (possibile solo se le tabelle di routing non sono statiche e se la topologia della rete lo permette - nel perfect shuffling ciò non è possibile)
- Cut-through: consiste nell'adottare una filosofia storeforward: i segmenti che non possono essere inoltrati vengono bufferizzati finché c'è spazio

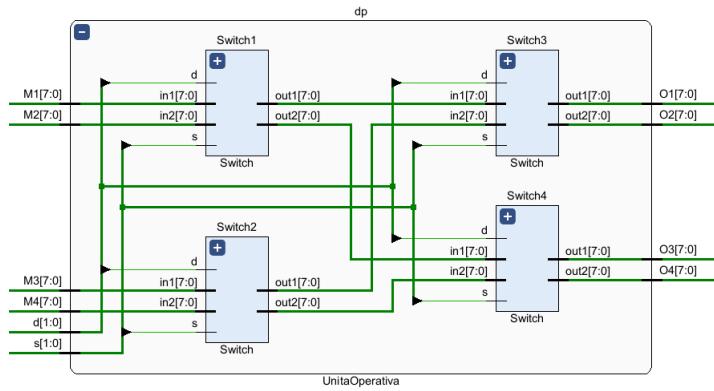
### 11.3 Progetto e architettura

Si é scelto di suddividere il progetto di design in Unitá Operativa ed Unitá di Controllo. L'approccio adottato per la realizzazione del progetto é di tipo strutturale e prevede la composizione di elementi elementari, partendo da mux2:1, demux1:2 che vanno a formare la cella di switch elementare che verrá istanziato 4 volte come richiesto. Si é implementato uno schema a prioritá fissa inserendo nell'unità di controllo un "arbitro" con lo scopo di permettere la trasmissione del messaggio solo al nodo con valore piú elevato.



La parte operativa é composta da 4 switch opportunamente interconnessi applicando la tecnica del perfect shuffling.





## 11.4 Implementazione

Si è proceduto all’implementazione partendo dalla cella elementare switch, ottenuta per composizione dei componenti elementari mux2:1 e demux1:2 (consultabili in appendice):

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Switch is
5 port (
6     in1 : in std_logic_vector(7 downto 0);
7     in2 : in std_logic_vector(7 downto 0);
8     s : in std_logic;
9     d : in std_logic;
10    out1 : out std_logic_vector(7 downto 0);
11    out2 : out std_logic_vector(7 downto 0)
12 );
13 end Switch;
14
15 architecture Structural of Switch is

```

```

15 signal muxtodemux : std_logic_vector(7 downto 0);
16 begin
17 mux : entity work.Mux2_1 generic map(
18 N => 8)
19 port map(
20 a1 => in1, a2 => in2, s => s, y => muxtodemux
21 );
22
23 demux : entity work.Demux2_1 generic map(N => 8)
24 port map(
25 a => muxtodemux, s => d, y1 => out1, y2 => out2
26 );
27 end Structural;

```

I singoli switch vengono poi richiamati ed istanziati nella unitá operativa, descritta attraverso un approccio strutturale:

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity UnitaOperativa is port(
5     s, d : in std_logic_vector(1 downto 0);
6     M1, M2, M3, M4 : in std_logic_vector(7 downto 0);
7     O1, O2, O3, O4 : out std_logic_vector(7 downto 0)
8 );
9 end UnitaOperativa;
10
11 architecture Structural of UnitaOperativa is
12 signal link_out00: std_logic_vector(7 downto 0);

```

```
13 signal link_out01: std_logic_vector(7 downto 0);
14 signal link_out10: std_logic_vector(7 downto 0);
15 signal link_out11: std_logic_vector(7 downto 0);
16 begin
17
18 Switch1 : entity work.Switch port map(
19     in1 => M1,
20     in2 => M2,
21     s => s(0),
22     d => d(1),
23     out1 => link_out00,
24     out2 => link_out01
25 );
26
27 Switch2 : entity work.Switch port map(
28     in1 => M3,
29     in2 => M4,
30     s => s(0),
31     d => d(1),
32     out1 => link_out10,
33     out2 => link_out11
34 );
35
36 Switch3 : entity work.Switch port map(
37     in1 => link_out00,
38     in2 => link_out10,
39     s => s(1),
40     d => d(0),
```

```
41      out1 => O1,
42      out2 => O2
43  );
44
45 Switch4 : entity work.Switch port map(
46     in1 => link_out01,
47     in2 => link_out11,
48     s => s(1),
49     d => d(0),
50     out1 => O3,
51     out2 => O4
52 );
53
54 end Structural;
```

Per determinare il percorso da uno specifico nodo sorgente verso un nodo destinazione si valuta ad ogni stadio il singolo bit associato all'indirizzo di destinazione (si considera come selezione dei mux e demux del primo stadio il primo bit della sorgente ed il secondo della destinazione):

1. Se il valore del bit è 0 allora si procede verso il ramo superiore del blocco
2. Viceversa se il valore del bit è 0 si procede verso il ramo di uscita inferiore del blocco

L'arbitro rappresenta una logica di controllo, prende in ingresso 4 bit

di selezione e restituisce un vettore di 2 bit che abilita un porto dando priorità al nodo con valore maggiore, il 4:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity Arbitro is port(
5     en0 : in std_logic;
6     en1 : in std_logic;
7     en2 : in std_logic;
8     en3 : in std_logic;
9     sel: out std_logic_vector(1 downto 0)
10 );
11 end Arbitro;
12
13 architecture Dataflow of Arbitro is
14 begin
15 process_Arbitro : process(en0,en1,en2,en3)
16 begin
17     if(en0 = '1')then
18         sel <= "00";
19     elsif(en1 = '1') then
20         sel <= "01";
21     elsif(en2 = '1') then
22         sel <= "10";
23     elsif(en3 = '1') then
24         sel <= "11";
25     else
26         sel <= "--";
```

```
27      end if;
28
29  end process;
30
31 end Dataflow;
```

Infine, l'unità di controllo rappresenta la logica di gestione dell'arbitro appena trattato. Essa realizza una vera e propria rete a priorità e allo stesso tempo estrapole il messaggio di ingresso e relativi indirizzi sorgente e destinazione per poi trasmetterli all'unità operativa:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4
5 entity UnitaControllo is port(
6     X0 : in std_logic_vector(7 downto 0);
7     X1 : in std_logic_vector(7 downto 0);
8     X2 : in std_logic_vector(7 downto 0);
9     X3 : in std_logic_vector(7 downto 0);
10
11    en0 : in std_logic;
12    en1 : in std_logic;
13    en2 : in std_logic;
14    en3 : in std_logic;
15
16    Y0 : out std_logic_vector(7 downto 0);
17    Y1 : out std_logic_vector(7 downto 0);
18    Y2 : out std_logic_vector(7 downto 0);
19    Y3 : out std_logic_vector(7 downto 0);
```

```
20
21     s,d: out std_logic_vector(1 downto 0)
22 );
23
24 end UnitaControllo;
25
26 architecture structural of UnitaControllo is
27 signal sel : std_logic_vector (1 downto 0);
28
29 begin
30
31 arbitro: entity work.Arbitro port map(
32     en0=>en0,
33     en1=>en1,
34     en2=>en2,
35     en3=>en3,
36     sel=>sel
37 );
38
39
40 --logica di controllo a prior[U+FFFD]
41
42 Y3 <= X3 when sel = "11" else(others=>'0');
43 Y2 <= X2 when sel = "10" else(others=>'0');
44 Y1 <= X1 when sel = "01" else(others=>'0');
45 Y0 <= X0 when sel = "00" else(others=>'0');
46
47 with sel select
```

```
48      s<=X3(7 downto 6) when "11",
49      X2(7 downto 6) when "10",
50      X1(7 downto 6) when "01",
51      X0(7 downto 6) when "00",
52      "—" when others;
53
54  with sel select
55    d<=X3(5 downto 4) when "11",
56    X2(5 downto 4) when "10",
57    X1(5 downto 4) when "01",
58    X0(5 downto 4) when "00",
59    "—" when others;
60
61
62 end structural;
```

## 11.5 Simulazione

## CHAPTER 11. SWITCH MULTISTADIO

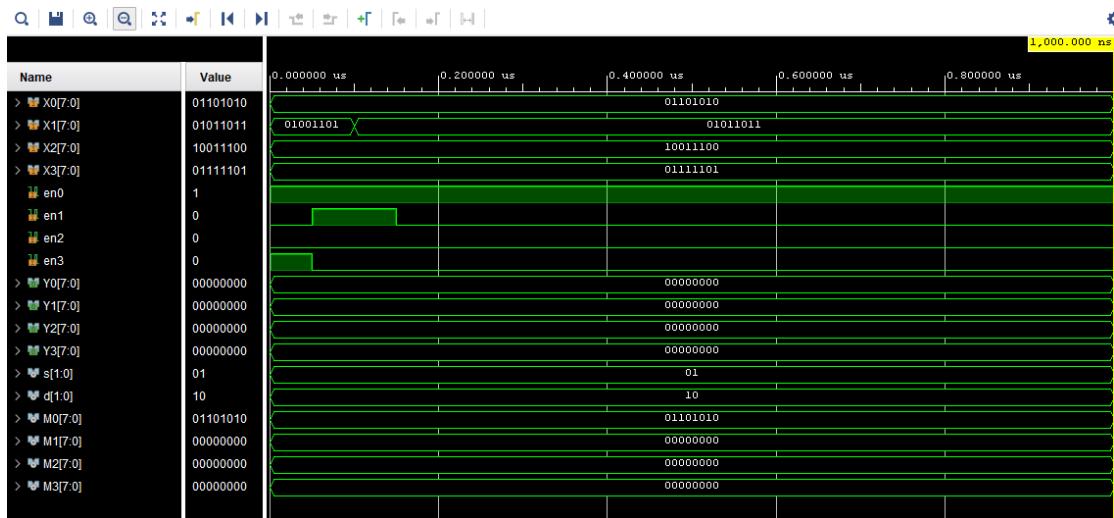


Figure 11.1: Switch multistadio



Figure 11.2: Omega Network

# Chapter 12

## Appendice

### 12.1 Flip Flop D

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity FF_D is
5     port (
6         clk, reset, d: in std_logic;
7         y: out std_logic := '0'
8     );
9 end FF_D;
10
11 architecture behavioural of FF_D is
12
13 begin
14
15     FFD: process(clk)
```

```

16      begin
17          if rising_edge(clk) then
18              if(reset='1') then
19                  Y<='0';
20              else
21                  Y<=d;
22              end if;
23          end if;
24      end process;
25
26  end behavioural;

```

## 12.2 MUX 2:1

```

1 --MUX INDIRIZZABILE 2:1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 --Entity
6 entity mux_2_1 is
7     port (
8         A : in std_logic_vector(1 downto 0); --2
9             ingressi
10            S : in std_logic; --selezione
11            Y : out std_logic --uscita
12        );
13    end mux_2_1;

```

```

13
14 --Architettura livello Dataflow
15 --quindi specificando la tabella di
16 architecture dataflow of mux_2_1 is
17
18   begin
19     Y <= ((A(0) AND (NOT S)) OR (A(1) AND S));
20       --assegnazione concorrente
21   end dataflow;

```

### 12.3 DEMUX 1:2

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4
5
6 entity Demux2_1 is generic(
7   N : integer := 2);
8 port(
9   a: in std_logic_vector(N-1 downto 0);
10  s: in std_logic;
11  y1: out std_logic_vector(N-1 downto 0);
12  y2: out std_logic_vector(N-1 downto 0)
13 );

```

```
14 end Demux2_1;  
15  
16  
17  
18  
19 architecture Behavioral of Demux2_1 is  
20  
21 begin  
22 process_demux : process(a,s)  
23 begin  
24 case s is  
25 when '0' => y1 <= a;  
26 when '1' => y2 <= a;  
27 end case;  
28 end process;  
29 end Behavioral;
```

## 12.4 ROM

```
1 library IEEE;  
2 use IEEE.STD_LOGIC_1164.ALL;  
3  
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;  
5  
6 entity ROM is port(  
7   CLK: in std_logic;  
8   READ_ROM: in std_logic;
```

```
9      ADDRESS: in std_logic_vector(2 downto 0);
10     DATA_OUT: out std_logic_vector(7 downto 0)
11   );
12 end ROM;
13
14 architecture Behavioral of ROM is
15 type registri is array (7 downto 0) of
16   std_logic_vector(7 downto 0);
17 signal memoria : registri := (
18   "00101011",
19   "10101000",
20   "10011001",
21   "11110001",
22   "00001001",
23   "10110101",
24   "00111001",
25   "01110101"
26 );
27
28 begin
29   memo_behavioral: process(CLK)
30   begin
31     if(rising_edge(CLK)) then
32       if(READ_ROM = '1') then
33         DATA_OUT <=
34           memoria(conv_integer(ADDRESS));
35       end if;
36     end if;
37   end process;
38 end Behavioral;
```

```

35      end process;
36  end Behavioral;
```

## 12.5 ROM Comb

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.std_logic_unsigned.ALL;
4
5  entity rom is
6    port (
7      addr: in std_logic_vector(2 downto 0);
8      data_out: out std_logic_vector(7 downto 0)
9    );
10
11
12 end rom;
13
14 architecture Behavioral of rom is
15
16 type rom_type is array (0 to 7) of std_logic_vector(7
17   downto 0);
18 signal ROM: rom_type := (
19   X"00",
20   X"11",
21   X"22",
22   X"33",
```

```

22      X"44",
23      X"55",
24      X"66",
25      X"77"
26 );
27
28 begin
29
30   data_out <= ROM(conv_integer(addr));
31
32 end Behavioral;

```

## 12.6 Contatore MOD M

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.math_real.all;
4 use IEEE.NUMERIC_STD.ALL;
5
6 entity Contatore is
7 generic(
8   M : integer := 60
9 ); port (
10   CLK : in std_logic ;
11   RST : in std_logic;
12   ENABLE : in std_logic;

```



```

37      elsif (ENABLE='1') then
38          if (to_integer(unsigned(T)) >= M) then
39              T<=(others=>'0');
40          else
41              T <= std_logic_vector(unsigned(T)
42                                  +1);
43          end if;
44      end if;
45  end process;
46 COUNT<=T;
47 end Behavioral;

```

## 12.7 Counter MOD 8

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity counter_mod8 is
6     port (
7         clk: in std_logic;
8         reset: in std_logic;
9         enable: in std_logic;
10        count: out std_logic_vector(2 downto 0)
11    );
12 end counter_mod8;

```

```
13
14 architecture Behavioral of counter_mod8 is
15
16 signal temp_c: std_logic_vector(2 downto 0) := (others
17 => '0');
18 begin
19
20 proc: process(clk)
21
22 begin
23
24 if rising_edge(clk) then
25
26 if (reset = '1') then
27
28 temp_c <= (others => '0');
29 else
30
31 if (enable = '1') then
32
33 temp_c <=
34 std_logic_vector(unsigned(temp_c) +
35
36 1);
37 end if;
38
39 end if;
40
41 end if;
42
43 end process;
44
45
46 count <= temp_c;
47
48
49 end Behavioral;
```

## 12.8 Memoria RW

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 use IEEE.MATH_REAL.ALL;
5 use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7
8 entity MemoriaRW is generic(
9     NumeroLocazioni: integer := 16;
10    LunghezzaWord: integer := 8
11 );
12
13 port (
14     CLK: in std_logic;
15     RST: in std_logic;
16     R: in std_logic;      --Read
17     W: in std_logic;      --Write
18     ADDRESS: in
19         std_logic_vector(integer(ceil(log2(real(NumeroLocazioni)))))-
20             downto 0);
21     DATA_INPUT: in std_logic_vector(LunghezzaWord-1
22             downto 0);
23     DATA_OUT: out std_logic_vector(LunghezzaWord-1
24             downto 0)
25 );
26
27 end MemoriaRW;
28
29
30 architecture Behavioral of MemoriaRW is

```

```

24  TYPE registri is array (NumeroLocazioni-1 downto 0) of
25      std_logic_vector(LunghezzaWord-1 downto 0);
26
27  signal memoria : registri;
28
29 begin
30
31     memo_behavioral: process(CLK)
32     begin
33         if(rising_edge(CLK)) then
34             if(RST = '1') then
35                 for k in 0 to NumeroLocazioni-1 loop
36                     memoria(k) <= (others =>'0');
37                     DATA_OUT <= (others => '0');
38                 end loop;
39             elsif(W = '1') then
40                 memoria(conv_integer(ADDRESS)) <=
41                     DATA_INPUT;
42             elsif(R = '1') then
43                 DATA_OUT <=
44                     memoria(conv_integer(ADDRESS));
45             end if;
46         end if;
47     end process;
48 end Behavioral;

```

## 12.9 Adder

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.all;
4
5
6 entity Adder is generic(
7     N: integer := 8
8 ); port(
9     x: in std_logic_vector(N-1 downto 0);
10    y: in std_logic_vector(N-1 downto 0);
11    output: out std_logic_vector(N-1 downto 0)
12 );
13 end Adder;
14
15 architecture behavioral of adder is
16 begin
17
18    output <= std_logic_vector(unsigned(x)+ unsigned(y));
19 end architecture;
```

## 12.10 Registro

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

```

6  entity Registro is generic(
7      M: integer := 8
8  ); port(
9      CLK: in std_logic;
10     RST: in std_logic;
11     input: in std_logic_vector(M-1 downto 0);
12     enable: in std_logic;
13     output: out std_logic_vector(M-1 downto 0)
14 );
15 end Registro;
16
17 architecture Behavioral of Registro is
18 signal reg: std_logic_vector(M-1 downto 0) := (others
19 => 'U');
20 begin
21     reg_behavioral: process(CLK)
22     begin
23         if(rising_edge(CLK)) then
24             if(RST = '1') then
25                 reg <= (others =>'0');
26             elsif(enable = '1') then
27                 reg <= input;
28             end if;
29         end if;
30     end process;
31     output <= reg;
32 end Behavioral;

```

## 12.11 Full Adder

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity full_adder is
5   port(
6     a: in std_logic; -- Primo operando del Full
7         Adder
8     b: in std_logic; -- Secondo operando del Full
9         Adder
10    cin: in std_logic; -- Carry-In, il riporto da
11      un'eventuale addizione precedente
12    cout: out std_logic; -- Carry-Out, il riporto
13      risultante dall'addizione dei due operandi e
14      del Carry-In
15    s: out std_logic -- Sum bit, il risultato
16      dell'addizione dei due operandi e del
17      Carry-In
18
19  );
20
21 end full_adder;
22
23
24 architecture dataflow of full_adder is
25
26 begin
27
28   -- Assegna al port di uscita S il risultato
29   -- dell'operazione XOR tra gli operandi e il
30   -- Carry-In.
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
678
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
746
747
748
748
749
749
750
751
752
753
754
755
756
756
757
758
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
776
777
778
778
779
779
780
781
782
783
784
785
785
786
787
787
788
788
789
789
790
791
792
793
794
795
795
796
797
797
798
798
799
799
800
801
802
803
804
805
805
806
807
807
808
808
809
809
810
811
812
813
814
815
815
816
817
817
818
818
819
819
820
821
822
823
824
825
825
826
827
827
828
828
829
829
830
831
832
833
834
835
835
836
837
837
838
838
839
839
840
841
842
843
844
845
845
846
847
847
848
848
849
849
850
851
852
853
854
855
855
856
857
857
858
858
859
859
860
861
862
863
864
865
865
866
867
867
868
868
869
869
870
871
872
873
874
875
875
876
877
877
878
878
879
879
880
881
882
883
884
885
885
886
887
887
888
888
889
889
890
891
892
893
894
894
895
896
896
897
897
898
898
899
899
900
901
902
903
903
904
905
905
906
906
907
907
908
908
909
909
910
911
912
913
913
914
915
915
916
916
917
917
918
918
919
919
920
921
922
923
924
924
925
926
926
927
927
928
928
929
929
930
931
932
933
934
934
935
936
936
937
937
938
938
939
939
940
941
942
943
943
944
945
945
946
946
947
947
948
948
949
949
950
951
952
953
953
954
955
955
956
956
957
957
958
958
959
959
960
961
962
963
963
964
965
965
966
966
967
967
968
968
969
969
970
971
972
973
973
974
975
975
976
976
977
977
978
978
979
979
980
981
982
983
983
984
985
985
986
986
987
987
988
988
989
989
990
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630

```

```

19      -- Questo valore del bit di somma per
20      -- l'addizione binaria.
21      s <= (a xor b) xor cin;
22      -- Assegna al port di uscita COUT il risultato
23      -- dell'operazione OR tra:
24      -- 1. L'AND degli operandi, che genera un carry se
25      -- entrambi gli operandi sono 1.
26      -- 2. L'AND tra il Carry-In e l'OR degli operandi,
27      -- che genera un carry se uno degli operandi e il
28      -- Carry-In sono 1.
29      -- Questo valore del Carry-Out per
30      -- l'addizione binaria.
31      cout <= (a and b) or (cin and (a or b));
32
33
34  end dataflow;

```

## 12.12 Ripple Carry Adder

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity ripple_carry is
5     port (
6         X, Y: in std_logic_vector(7 downto 0); --
7             Ingressi: due operandi a 8 bit da sommare.
8         c_in: in std_logic; -- Ingresso: il riporto
9             iniziale.

```

```
8      c_out: out std_logic; -- Uscita: il riporto
9          finale.
10
11     Z: out std_logic_vector(7 downto 0) -- Uscita:
12         risultato della somma a 8 bit.
13
14 );
15
16 end ripple_carry;
17
18
19 architecture structural of ripple_carry is
20
21     component full_adder is
22
23         port (
24
25             a,b: in std_logic;
26
27             cin: in std_logic;
28
29             cout, s: out std_logic);
30
31     end component;
32
33
34
35     signal temp: std_logic_vector(7 downto 0);
36
37
38 begin
39
40
41     RA0: full_adder port map(X(0), Y(0), c_in, temp(0),
42
43         Z(0));
44
45
46     RA1to6: FOR i IN 1 TO 6 GENERATE
47
48         RA: full_adder port map(X(i), Y(i),
49
50             temp(i-1), temp(i), Z(i));
51
52     END GENERATE;
```

```

31    RA7: full_adder port map(X(7), Y(7), temp(6),
32                            c_out, Z(7));
33 end structural;

```

## 12.13 Adder / Subtractor

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity adder_sub is
5   port (
6     X, Y: in std_logic_vector(7 downto 0); --
7           Ingressi: due operandi a 8 bit.
8     cin: in std_logic; -- Ingresso: segnale di
9           controllo per selezionare addizione ('0') o
10          sottrazione ('1').
11     Z: out std_logic_vector(7 downto 0); -- Uscita:
12          risultato dell'addizione o sottrazione.
13     cout: out std_logic -- Uscita: riporto in
14          uscita per la sottrazione.
15   );
16 end adder_sub;
17
18
19 architecture Structural of adder_sub is
20
21   component ripple_carry is

```

```

16      port (
17          X, Y: in std_logic_vector(7 downto 0);
18          c_in: in std_logic;
19          c_out: out std_logic;
20          Z: out std_logic_vector(7 downto 0)
21      );
22  end component;
23
24  -- Segnale interno usato per mantenere la versione
25  -- complementata di Y.
26
27 signal complementoy: std_logic_vector(7 downto 0);
28
29 begin
30
31  -- Generazione del complemento di Y in base al
32  -- valore di cin.
33  -- Se cin = '0' (addizione), Y rimane invariato.
34  -- Se cin = '1' (sottrazione), Y viene
35  -- complementato.
36  complemento_y: FOR i IN 0 TO 7 GENERATE
37      complementoy(i) <= Y(i) xor cin; -- Operazione
38      xor per complementare Y se necessario.
39  END GENERATE;
40
41
42  -- Istanza del componente ripple_carry
43  RA: ripple_carry port map(X, complementoy, cin,
44      cout, Z);
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
639
639
640
641
642
643
644
644
645
646
646
647
647
648
648
649
649
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558

```

```
39 | end Structural;
```

## 12.14 Button Debouncer

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity ButtonDebouncer is
6     generic(
7         CLK_period: integer := 10; -- periodo del clock
8             della board (in nanosecondi)
9             btn_noise_time: integer := 10000000 -- durata
10            stimata dell'oscillazione del bottone
11            -- il
12            valore
13            di
14            default
15            [U+FFFØ
16            millisecondi
17        );
18
19     port (
20         RST: in std_logic;
21         CLK: in std_logic;
22         BTN: in std_logic;
23         CLEARED_BTN: out std_logic -- segnale di output
24             ripulito
25     );
26 end entity;

```

```
16      );
17 end ButtonDebouncer;
18
19 architecture Behavioral of ButtonDebouncer is
20
21 type state is (NOT_PRESSED, CHK_PRESSED, PRESSED,
22                 CHK_NOT_PRESSED);
23
24 signal BTN_state: state := NOT_PRESSED;
25
26 constant max_count : integer :=
27     btn_noise_time/CLK_period; -- 10000000/10= conto
28     1000000 colpi di clock
29
30 begin
31
32     deb: process(CLK)
33
34         variable count: integer := 0;
35
36         begin
37
38             if rising_edge(CLK) then
39
40                 if(RST = '1') then
41                     BTN_state <= NOT_PRESSED;
42                     CLEARED_BTN <= '0';
43                 else
44                     case BTN_state is
```

```

41      when NOT_PRESSED =>
42          if(BTN = '1') then
43              BTN_state <=
44                  CHK_PRESSED;
45          else
46              BTN_state <=
47                  NOT_PRESSED;
48          end if;
49
50      when CHK_PRESSED =>
51          if(count = max_count-1)
52              then --conta i cicli di
53                  clock fino a max_count-1
54                  (ovvero 10ms)
55          if(BTN = '1') then --se
56              arrivo a count max
57                  #d+FFF#cora alto
58                  vuol dire che non
59                  era un bounce, devo
60                  alzare CLEARED_BTN
61          count := 0;
62          CLEARED_BTN <= '1';
63          BTN_state <=
64              PRESSED;
65      else
66          count := 0;
67          BTN_state <=
68              NOT_PRESSED;

```

```

57           end if;

58       else

59           count := count+1;

60           BTN_state <=
61                           CHK_PRESSED;

62           end if;

63
64           when PRESSED => --Abbasso subito
65                           CLEARED_BIN per avere un
66                           singolo impulso
67                           CLEARED_BTN <= '0';

68           if(BTN = '0') then
69               BTN_state <=
70                           CHK_NOT_PRESSED;
71           else
72               BTN_state <= PRESSED;
73           end if;

74
75           when CHK_NOT_PRESSED =>
76               if(count = max_count-1) then
77                   if(BTN = '0') then --se
78                       arrivo a count max
79                       ¢d+FFF¾cra basso
80                       vuol dire che non
81                       era un bounce e il
82                       bottone ¢+FFF¾ato
83                       rilasciato
84
85               count := 0;

```

```
75          BTN_state <=
76              NOT_PRESSED;
77
78      else
79          count := 0;
80          BTN_state <=
81              PRESSED;
82
83      end if;
84
85      else
86          count := count+1;
87          BTN_state <=
88              CHK_NOT_PRESSED;
89
90      end if;
91
92  end Behavioral;
```