



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

Architettura dei Sistemi Digitali

Anno Accademico 2023/2024

Prof.ssa Alessandra De Benedictis

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

1 Multiplexer	1
1.1 Traccia	1
1.2 Progetto e architettura	2
1.2.1 Multiplexer 16:1	2
1.2.2 Rete di interconnessione	2
1.3 Implementazione	3
1.3.1 Multiplexer 16:1	3
1.4 Simulazione	5
1.4.1 Multiplexer 16:1	5
1.4.2 Rete di Interconnessione	6
1.5 Sintesi su board di sviluppo	8
2 Sistema ROM+M	12
2.1 Traccia	12
2.2 Progetto e architettura	13
2.3 Implementazione	13
2.3.1 Memoria ROM 16 locazioni 8 bit	13
2.3.2 Macchina combinatoria: shift di 4 locazioni	15
2.3.3 Sistema complessivo ROM+M	16
2.3.4 Simulazione	17
2.4 Sintesi e implementazione su board di sviluppo	19
3 Riconoscitore di sequenze 101	21
3.1 Traccia	21
3.2 Automa	22

3.3	Progetto e architettura	23
3.4	Implementazione	23
3.5	Simulazione	31
3.6	Sintesi su board di sviluppo	35
3.7	Timing Analysis	44
4	Shift Register	49
4.1	Traccia	49
4.2	Progetto e architettura	49
4.3	Implementazione: approccio behavioral	50
4.4	Implementazione: approccio structural	53
4.5	Simulazione: approccio behavioral	59
4.6	Simulazione: approccio strutturale	60
5	Cronometro	61
5.1	Traccia 4.1	61
5.1.1	Progetto e architettura	61
5.1.2	Implementazione	62
5.1.3	Simulazione	71
5.2	Traccia 5.2	71
5.2.1	Progetto e architettura	72
5.2.2	Sintesi su board	86
5.3	Traccia 5.3	92
5.3.1	Progetto e architettura	92
5.3.2	Implementazione	95
5.3.3	Sintesi su Board	101
6	Sistema PO_PC	106
6.1	Traccia	106
6.2	Progetto e architettura	106
6.3	Implementazione	108
6.3.1	Control Unit	108
6.3.2	SistemaPO_PC	110
6.3.3	Macchina Combinatoria	112

6.4	Simulazione	113
6.5	Implementazione su Board	113
6.5.1	CU_sintesi	114
6.5.2	SistemaOnBoard	116
7	Moltiplicatore Booth	122
7.1	Traccia	122
7.2	Progetto e Architettura	122
7.3	Implementazione	125
7.3.1	Booth Multiplier	125
7.3.2	Unità di Controllo	127
7.3.3	Unità Operativa	130
7.3.4	Shift Register	133
7.3.5	Registro M	135
7.3.6	Addizionatore - Sottrattore	136
7.4	Implementazione su Board	139
8	Handshaking	144
8.1	Traccia	144
8.2	Progetto e Architettura	144
8.3	Implementazione	148
8.3.1	Nodo A	148
8.3.2	CU A	150
8.3.3	Nodo B	152
8.3.4	CU B	154
8.4	Simulazione	156
9	Processore MIC-1	158
9.1	Traccia	158
9.2	Architettura del processore	158
9.2.1	Datapath	160
9.2.2	Control Unit	162
9.3	Istruzioni Analizzate	164
9.3.1	BIPUSH	164

9.3.2 IOR	166
9.3.3 Istruzione modificata: IADD3OP	168
10 Interfaccia Seriale	171
10.1 Traccia	171
10.2 Progetto e architettura	171
10.3 Trasmettitore TX - Architettura UART	173
10.4 Ricevitore RX - Architettura UART	174
10.5 Implementazione progetto	176
10.5.1 Unità A	176
10.5.2 Automa a stati finiti - CU A	177
10.5.3 Unità B	178
10.5.4 Automa a stati finiti - CU B	178
10.6 Implementazione	179
10.6.1 Comunicazione_Seriale	179
10.6.2 CU A	180
10.6.3 Unita A	183
10.6.4 CU B	184
10.6.5 Unita B	187
10.7 Simulazione	189
11 Switch Multistadio	190
11.1 Traccia	190
11.2 Cenni teorici	190
11.3 Progetto e architettura	194
11.4 Implementazione	195
11.5 Simulazione	201
12 Appendice	203
12.1 Flip Flop D	203
12.2 MUX 2:1	204
12.3 DEMUX 1:2	205
12.4 ROM	205
12.5 ROM Comb	207

12.6 Contatore MOD M	208
12.7 Counter MOD 8	209
12.8 Memoria RW	211
12.9 Adder	212
12.10 Registro	213
12.11 Full Adder	214
12.12 Ripple Carry Adder	215
12.13 Adder / Subtractor	216
12.14 Button Debouncer	218

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

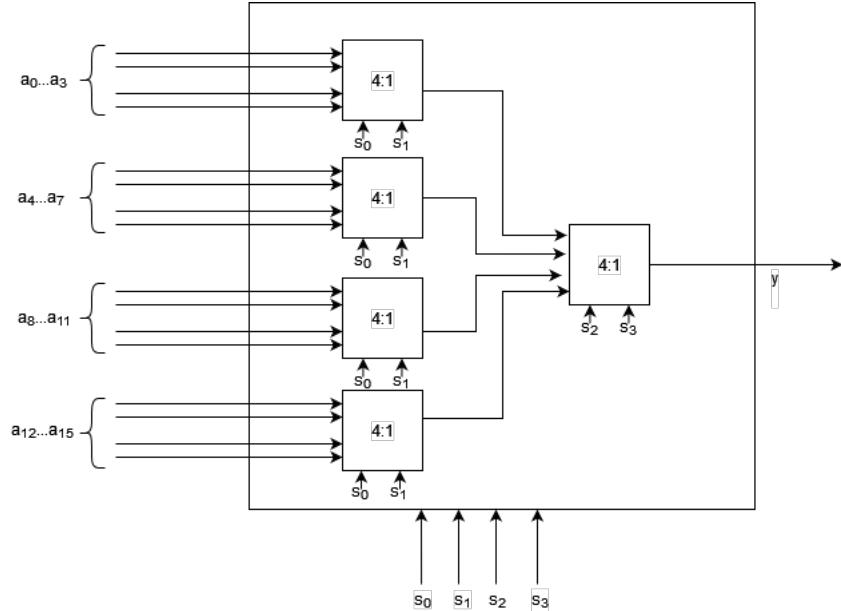


Figure 1.1: MUX 16:1

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 costituiti 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.

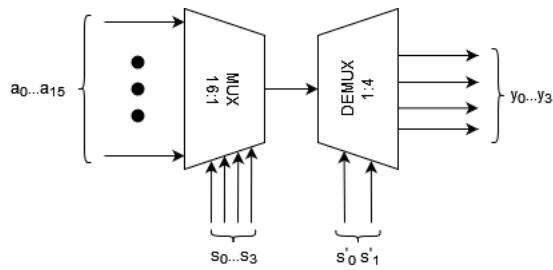


Figure 1.2: Rete di interconnessione

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 di ingressi
4           in input (dim=16)
5     S: in std_logic_vector(3 downto 0);--vettore di ingressi
6           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 ingressi dato
6              S: in std_logic_vector(1 downto 0); --2 ingressi di
7                  selezione
8              Y: out std_logic --1 uscita
9          );
10
11     --vettore per memorizzare le uscite dei mux di primo livello
12     signal U: std_logic_vector(3 downto 0);
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 mux_4_1
19             M: mux_4_1 port map(
20                 --ingressi 3,2,1,0 - 7,6,5,4 - 11,10,9,8 -
21                 15,14,13,12
22                 A => B(i*4+3 downto i*4),
23                 S => S(1 downto 0), --intervengono solo i primi 2
24                     bit di selezione
25                 Y => U(i)
26                 --le uscite dei multiplexer di primo livello
27                 vengono memorizzate in U
28             );
29
30     END GENERATE;
31
32 
```

```

28      M_4: IF i = 4 GENERATE --Genero l'ultimo mux_4_1 (di
29          secondo livello)
30          M: mux_4_1 port map (
31              A => U,--prende in ingresso le uscite dei mu di
32                  primo livello
33              S => S(3 downto 2), --ultimi 2 bit di selezione
34              Y => Y --uscita del mux_16_1
35          );
36      END GENERATE;
37  END GENERATE;
38
39 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 lavorano 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 di
8       ingresso di 16 bit
9     S: in std_logic_vector(3 downto 0); -- Vettore di
10      selezione di 4 bit per il Mux
11     L: in std_logic_vector(1 downto 0); -- Vettore di
12       selezione di 2 bit per il Demux
13     Y: out std_logic_vector(3 downto 0); -- Vettore di uscita
14   );
15 end entity;

```

```

di 4 bit dal Demux
);
end rete_interconnessione;

architecture Structural of rete_interconnessione is

signal U: std_logic; -- Segnale intermedio tra Mux e Demux

-- Dichiarazione del componente mux_16_1
component mux_16_1 is
port(
    B: in std_logic_vector(15 downto 0); -- Ingresso di 16
        bit per il Mux
    S: in std_logic_vector(3 downto 0); -- Selezione di 4
        bit per il Mux
    Y: out std_logic                         -- Uscita singola
        dal Mux
);
end component;

-- Dichiarazione del componente demux_1_4
component demux_1_4 is
port(
    A: in std_logic;                      -- Ingresso
        singolo per il Demux
    S: in std_logic_vector(1 downto 0); -- Selezione di 2
        bit per il Demux
    Y: out std_logic_vector(3 downto 0) -- Uscita di 4 bit
        dal Demux
);
end component;

begin
-- Istanziamento del Mux

```

```

38 Mux: mux_16_1 port map(
39     B => A, -- Collegamento ingresso A al Mux
40     S => S, -- Collegamento selezione S al Mux
41     Y => U -- Collegamento uscita Mux al segnale intermedio U
42 );
43
44 -- Istanziamento del Demux
45 Demux: demux_1_4 port map(
46     A => U, -- Collegamento segnale intermedio U al Demux
47     S => L, -- Collegamento selezione L al Demux
48     Y => Y -- Collegamento uscita Demux al vettore di uscita
49     Y
50 );
51 end Structural;

```

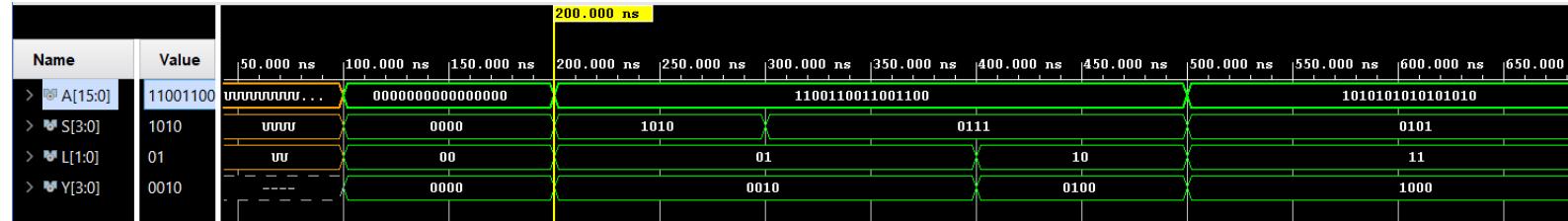


Figure 1.4: Simulazione

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 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  ##Switches
6  set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 }
   [get_ports { switchData[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
7  set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 }
   [get_ports { switchData[1] }]; #IO_L3N_T0_DQS_EMCCCLK_14
   Sch=sw[1]
8  set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 }
   [get_ports { switchData[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
9  set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 }
   [get_ports { switchData[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
10 set_property -dict { PACKAGE_PIN R17     IOSTANDARD LVCMOS33 }
```

```

11      [get_ports { switchData[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
12  set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 }
13      [get_ports { switchData[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
14  set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 }
15      [get_ports { switchData[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
16  set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 }
17      [get_ports { switchData[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
18  set_property -dict { PACKAGE_PIN T8      IOSTANDARD LVCMOS18 }
19      [get_ports { switch_selM[0] }]; #IO_L24N_T3_34 Sch=sw[8]
20  set_property -dict { PACKAGE_PIN U8      IOSTANDARD LVCMOS18 }
21      [get_ports { switch_selM[1] }]; #IO_25_34 Sch=sw[9]
22  set_property -dict { PACKAGE_PIN R16      IOSTANDARD LVCMOS33 }
23      [get_ports { switch_selM[2] }]; #IO_L15P_T2_DQS_RDWR_B_14
24      Sch=sw[10]
25  set_property -dict { PACKAGE_PIN T13      IOSTANDARD LVCMOS33 }
26      [get_ports { switch_selM[3] }]; #IO_L23P_T3_A03_D19_14
27      Sch=sw[11]
28
29  set_property -dict { PACKAGE_PIN H6      IOSTANDARD LVCMOS33 }
30      [get_ports { switch_selD[0] }]; #IO_L24P_T3_35 Sch=sw[12]
31  set_property -dict { PACKAGE_PIN U12      IOSTANDARD LVCMOS33 }
32      [get_ports { switch_selD[1] }]; #IO_L20P_T3_A08_D24_14
33      Sch=sw[13]
34
35
36  ## LEDs
37  set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 }
38      [get_ports { value_out[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
39  set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 }
40      [get_ports { value_out[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
41  set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMOS33 }
42      [get_ports { value_out[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
43  set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 }
44      [get_ports { value_out[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
45
46
47  #Buttons

```

```
28 set_property -dict { PACKAGE_PIN N17    IOSTANDARD LVCMOS33 }
      [get_ports { reset }]; #IO_L9P_T1_DQS_14 Sch=btnc
29 set_property -dict { PACKAGE_PIN M18    IOSTANDARD LVCMOS33 }
      [get_ports { load_sel }]; #IO_L4N_T0_D05_14 Sch=btnu
30 set_property -dict { PACKAGE_PIN P17    IOSTANDARD LVCMOS33 }
      [get_ports { load_first_part }]; #IO_L12P_T1_MRCC_14 Sch=btln
31 set_property -dict { PACKAGE_PIN M17    IOSTANDARD LVCMOS33 }
      [get_ports { load_second_part }]; #IO_L10N_T1_D15_14 Sch=btnr
```

Chapter 2

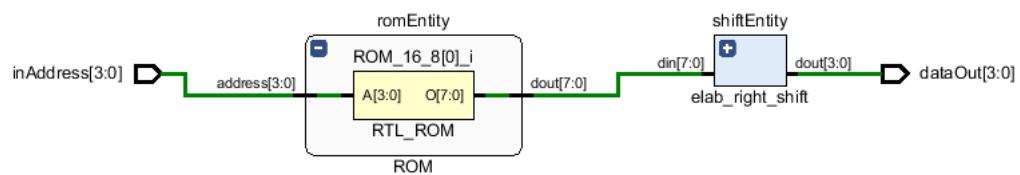
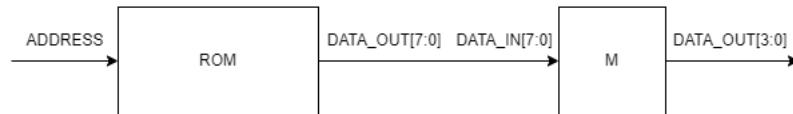
Sistema ROM+M

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
14     downto 0);
15   constant ROM_16_8 : MEMORY_16_8 := (
16     x"00",
17     x"11",
18     x"22",
19     x"33",
20     x"44",
21     x"55",
22     x"66",
23     x"77",
24     x"88",
25     x"99",
26     x"aa",
27     x"bb",
28     x"cc",
29     x"dd",
30     x"ee",
31     x"ff"
32   );
33 begin
34   main : process(address)
35   begin
36     dout <= ROM_16_8(to_integer(unsigned(address)));
37   end process main;
38 end architecture RTL;

```

All'interno dell' architecture viene definito un tipo (type MEMORY_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 locazioni

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 posizioni
15 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   component ROM is
3     port (
4       address : in  std_logic_vector(3 downto 0);
5       dout    : out std_logic_vector(7 downto 0)
6     );
7   end component ROM;
8
9   component elab_right_shift is
10    port (
11      din   : in  std_logic_vector(7 downto 0);
12      dout  : out std_logic_vector(3 downto 0)
13    );
14  end component elab_right_shift;
15
16  signal rom_out : std_logic_vector(7 downto 0);
17

```

```

18 begin
19     romEntity: ROM port map (
20         address => inAddress, --mappa l'ingresso della ROM
21         all'ingresso del sistema
22         --quando un indirizzo viene fornito al sistema,
23         viene passato alla ROM
24         dout    => rom_out --i dati in uscita dalla ROM vengono
25         passati al segnale
26     );
27
28     shiftEntity: elab_right_shift port map (
29         din  => rom_out, --i dati di uscita ROM vengono passati
30         come ingresso alla macchina combinatoria
31         dout => dataOut --mappa l'uscita della macchina
32         all'uscita del sistema
33     );
34
35 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;
7
8 architecture bench of Sistema_tb is
9
10 component Sistema
11     port(

```

```

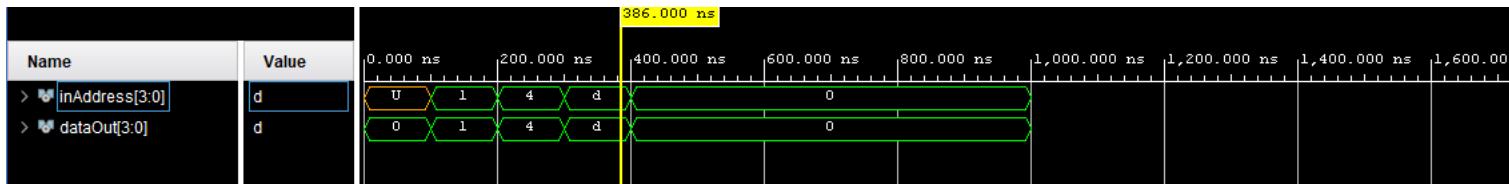
12      inAddress : in  std_logic_vector(3 downto 0);
13      dataOut     : out std_logic_vector(3 downto 0)
14  );
15 end component;
16
17 signal inAddress: std_logic_vector(3 downto 0);
18 signal dataOut: std_logic_vector(3 downto 0) ;
19
20 begin
21
22 uut: Sistema port map ( inAddress => inAddress,
23                           dataOut     => dataOut );
24
25 stimulus: process
26 begin
27
28   wait for 100ns;
29
30   inAddress <= "0001";
31   wait for 100ns;
32   assert dataOut = "0001" report "Test fallito" severity error;
33
34   inAddress <= "0100";
35   wait for 100ns;
36   assert dataOut = "0100" report "Test fallito" severity error;
37
38   inAddress <= "1101";
39   wait for 100ns;
40   assert dataOut = "1101" report "Test fallito" severity error;
41
42   inAddress <= (others => '0');
43   wait for 100ns;
44   assert dataOut = "0" report "Test fallito per tutti gli zeri"
        severity error;

```

```

45
46     inAddress <= (others => '0');
47
48     wait for 100 ns;
49
50     assert dataOut = "1" report "Test fallito per tutti gli uno"
51         severity error;
52
53
54
55 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
3  set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMS33 }
4      [get_ports { inAddress[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
5
6  set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMS33 }
7      [get_ports { inAddress[1] }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
8
9  set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMS33 }
10     [get_ports { inAddress[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
11
12 set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMS33 }
13     [get_ports { inAddress[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]

```

```
1 ## LEDs
2 set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 }
   [get_ports { dataOut[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
3 set_property -dict { PACKAGE_PIN K15    IOSTANDARD LVCMOS33 }
   [get_ports { dataOut[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
4 set_property -dict { PACKAGE_PIN J13    IOSTANDARD LVCMOS33 }
   [get_ports { dataOut[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
5 set_property -dict { PACKAGE_PIN N14    IOSTANDARD LVCMOS33 }
   [get_ports { dataOut[3] }]; #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.

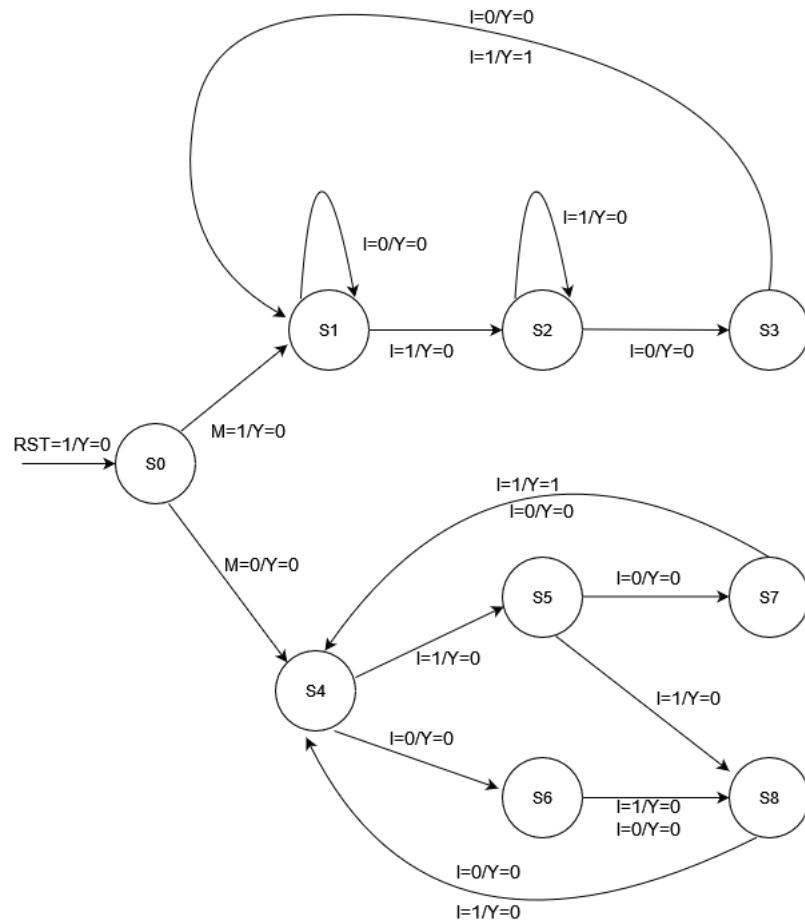
3.1 Traccia

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 temporizzazione e un segnale M di modo, che ne disciplina il funzionamento, e fornisce un'uscita Y alta quando la sequenza viene riconosciuta. In particolare,

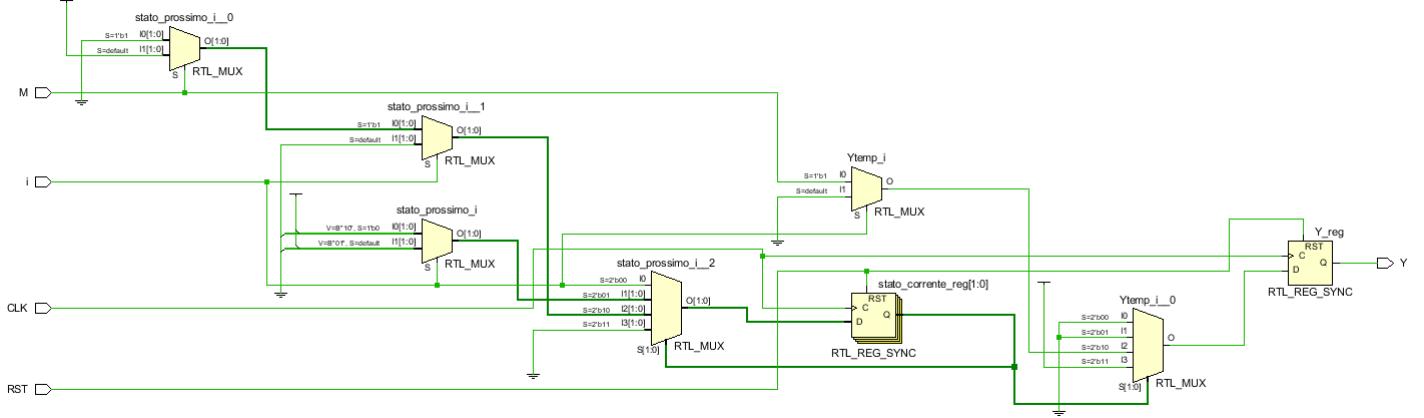
- se $M=0$, la macchina valuta i bit seriali in ingresso a gruppi di 3 (sequenze non sovrapposte), se $M=1$, la macchina valuta i bit seriali in ingresso uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta (sequenze parzialmente sovrapposte).

- 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 temporizzazione 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 è 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 è 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 permetterà 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 è 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 è deciso di adottare un pattern con due 'process' in modo creare un design quanto più 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          RST: in std_logic;
8          CLK: in std_logic;
9          M: in std_logic;
10         Y: out std_logic;
11         state: out std_logic_vector(3 downto 0)
12     );
13 end Riconoscitore_101;
14
15 architecture Behavioral of Riconoscitore_101 is
16
17 type stato is (S0, S1, S2, S3, S4, S5, S6, S7, S8);
18
19 signal stato_corrente: stato := S0;
20 signal stato_prossimo: stato;
21 signal Ytemp: std_logic;
22
23 begin
24
25     stato_uscita: process(i, stato_corrente)
26     begin
27
28         case stato_corrente is
29             when S0 =>
30                 if(M='0') then
31                     stato_prossimo <= S4;
32                     Ytemp <= '0';
33                 else
34                     stato_prossimo <= S1;
35                     Ytemp <= '0';
36                 end if;
37
38             when S1 =>

```

```

39      if(i='1') then
40          stato_prossimo <= S2;
41          Ytemp <= '0';
42      else
43          stato_prossimo <= S1;
44          Ytemp <= '0';
45      end if;
46
47      when S2 =>
48          if(i='1') then
49              stato_prossimo <= S2;
50              Ytemp <= '0';
51          else
52              stato_prossimo <= S3;
53              Ytemp <= '0';
54          end if;
55
56      when S3 =>
57          if(i='1') then
58              stato_prossimo <= S1;
59              Ytemp <= '1';
60          else
61              stato_prossimo <= S1;
62              Ytemp <= '0';
63          end if;
64
65      when S4 =>
66          if(i='1') then
67              stato_prossimo <= S5;
68              Ytemp <= '0';
69          else
70              stato_prossimo <= S6;
71              Ytemp <= '0';
72      end if;

```

```

73
74     when S5 =>
75         if(i='1') then
76             stato_prossimo <= S8;
77             Ytemp <= '0';
78         else
79             stato_prossimo <= S7;
80             Ytemp <= '0';
81         end if;
82
83     when S6 =>
84         Ytemp <= '0';
85         stato_prossimo <= S8;
86
87     when S7 =>
88         if(i='1') then
89             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
mem: process(CLK)

```

```

106 begin
107     if rising_edge(CLK) then
108         if RST = '1' then
109             stato_corrente <= S0;
110             Y <= '0';
111         else
112             stato_corrente <= stato_prossimo;
113             -- Sincronizza Y con il clock
114             Y <= Ytemp;
115         end if;
116     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 label '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 (Ytemp) 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 "one_hot";
19
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         Y <= '0';
28     else
29         if(M='1') then
30             case stato_corrente is
31                 when S0 =>

```

```

32         if( i = '0' ) then
33             stato_corrente <= S0;
34             Y <= '0';
35         else
36             stato_corrente <= S1;
37             Y <= '0';
38         end if;
39
40     when S1 =>
41         if( i = '0' ) then
42             stato_corrente <= S2;
43             Y <= '0';
44         else
45             stato_corrente <= S1;
46             Y <= '0';
47         end if;
48
49     when S2 =>
50         if( i = '0' ) then
51             stato_corrente <= S0;
52             Y <= '0';
53         else
54             stato_corrente <= S0;
55             Y <= '1';
56         end if;
57
58     when others =>
59         stato_corrente <= S0;
60         Y <= '0';
61
62     end case;
63
64     else
65         case stato_corrente is
66             when S0 =>
67                 if i = '1' then
68                     stato_corrente <= S1;
69                 else
70                     stato_corrente <= S0;
71                 end if;
72             end case;
73         end if;
74     end if;
75
76     end process;
77
78 end architecture Behavioral;
```

```

66         end if;
67         Y <= '0';
68     when S1 =>
69         if i = '0' then
70             stato_corrente <= S2;
71         else
72             stato_corrente <= S1;
73         end if;
74         Y <= '0';
75     when S2 =>
76         if i = '1' then
77             stato_corrente <= S3;
78         else
79             stato_corrente <= S0;
80         end if;
81         Y <= '0';
82     when S3 =>
83         -- Aggiungo uno stato info+FFF per gestire la
84         -- non sovrapposizione.
85         stato_corrente <= S0; -- Torna allo stato
86         iniziale dopo aver riconosciuto la sequenza.
87         Y <= '1'; -- Sequenza riconosciuta, l'uscita
88         diventa '1'.
89     when others =>
90         stato_corrente <= S0;
91         Y <= '0';
92     end case;
93     end if;
94 end if;
95 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;
```

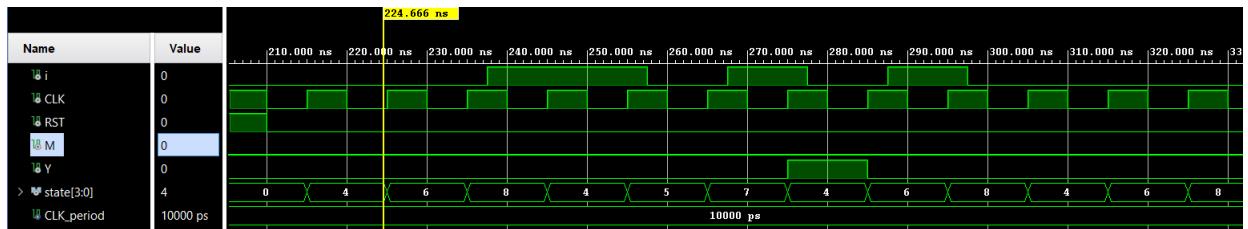


Figure 3.1: M=0

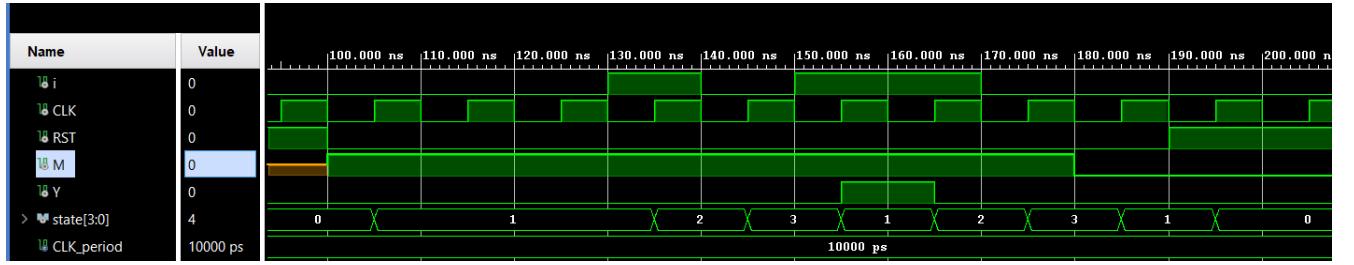


Figure 3.2: M=1

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
13      Y: out std_logic;
14
15      state: out std_logic_vector(3 downto 0)
16
17  );
18
19 end Riconoscitore_onBoard;
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

```

```

      m_read: in std_logic;
      Y: out std_logic;
      state: out std_logic_vector(3 downto 0)
    );
end Riconoscitore_onBoard;

architecture Structural of Riconoscitore_onBoard is

component ButtonDebouncer is
  generic(
    CLK_period: integer := 10;
    btn_noise_time: integer := 10000000
  );
  port(
    RST: in std_logic;
    CLK: in std_logic;
    BTN: in std_logic;
    CLEARED_BTN: out std_logic -- segnale di output
      ripulito
  );
end component;

component Riconoscitore_101 is
  port(
    i: in std_logic;
    RST: in std_logic;
    CLK: in std_logic;
    M: in std_logic;
    i_read: in std_logic;
    M_read: in std_logic;
    Y: out std_logic;
    state: out std_logic_vector(3 downto 0)
  );
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             Ytemp <= '0';
44         else
45             stato_prossimo <= S1;
46             Ytemp <= '0';
47         end if;
48
49     when S2 =>
50         if(i='1') then
51             stato_prossimo <= S2;
52             Ytemp <= '0';
53         else
54             stato_prossimo <= S3;
55             Ytemp <= '0';
56         end if;
57
58     when S3 =>
59         if(i='1') then
60             stato_prossimo <= S1;
```

```
61           Ytemp <= '1';
62
63           else
64               stato_prossimo <= S1;
65               Ytemp <= '0';
66
67           end if;
68
68
69           when S4 =>
70
71               if(i='1') then
72                   stato_prossimo <= S5;
73                   Ytemp <= '0';
74
75               else
76                   stato_prossimo <= S6;
77                   Ytemp <= '0';
78
79               end if;
80
80
81           when S5 =>
82
83               if(i='1') then
84                   stato_prossimo <= S8;
85                   Ytemp <= '0';
86
87               else
88                   stato_prossimo <= S7;
89                   Ytemp <= '0';
90
91               end if;
92
92
93           when S6 =>
94
95               Ytemp <= '0';
96
97               stato_prossimo <= S8;
98
98
99           when S7 =>
100
101               if(i='1') then
102                   stato_prossimo <= S4;
103                   Ytemp <= '1';
104
105               else
106
107                   stato_prossimo <= S4;
```

```

95          Ytemp <= '0';
96      end if;
97
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 mem: process(CLK)
109 begin
110     if rising_edge(CLK) then
111         if RST = '1' then
112             stato_corrente <= S0;
113             Y <= '0';
114         else
115             if(stato_corrente = S0 and m_read='1') then
116                 stato_corrente <= stato_prossimo;
117                 Y <= Ytemp;
118             elsif(stato_corrente /= S0 and i_read='1') then
119                 stato_corrente <= stato_prossimo;
120                 Y <= Ytemp;
121             end if;
122         end if;
123     end if;
124
125     end process;
126
127     with stato_corrente select
128         state <= x"0" when S0,
129                     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      IOSTANDARD LVCMOS33 }
   [get_ports { CLK }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
3  create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
   [get_ports {CLK}];
4
5
6  ##Switches
7  set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 }
   [get_ports { i }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
8  set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 }
   [get_ports { M }]; #IO_L3N_T0_DQS_EMCCCLK_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 }
17     [get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
18 #set_property -dict { PACKAGE_PIN U17      IOSTANDARD LVCMOS33 }
19     [get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
20 #set_property -dict { PACKAGE_PIN U16      IOSTANDARD LVCMOS33 }
21     [get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
22 #set_property -dict { PACKAGE_PIN V16      IOSTANDARD LVCMOS33 }
23     [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
24 #set_property -dict { PACKAGE_PIN T15      IOSTANDARD LVCMOS33 }
25     [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
26 #set_property -dict { PACKAGE_PIN U14      IOSTANDARD LVCMOS33 }
27     [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
28 #set_property -dict { PACKAGE_PIN T16      IOSTANDARD LVCMOS33 }
29     [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14
30     Sch=led[11]
31 set_property -dict { PACKAGE_PIN V15      IOSTANDARD LVCMOS33 }
32     [get_ports { state[0] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
33 set_property -dict { PACKAGE_PIN V14      IOSTANDARD LVCMOS33 }
34     [get_ports { state[1] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
35 set_property -dict { PACKAGE_PIN V12      IOSTANDARD LVCMOS33 }
36     [get_ports { state[2] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
37 set_property -dict { PACKAGE_PIN V11      IOSTANDARD LVCMOS33 }
38     [get_ports { state[3] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
39
40 ##Buttons
41 #set_property -dict { PACKAGE_PIN C12      IOSTANDARD LVCMOS33 }
42     [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15
43     Sch=cpu_resetn
44 set_property -dict { PACKAGE_PIN N17      IOSTANDARD LVCMOS33 }
45     [get_ports { RST }]; #IO_L9P_T1_DQS_14 Sch=btnc
46 #set_property -dict { PACKAGE_PIN M18      IOSTANDARD LVCMOS33 }
47     [get_ports { BTNU }]; #IO_L4N_T0_D05_14 Sch=btnu
48 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

```

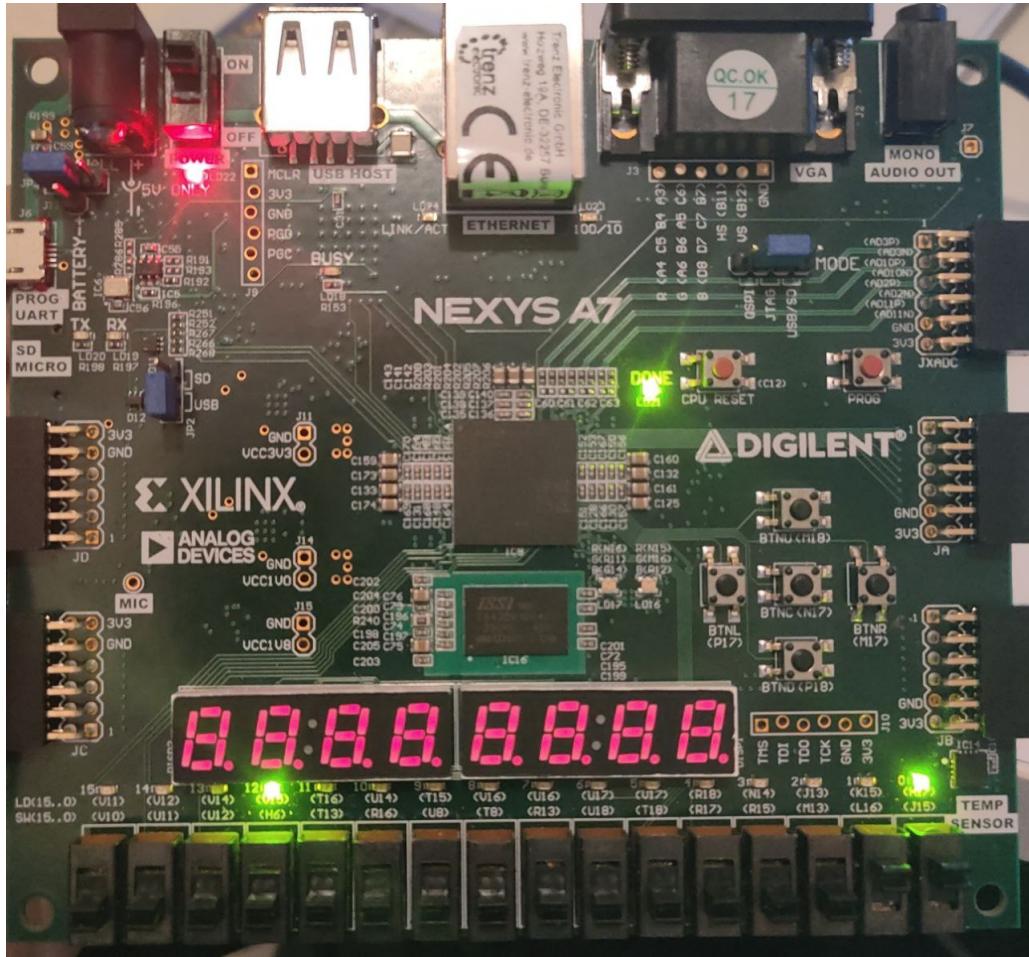


Figure 3.3: Esempio di sequenza riconosciuta

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

The screenshot shows the Vivado interface with the following details:

- Project Summary:** Shows the project structure: Riconoscitore_onBoard.vhd and Nexys-A7-50T-Master.xdc.
- Timing Analysis Window:**
 - Script pane (Tcl Console): Contains XDC constraints for clock and switches.
 - Report pane (Design Timing Summary):

	Setup	Hold	Pulse Width
Worst Negative Slack (WNS):	4,660 ns	0,212 ns	Worst Pulse Width Slack (WPWS): 4,500 ns
Total Negative Slack (TNS):	0,000 ns	0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints:	0	0	Number of Failing Endpoints: 0
Total Number of Endpoints:	215	215	Total Number of Endpoints: 82

 A note at the bottom states: "All user specified timing constraints are met."

la frequenza del clock del design ed eseguire una timing analysis finchè non si ottiene uno slack negativo (worst negative slack). Prima 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 presente 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

The top screenshot shows the 'Timing' tab in the navigation bar. The left pane displays a tree view of timing analysis categories, including 'Check Timing (10)' which is expanded to show items like 'no_clock (0)', 'constant_clock (0)', etc. The right pane shows a table titled 'no_input_delay' with columns 'Name' and 'Severity'. It lists five entries under 'Ports with no input delay (5)', all marked as 'High': M, RST, i, i_read, and m_read.

The bottom screenshot also shows the 'Timing' tab. The left pane shows a tree view with 'Intra-Clock Paths - sys_clk_pin - Setup' selected. The right pane displays a table of 10 intra-clock paths from 'deb_m/count_reg[2]/C' to various destination registers ('deb_m/count_reg[1]/R' through 'deb_m/count_reg[7]/R'). The table includes columns for Name, Slack, Levels, High Fanout, From, To, Total Delay, Logic Delay, Net Delay, and Requirement. All paths have a slack of 4.660 ns or higher, and a total delay of approximately 4.573 ns.

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, 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

```

5
6 ## Clock signal
7 set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { CLK }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
8 create_clock -add -name sys_clk_pin -period 4.00 -waveform {0 2} [get_ports {CLK}];
9
10
11 ##Switches
12 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { i }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
<

```

Name	Slack	Levels	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/i/count_reg[11]C	deb_i/i/count_reg[26]R	2.962	0.890	2.072	4.0

del clock pari a 3.5 ed una relativa waveform (0 1.75):

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.

```

5
6 ## Clock signal
7 set_property -dict { PACKAGE_PIN E3 IOSTANDARD 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 IOSTANDARD LVCMOS33 } [get_ports { i }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
<

```

Tcl Console | Messages | Log | Reports | Design Runs | DRC | Methodology | Power | **Timing** | x | ? | _ | □ | ☰ |

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-0,138 ns	Worst Hold Slack (WHS):	0,239 ns	Worst Pulse Width Slack (WPWS):	1,250 ns
Total Negative Slack (TNS):	-4,118 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	54	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

Timing constraints are not met.

Tcl Console | Messages | Log | Reports | Design Runs | DRC | Methodology | Power | **Timing** | x | ? | _ | □ | ☰ |

Intra-Clock Paths - sys_clk_pin - Setup

Name	Slack ^{^ 1}	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	S
Path 1	-0.138	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	-0.133	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	-0.132	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	-0.128	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	-0.124	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	-0.124	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	-0.124	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	-0.124	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	-0.124	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	-0.120	3	32	deb_i/count_reg[18]C	deb_i/count_reg[12]R	3.132	0.828	2.304	3.5	s

1 Path 1

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 caricare 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.

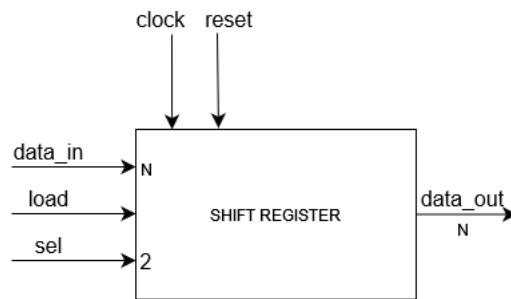


Figure 4.1: Interfaccia Shift Register da N bit

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   port(
9     clk: in std_logic;
10    reset: in std_logic;
11    load: in std_logic;
12    data_in: in std_logic_vector(N-1 downto 0);
13    sel: in std_logic_vector(1 downto 0); -- Segnale di
14      selezione per lo shift
15    data_out: out std_logic_vector(N-1 downto 0)
16  );
17
18 architecture Behavioral of shift_register is
19   signal temp_data: std_logic_vector(N-1 downto 0);
20
21 begin
22   proc: process(clk)
23   begin
24     if rising_edge(clk) then
25       if reset = '1' then
26         temp_data <= (others => '0');
27       else
28         if load='1' then
29           temp_data <= data_in;
30         else

```

```

31      case sel is
32          when "00" => -- Shift a sinistra di una
33              posizione
34                  temp_data <= temp_data(N-2 downto 0) &
35                      '0';
36          when "01" => -- Shift a destra di una
37              posizione
38                  temp_data <= '0' & temp_data(N-1
39                      downto 1);
40          when "10" => -- Shift a sinistra di due
41              posizioni
42                  temp_data <= temp_data(N-3 downto 0) &
43                      "00";
44          when "11" => -- Shift a destra di due
45              posizioni
46                  temp_data <= "00" & temp_data(N-1
47                      downto 2);
48          when others =>
49              temp_data <= temp_data; -- Nessun
50                  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.

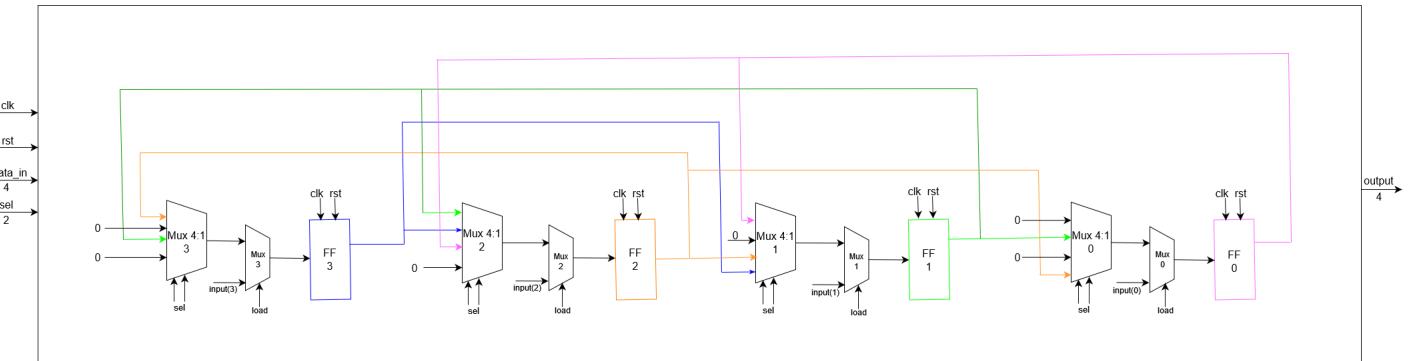


Figure 4.2: Shift Register Strutturale da N bit

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 end component;
38
39 component mux_4_1
40 port (
41     A: in std_logic_vector(3 downto 0);
42     S: in std_logic_vector(1 downto 0);
43     Y: out std_logic
44 );
45 end component;
46
47 begin
48
49 --Primo componente (N-1)
50 Mux4_Primo: mux_4_1
51     port map(
52         A(0) => out_FFD(N-2),
53         A(1) => '0',
54         A(2) => out_FFD(N-3),
55         A(3) => '0',
56         S => sel,
57         Y => out_mux4(N-1)
58 );
59
60 Mux2_Primo: mux_2_1
61     port map(

```

```

62          A(0) => out_mux4(N-1),
63          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      port map(
97          clk => clk,
98          reset => reset,
99          d => out_mux2(N-2),
100         y => out_FFD(N-2)
101     );
102
103 Mux4_intermedi: for i in N-3 downto 2 generate
104     mux: mux_4_1
105         port map(
106             A(0) => out_FFD(i-1),
107             A(1) => out_FFD(i+1),
108             A(2) => out_FFD(i-2),
109             A(3) => out_FFD(i+2),
110             S => sel,
111             Y => out_mux4(i)
112         );
113 end generate Mux4_intermedi;
114
115 Mux2_intermedi: for i in N-3 downto 2 generate
116     mux: mux_2_1
117         port map(
118             A(0) => out_mux4(i),
119             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").

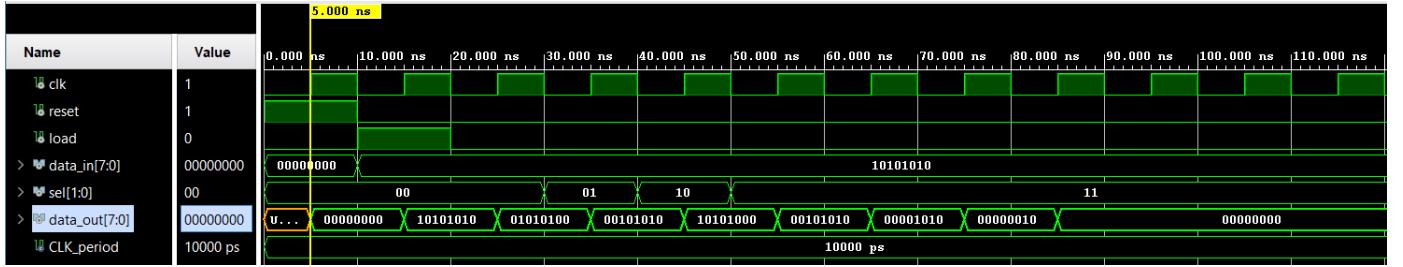


Figure 4.3: Simulazione Shift Register Comportamentale

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.

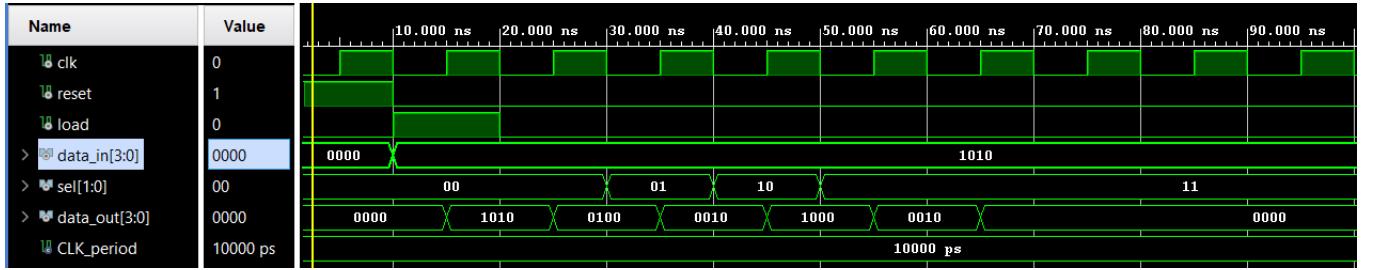


Figure 4.4: Simulazione Shift Register Strutturale

Chapter 5

Cronometro

5.1 Traccia 4.1

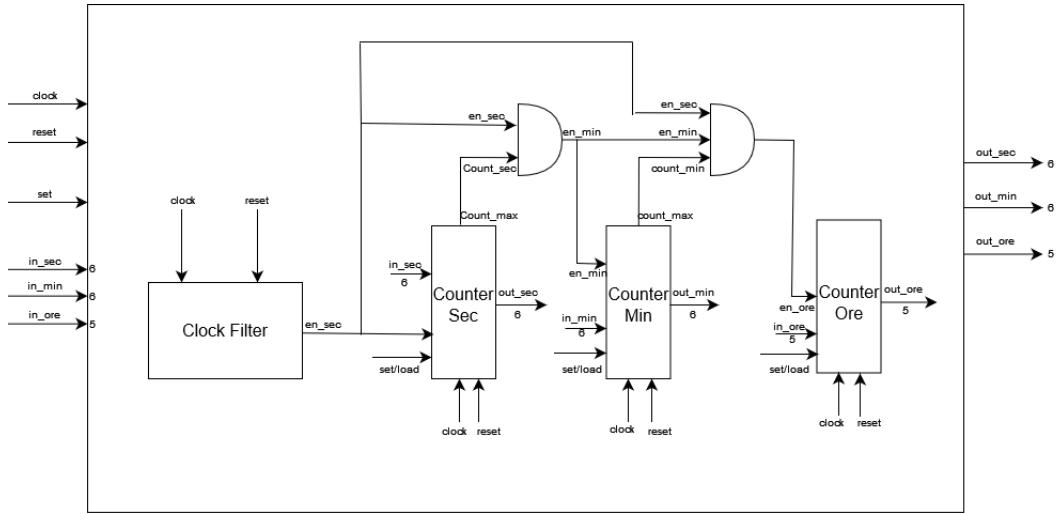
- Progettare, implementare in VHDL e testare mediante simulazione un cronometro, in grado di scandire secondi, minuti e ore a partire da una base dei tempi prefissata (es. si consideri il clock a disposizione sulla board). Il progetto deve prevedere la possibilità di inizializzare il cronometro con un 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 conteggio 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) := (others =>
24         '0');
25
26     signal temp_count: std_logic := '0';
27
28 begin
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+FFF#privato al massimo
39          valore (modulo-1)
40          if (temp_y =
41              std_logic_vector(TO_UNSIGNED(modulo-1,n))) then
42              temp_y <= (others => '0');
43          else
44              temp_y <= std_logic_vector(unsigned(temp_y) +
45                  1);
46          end if;
47      end if;
48  end process;

49
50 proc_count_max: process(clk)
51 begin
52     if rising_edge(clk) then
53         if (reset = '1') then
54             temp_count <= '0';
55         elsif (temp_y =
56             std_logic_vector(TO_UNSIGNED(modulo-1,n))) then
57             temp_count <= '1';
58         else
59             temp_count <= '0';
60         end if;
61     end if;
62 end process;
63
64 countMax <= temp_count;

```

```
63 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 board 100MHz
7         CLKOUT_freq : integer := 1           --frequenza
8             desiderata 1Hz
9     );
10    Port (
11        clock_in : in STD_LOGIC;
12        reset : in STD_LOGIC;
13        clock_out : out STD_LOGIC -- attenzione: nFH+FFFFD
14            vero clock ma un impulso che si sato come enable
15    );
16 end clock_filter;
17
18 architecture Behavioral of clock_filter is
19
20 signal clockfx : std_logic := '0';
21
22 constant count_max_value : integer := CLKIN_freq/(CLKOUT_freq)-1;
23
24 begin
25
26 clock_out <= clockfx;
27
28 count_for_division: process(clock_in)
29
30 variable counter : integer range 0 to count_max_value := 0;
31 begin

```

```

30
31     if rising_edge(clock_in) then
32         if( reset = '1') then
33             counter := 0;
34             clockfx <= '0';
35         else
36             if counter = count_max_value then
37                 clockfx <= '1';
38                 counter := 0;
39             else
40                 clockfx <= '0';
41                 counter := counter + 1;
42             end if;
43         end if;
44     end if;
45 end process;
46
47
48 end Behavioral;

```

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'istanziazione di due componenti principali: clock_filter e counter. Il primo componente genera un segnale 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 di clock in
7                           ingresso.
8     rst: in std_logic;                      -- Segnale di reset in
9                           ingresso.
10    set: in std_logic;                      -- Segnale per
11      impostare i valori iniziali.
12    in_sec: in std_logic_vector(5 downto 0); -- Valore
13      iniziale per i secondi.
14    in_min: in std_logic_vector(5 downto 0); -- Valore
15      iniziale per i minuti.
16    in_ore: in std_logic_vector(4 downto 0); -- Valore
17      iniziale per le ore.
18    out_sec: out std_logic_vector(5 downto 0); -- Output dei
19      secondi.
20    out_min: out std_logic_vector(5 downto 0); -- Output dei
21      minuti.
22    out_ore: out std_logic_vector(4 downto 0)  -- Output delle
23      ore.
24  );
25 end cronometro;
26
27
28 architecture Structural of cronometro is
29
30   -- Segnali interni per abilitare i contatori e per segnalare
31   -- il conteggio massimo.
32   signal en_sec: std_logic := '0';

```

```

22      signal en_min: std_logic := '0';
23      signal en_ore: std_logic := '0';
24      signal count_sec: std_logic := '0';
25      signal count_min: std_logic := '0';
26      signal count_ore: std_logic := '0';

27
28  component clock_filter
29
30    generic(
31      CLKIN_freq : integer := 100000000; -- Frequenza di
32                  -- clock della scheda (100MHz).
33      CLKOUT_freq : integer := 1           -- Frequenza
34                  -- desiderata di output (1Hz).
35    );
36
37    Port (
38      clock_in : in STD_LOGIC;          -- Clock in ingresso.
39      reset : in STD_LOGIC;            -- Segnale di reset.
40      clock_out : out STD_LOGIC        -- Clock ridotto in
41                  -- uscita.
42    );
43
44  end component;

45
46  component counter
47
48    generic(
49      modulo: integer := 8;
50      n: integer := 3
51    );
52
53    port(
54      clk: in std_logic;
55      reset: in std_logic;
56      enable: in std_logic;
57      load: in std_logic;
58      input: in std_logic_vector(n-1 downto 0);
59      y: out std_logic_vector(n-1 downto 0);
60      countMax: out std_logic
61    );
62

```

```

53      );
54  end component;
55
56 begin
57
58 -- Instanziazione del componente clock_filter per generare il
59 -- segnale "en_sec".
60 base_dei_tempi: clock_filter
61   generic map( 100000000, 100000000)
62   port map(
63     clock_in => clk,
64     reset => rst,
65     clock_out => en_sec
66   );
67
68 counter_secondi: counter
69   generic map(
70     60,                                -- Conta fino a 60 per
71     i secondi.
72     6                                -- Larghezza del
73     contatore di 6 bit.
74   )
75   port map(
76     clk => clk,
77     reset => rst,
78     enable => en_sec,
79     load => set,
80     input => in_sec,
81     y => out_sec,
82     countMax => count_sec
83   );
84
85 -- Abilitazione del contatore dei minuti quando i secondi
86 -- raggiungono il massimo.

```

```

83      en_min <= en_sec and count_sec;
84
85      counter_minuti: counter
86          generic map(
87              60,                                     -- Conta fino a 60 per
88              6                                         -- Larghezza del
89              contatore di 6 bit.
90          )
91          port map(
92              clk => clk,
93              reset => rst,
94              enable => en_min,
95              load => set,
96              input => in_min,
97              y => out_min,
98              countMax => count_min
99          );
100
101      -- Abilitazione del contatore delle ore quando i minuti
102      -- raggiungono il massimo.
103      en_ore <= en_sec and en_min and count_min;
104
105      counter_ore: counter
106          generic map(
107              24,                                     -- Conta fino a 24 per
108              5                                         -- Larghezza del
109              le ore.
110              contatore di 5 bit.
111          )
112          port map(
113              clk => clk,
114              reset => rst,
115              enable => en_ore,

```

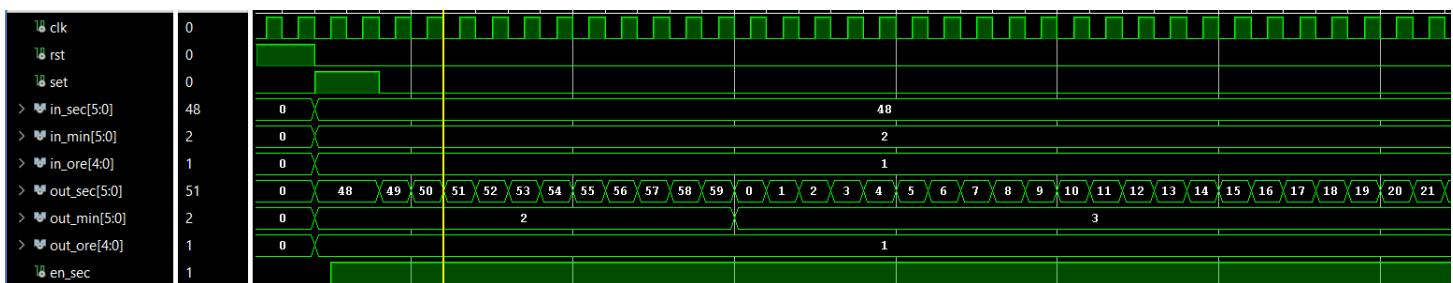
```

112      load => set,
113      input => in_ore,
114      y => out_ore,
115      countMax => count_ore
116    );
117
118 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 discendente 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

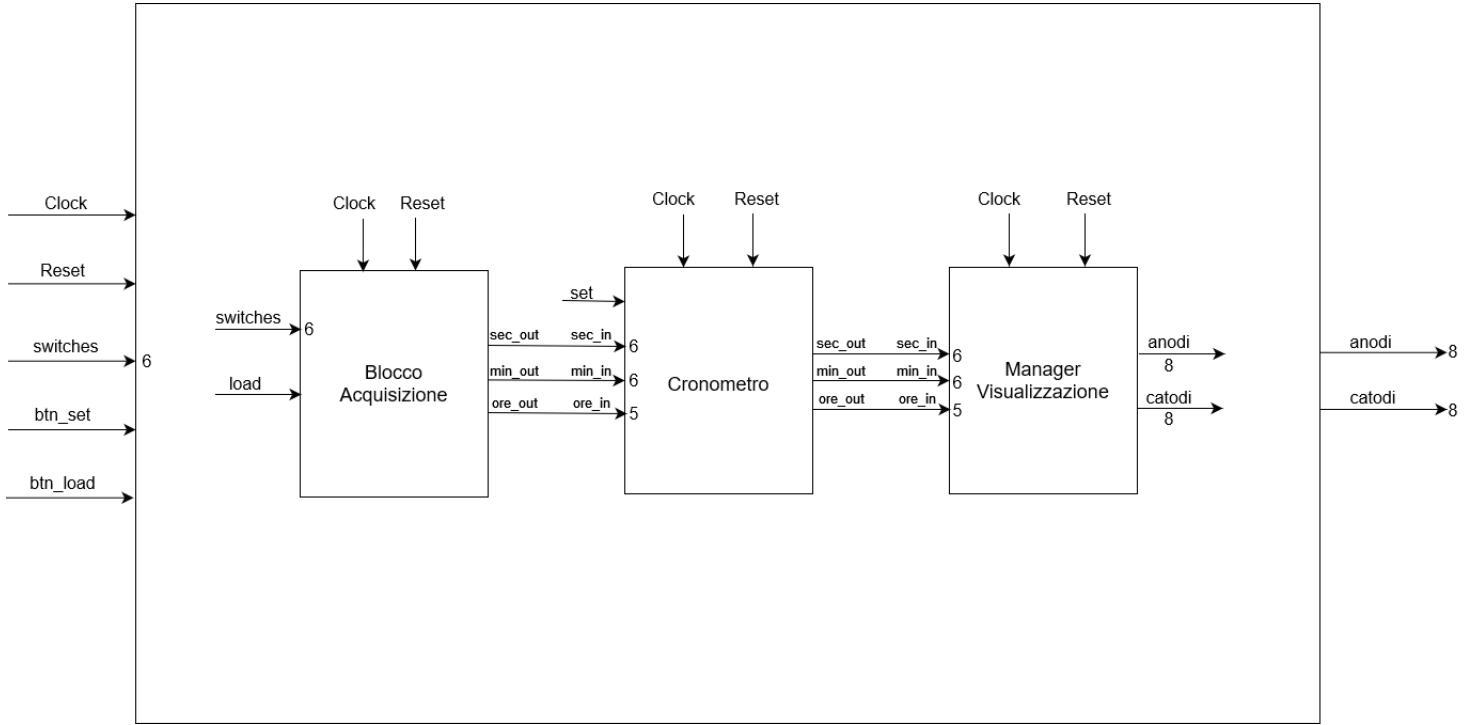
- Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando i display a 7 segmenti per la visualizzazione dell’orario (o una combinazione di display e led nel caso in cui i display a disposizione siano in numero inferiore a quello necessario), gli switch per l’immissione 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_{onBoard})

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 corrispondente è alto.

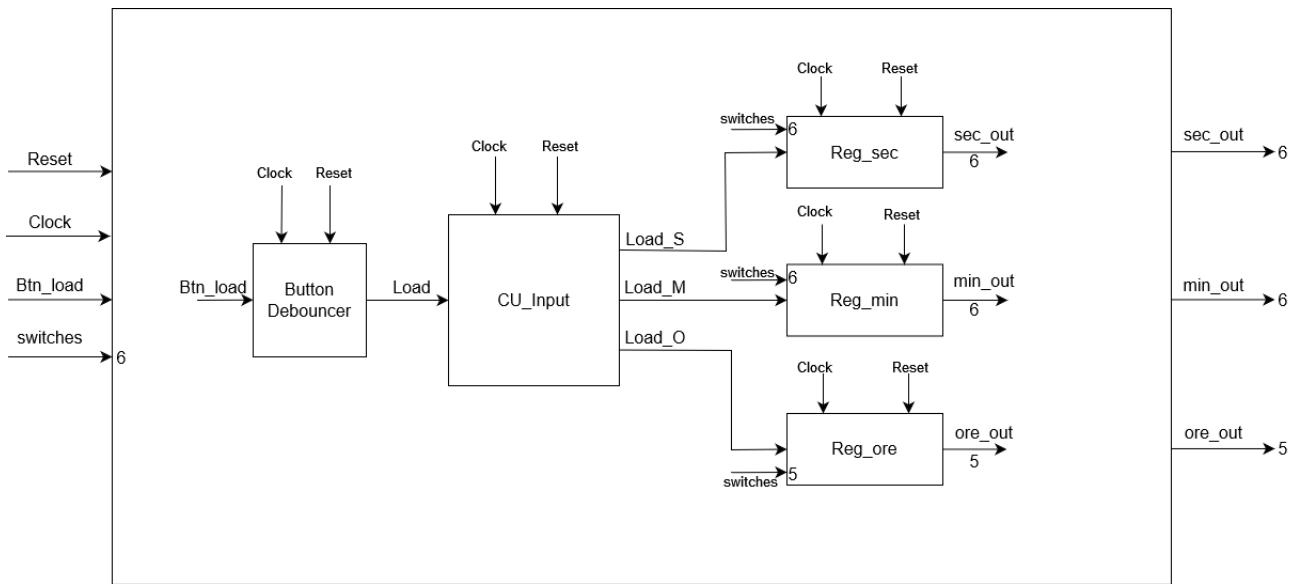


Figure 5.2: Blocco-Acquisizione

```

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 port(
20     clk: in std_logic;

```

```

21         reset: in std_logic;
22         load: in std_logic;
23         load_sec: out std_logic;
24         load_min: out std_logic;
25         load_ore: out std_logic
26     );
27 end component;
28
29 component ButtonDebouncer
30     generic (
31         CLK_period: integer := 10; -- periodo del clock in
32                     nanosec
33         btn_noise_time: integer := 10000000 --durata
34                     dell'oscillazione in nanosec
35     );
36     Port (
37         RST : in STD_LOGIC;
38         CLK : in STD_LOGIC;
39         BTN : in STD_LOGIC;
40         CLEARED_BTN : out STD_LOGIC
41     );
42 end component;
43
44 component registro_acq is
45     generic(
46         N: integer := 8 --dimensione del registro
47     );
48     port(
49         clk: in std_logic;
50         reset: in std_logic;
51         load: in std_logic; --segnale di 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     );

```

```
53      end component;  
54  
55      signal load_temp: std_logic := '0';  
56      signal load_sec_temp: std_logic := '0';  
57      signal load_min_temp: std_logic := '0';  
58      signal load_ore_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.

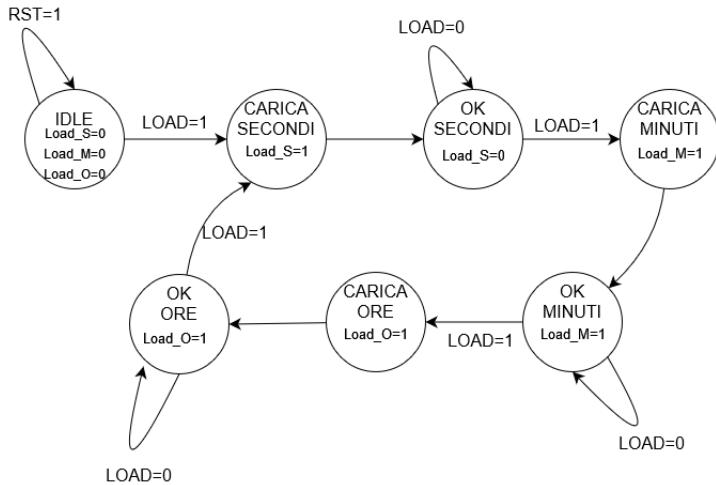


Figure 5.3: FSM CU_input

```

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,
18    ok_minuti, carica_ore, ok_ore);
19 signal stato_corrente: stato := idle;

```

```

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 <= carica_secondi;
38             else
39                 stato_corrente <= idle;
40             end if;
41         when carica_secondi =>
42             load_sec <= '1';
43             stato_corrente <= ok_secondi;
44         when ok_secondi =>
45             load_sec <= '0';
46             if(load = '1') then
47                 stato_corrente <= carica_minuti;
48             else
49                 stato_corrente <= ok_secondi;
50             end if;
51         when carica_minuti =>
52             load_min <= '1';
53             stato_corrente <= ok_minuti;

```

```

54         when ok_minuti =>
55             load_min <= '0';
56             if(load = '1') then
57                 stato_corrente <= carica_ore;
58             else
59                 stato_corrente <= ok_minuti;
60             end if;
61         when carica_ore =>
62             load_ore <= '1';
63             stato_corrente <= ok_ore;
64         when ok_ore =>
65             load_ore <= '0';
66             if(load = '1') then
67                 stato_corrente <= 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.

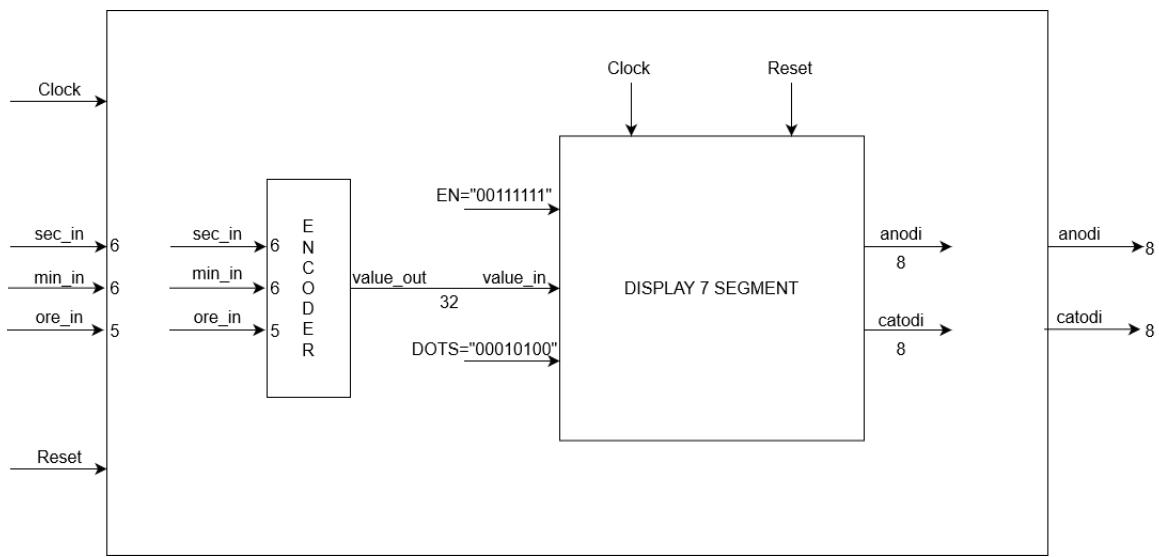


Figure 5.4: Manager_Visualizzazione

```

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 0);
35             ENABLE : in STD_LOGIC_VECTOR (7 downto 0); --
36                         decide quali cifre abilitare
37             DOTS : in STD_LOGIC_VECTOR (7 downto 0); -- decide
38                         quali punti visualizzare
39             ANODES : out STD_LOGIC_VECTOR (7 downto 0);
40             CATHODES : out STD_LOGIC_VECTOR (7 downto 0));
41 end component;
42
43 signal value_out_temp: std_logic_vector(31 downto 0) :=
44     (others => '0');
45
46 begin
47     enc: encoder
48         port map(
49             sec_in => sec_in,
50             min_in => min_in,
51             ore_in => ore_in,
52             value_out => value_out_temp
53         );

```

```

53      dis: display_seven_segments
54
55          generic map(
56              CLKIN_freq => 100000000,
57              CLKOUT_freq => 500
58          )
59
60          port map(
61              CLK => clk,
62              RST => reset,
63              VALUE => value_out_temp,
64              ENABLE => "00111111", --accendiamo tutte le cifre
65                  tranne le prime due
66              DOTS => "00010100", --accendiamo solo i punti tra ore
67                  e min e tra min e sec
68              ANODES => anodi,
69              CATHODES => catodi
70          );
71
72
73      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) := (others =>
18     '0');
19   signal min_temp : std_logic_vector (7 downto 0) := (others =>
20     '0');
21   signal ore_temp : std_logic_vector (7 downto 0) := (others =>
22     '0');
23
24 begin
25
26   value_out <= "00000000" & ore_temp & min_temp & sec_temp;
27
28   -- Prima cifra dei secondi
29   with to_integer(unsigned(sec_in)) select
30     sec_temp (3 downto 0) <= "0000" when 0|10|20|30|40|50,
31                                         "0001" when 1|11|21|31|41|51,
32                                         "0010" when 2|12|22|32|42|52,
33                                         "0011" when 3|13|23|33|43|53,
34                                         "0100" when 4|14|24|34|44|54,
                                         "0101" when 5|15|25|35|45|55,
                                         "0110" when 6|16|26|36|46|56,
                                         "0111" when 7|17|27|37|47|57,

```

```

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

```

69          "0100" when 40 to 49,
70          "0101" when 50 to 59,
71          "1111" when others;
72
73      -- Prima cifra delle ore
74      with to_integer(unsigned(ore_in)) select
75          ore_temp (3 downto 0) <= "0000" when 0|10|20,
76                      "0001" when 1|11|21,
77                      "0010" when 2|12|22,
78                      "0011" when 3|13|23,
79                      "0100" when 4|14,
80                      "0101" when 5|15,
81                      "0110" when 6|16,
82                      "0111" when 7|17,
83                      "1000" when 8|18,
84                      "1001" when 9|19,
85                      "1111" when others;
86
87      -- Seconda cifra delle ore
88      with to_integer(unsigned(ore_in)) select
89          ore_temp (7 downto 4) <= "0000" when 0 to 9,
90                      "0001" when 10 to 19,
91                      "0010" when 20 to 23,
92                      "1111" when others;
93
94  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
34          reset in ingresso.
35      set: in std_logic;                      -- Segnale per
36          impostare i valori iniziali.
37      in_sec: in std_logic_vector(5 downto 0);  -- Valore
38          iniziale per i secondi.
39      in_min: in std_logic_vector(5 downto 0);  -- Valore
40          iniziale per i minuti.
41      in_ore: in std_logic_vector(4 downto 0);  -- Valore
42          iniziale per le ore.
43
44      out_sec: out std_logic_vector(5 downto 0); -- Output
45          dei secondi.
46      out_min: out std_logic_vector(5 downto 0); -- Output
47          dei minuti.
48      out_ore: out std_logic_vector(4 downto 0)  -- Output
49          delle ore.
50
51  );
52
53 end component;

54
55
56 component manager_visualizzazione
57
58     port(
59         clk: in std_logic;
60         reset: in std_logic;
61         sec_in: in std_logic_vector(5 downto 0);
62         min_in: in std_logic_vector(5 downto 0);
63         ore_in: in std_logic_vector(4 downto 0);
64         anodi: out std_logic_vector(7 downto 0);
65         catodi: out std_logic_vector(7 downto 0)
66     );
67
68 end component;
69
70
71 signal sec_in_temp: std_logic_vector(5 downto 0) := (others =>
72             '0');

```

```

57      signal min_in_temp: std_logic_vector(5 downto 0) := (others =>
58          '0');
59      signal ore_in_temp: std_logic_vector(4 downto 0) := (others =>
60          '0');
61      signal sec_out_temp: std_logic_vector(5 downto 0) := (others
62          => '0');
63      signal min_out_temp: std_logic_vector(5 downto 0) := (others
64          => '0');
65      signal ore_out_temp: std_logic_vector(4 downto 0) := (others
66          => '0');

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     cron: cronometro port map(
80         clk => clk,
81         rst => reset,
82         set => btn_set,
83         in_sec => sec_in_temp,
84         in_min => min_in_temp,
85         in_ore => ore_in_temp,
86         out_sec => sec_out_temp,
87         out_min => min_out_temp,
88         out_ore => ore_out_temp
89     );

```

```

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 LVCMOS33 }
3             [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
4 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
5             [get_ports {clk}];
6
7 ##Switches
8 set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 }
9             [get_ports { switches[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
10 set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 }
11             [get_ports { switches[1] }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
12 set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 }
13             [get_ports { switches[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
14 set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 }
15             [get_ports { switches[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
16 set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 }
17             [get_ports { switches[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
18 set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 }
19             [get_ports { switches[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
20
21
22

```

```

13 ##7 segment display
14 set_property -dict { PACKAGE_PIN T10    IOSTANDARD LVCMOS33 }
15     [get_ports { catodi[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
16 set_property -dict { PACKAGE_PIN R10    IOSTANDARD LVCMOS33 }
17     [get_ports { catodi[1] }]; #IO_25_14 Sch=cb
18 set_property -dict { PACKAGE_PIN K16    IOSTANDARD LVCMOS33 }
19     [get_ports { catodi[2] }]; #IO_25_15 Sch=cc
20 set_property -dict { PACKAGE_PIN K13    IOSTANDARD LVCMOS33 }
21     [get_ports { catodi[3] }]; #IO_L17P_T2_A26_15 Sch=cd
22 set_property -dict { PACKAGE_PIN P15    IOSTANDARD LVCMOS33 }
23     [get_ports { catodi[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
24 set_property -dict { PACKAGE_PIN T11    IOSTANDARD LVCMOS33 }
25     [get_ports { catodi[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
26 set_property -dict { PACKAGE_PIN L18    IOSTANDARD LVCMOS33 }
27     [get_ports { catodi[6] }]; #IO_L4P_T0_D04_14 Sch=cg
28 set_property -dict { PACKAGE_PIN H15    IOSTANDARD LVCMOS33 }
29     [get_ports { catodi[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
30 set_property -dict { PACKAGE_PIN J17    IOSTANDARD LVCMOS33 }
31     [get_ports { anodi[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
32 set_property -dict { PACKAGE_PIN J18    IOSTANDARD LVCMOS33 }
33     [get_ports { anodi[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
34 set_property -dict { PACKAGE_PIN T9     IOSTANDARD LVCMOS33 }
35     [get_ports { anodi[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
36 set_property -dict { PACKAGE_PIN J14    IOSTANDARD LVCMOS33 }
37     [get_ports { anodi[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
38 set_property -dict { PACKAGE_PIN P14    IOSTANDARD LVCMOS33 }
39     [get_ports { anodi[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
40 set_property -dict { PACKAGE_PIN T14    IOSTANDARD LVCMOS33 }
41     [get_ports { anodi[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
42 set_property -dict { PACKAGE_PIN K2     IOSTANDARD LVCMOS33 }
43     [get_ports { anodi[6] }]; #IO_L23P_T3_35 Sch=an[6]
44 set_property -dict { PACKAGE_PIN U13    IOSTANDARD LVCMOS33 }
45     [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 }
33          [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15
34          Sch=cpu_resetn
35  set_property -dict { PACKAGE_PIN N17      IOSTANDARD LVCMOS33 }
36          [get_ports { reset }]; #IO_L9P_T1_DQS_14 Sch=btnc
37  #set_property -dict { PACKAGE_PIN M18      IOSTANDARD LVCMOS33 }
38          [get_ports { BTNU }]; #IO_L4N_T0_D05_14 Sch=btnu
39  set_property -dict { PACKAGE_PIN P17      IOSTANDARD LVCMOS33 }
40          [get_ports { btn_set }]; #IO_L12P_T1_MRCC_14 Sch=btnl
41  set_property -dict { PACKAGE_PIN M17      IOSTANDARD LVCMOS33 }
42          [get_ports { btn_load }]; #IO_L10N_T1_D15_14 Sch=btnr
43  #set_property -dict { PACKAGE_PIN P18      IOSTANDARD LVCMOS33 }
44          [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

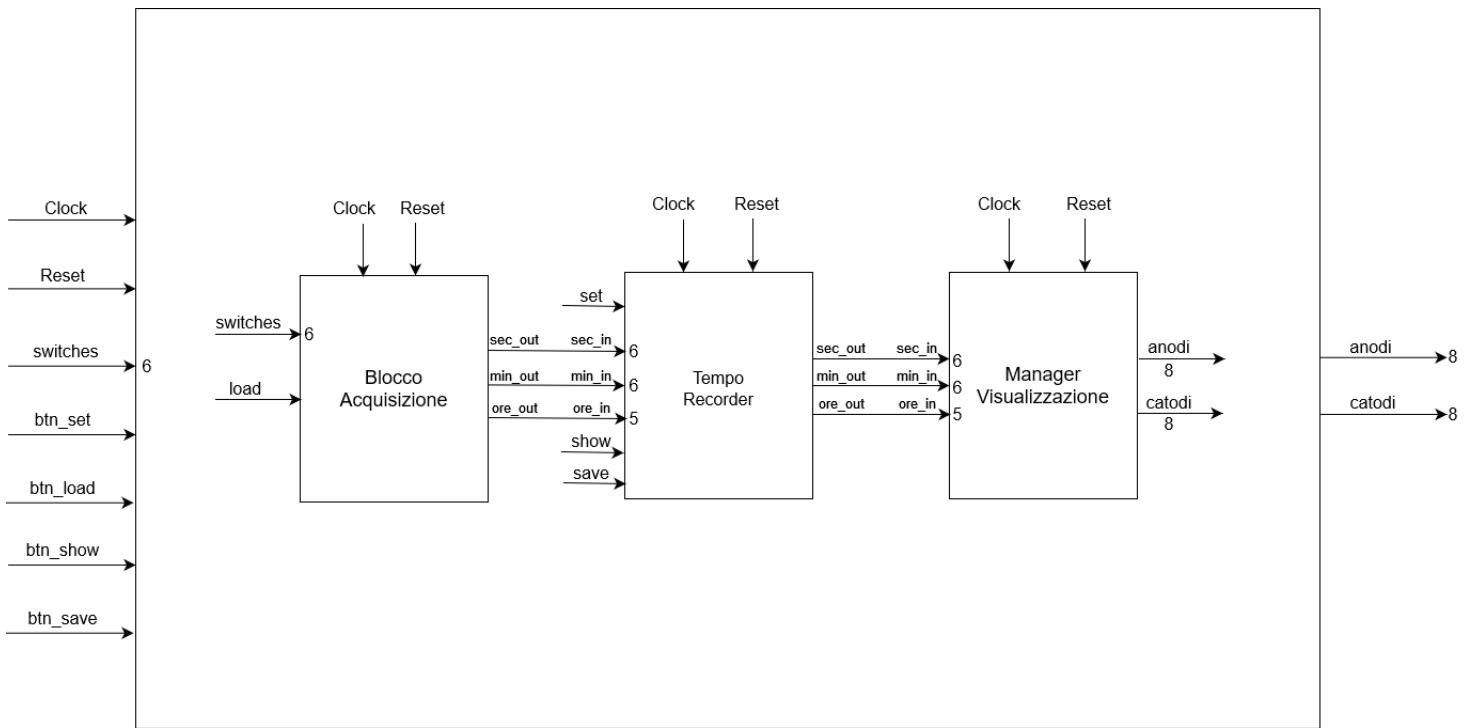
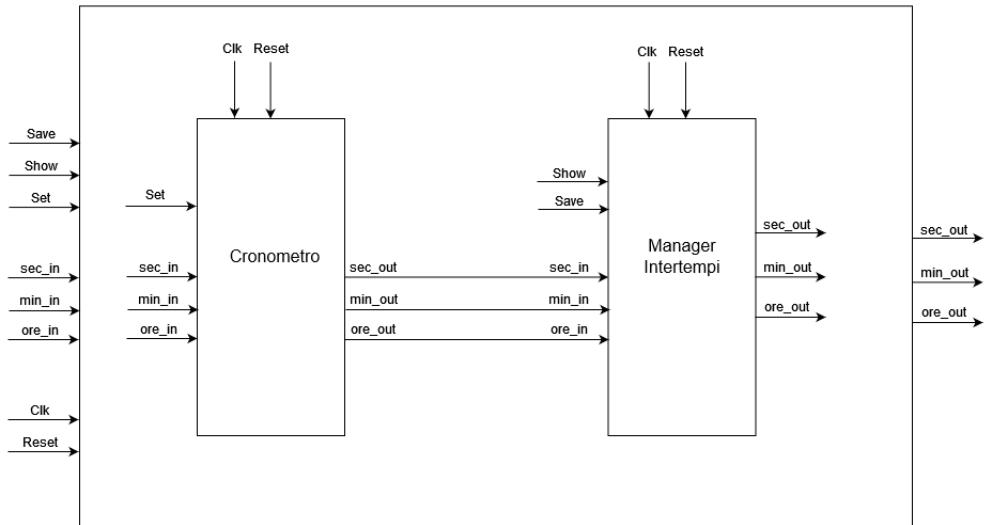

 Figure 5.5: Architettura complessiva (Cronometro_{onBoard})


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.

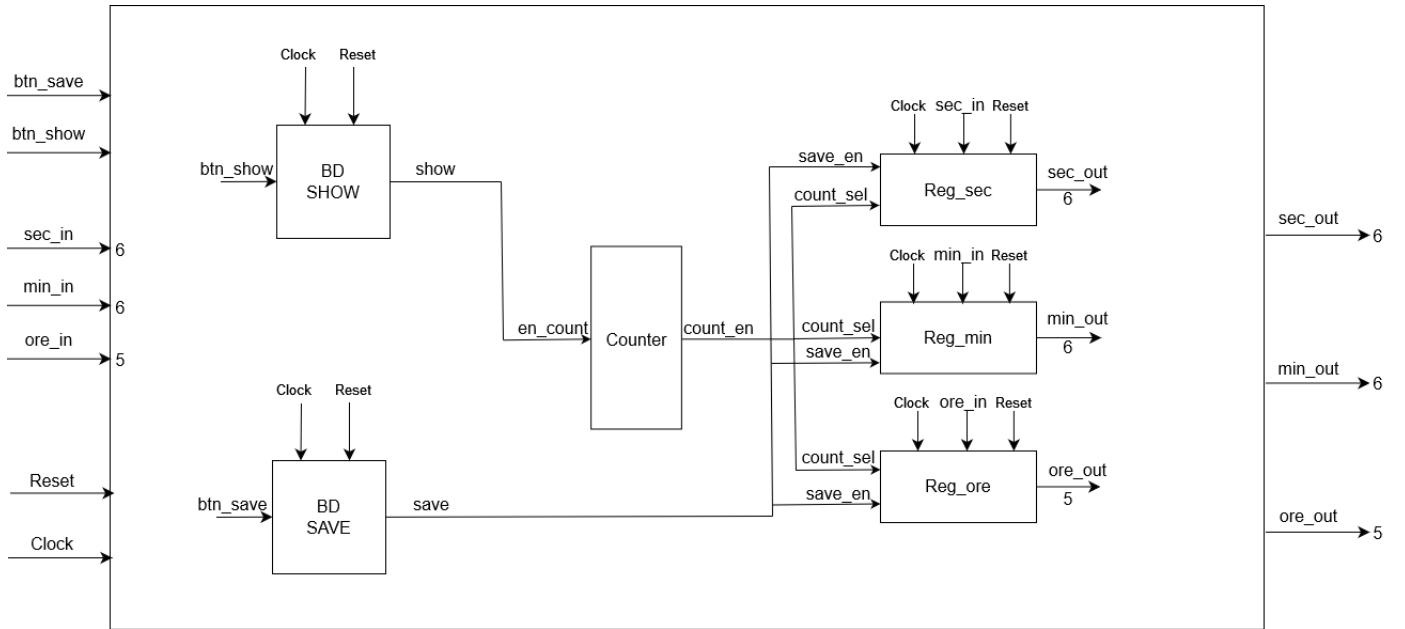


Figure 5.7: Manager_{intertempi}

Il componente registro_{_intertempi} 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_{_in} 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_{_in} ad ogni ciclo di clock.

Il segnale di uscita di ogni registro a scorrimento viene poi inviato a un multiplexer (mux_{_8_1}) 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;

```



```

56      end component;
57
58      signal show_en_temp: std_logic := '0';
59      signal save_en_temp: std_logic := '0';
60      signal sel_temp: std_logic_vector(2 downto 0) := (others =>
61          '0');
62
63 begin
64
65     deb_show: ButtonDebouncer port map(
66         RST => reset,
67         CLK => clk,
68         BTN => show,
69         CLEARED_BTN => show_en_temp
70     );
71
72     deb_save: ButtonDebouncer port map(
73         RST => reset,
74         CLK => clk,
75         BTN => save,
76         CLEARED_BTN => save_en_temp
77     );
78
79     counter: counter_mod8 port map(
80         clock => clk,
81         reset => reset,
82         enable => show_en_temp,
83         counter => sel_temp
84     );
85
86     reg_s: registro_intertempi --secondi
87         generic map(6)
88         port map(
89             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             enable : in std_logic;
27             input : in std_logic;
28             Y : out std_logic_vector(n-1 downto 0)
29         );
30     end component;
31
32     component mux_8_1
33         port(
34             data : in STD_LOGIC_VECTOR (7 downto 0);
35             sel : in STD_LOGIC_VECTOR (2 downto 0);

```

```

36         output : out STD_LOGIC
37     );
38 end component;
39
40 type out_array is array (N-1 downto 0) of std_logic_vector(6
41     downto 0);
42 signal output_temp: out_array;
43
44 begin
45 sr_n_1_to_0: for i in N-1 downto 0 generate
46
47     sr_i: shift_register
48         generic map(7)
49         port map(
50             clk => clk,
51             reset => reset,
52             enable => en,
53             input => data_in(i),
54             Y => output_temp(i)
55         );
56
57     mux_i: mux_8_1 port map(
58         data => output_temp(i) & data_in(i),
59         sel => sel,
60         output => data_out(i)
61     );
62
63 end generate;
64
65 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 port(
34     clk: in std_logic;
35     reset: in std_logic;
36     set: in std_logic;
37     save: in std_logic;
38     show: in std_logic;
39     sec_in: in std_logic_vector(5 downto 0);
40     min_in: in std_logic_vector(5 downto 0);
41     ore_in: in std_logic_vector(4 downto 0);
42     sec_out: out std_logic_vector(5 downto 0);
43     min_out: out std_logic_vector(5 downto 0);
44     ore_out: out std_logic_vector(4 downto 0)
45 );
46 end component;
47
48 component manager_visualizzazione
49 port(
50     clk: in std_logic;
51     reset: in std_logic;
52     sec_in: in std_logic_vector(5 downto 0);
53     min_in: in std_logic_vector(5 downto 0);
54     ore_in: in std_logic_vector(4 downto 0);
55     anodi: out std_logic_vector(7 downto 0);
56     catodi: out std_logic_vector(7 downto 0)
57 );
58 end component;
59
60 signal sec_in_temp: std_logic_vector(5 downto 0) := (others =>

```

```

'0');

61 signal min_in_temp: std_logic_vector(5 downto 0) := (others =>
'0');

62 signal ore_in_temp: std_logic_vector(4 downto 0) := (others =>
'0');

63 signal sec_out_temp: std_logic_vector(5 downto 0) := (others =>
'0');

64 signal min_out_temp: std_logic_vector(5 downto 0) := (others =>
'0');

65 signal ore_out_temp: std_logic_vector(4 downto 0) := (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     set => btn_set,
83     save => btn_save,
84     show => btn_show,
85     sec_in => sec_in_temp,
86     min_in => min_in_temp,
87     ore_in => ore_in_temp,
88     sec_out => sec_out_temp,

```

```

89      min_out => min_out_temp,
90      ore_out => ore_out_temp
91  );
92
93  mv: manager_visualizzazione port map(
94      clk => clk,
95      reset => reset,
96      sec_in => sec_out_temp,
97      min_in => min_out_temp,
98      ore_in => ore_out_temp,
99      anodi => anodi,
100     catodi => catodi
101 );
102
103
104 end Structural;

```

```

1    ##Buttons
2    #set_property -dict { PACKAGE_PIN C12      IOSTANDARD LVCMOS33 }
3    [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15
4    Sch=cpu_resetn
5
6    set_property -dict { PACKAGE_PIN N17      IOSTANDARD LVCMOS33 }
7    [get_ports { reset }]; #IO_L9P_T1_DQS_14 Sch=btnc
8
9    set_property -dict { PACKAGE_PIN M18      IOSTANDARD LVCMOS33 }
10   [get_ports { btn_show }]; #IO_L4N_T0_D05_14 Sch=btnu
11
12  set_property -dict { PACKAGE_PIN P17      IOSTANDARD LVCMOS33 }
13  [get_ports { btn_load }]; #IO_L12P_T1_MRCC_14 Sch=btnl
14
15  set_property -dict { PACKAGE_PIN M17      IOSTANDARD LVCMOS33 }
16  [get_ports { btn_set }]; #IO_L10N_T1_D15_14 Sch=btnr
17
18  set_property -dict { PACKAGE_PIN P18      IOSTANDARD LVCMOS33 }
19  [get_ports { btn_save }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd

```

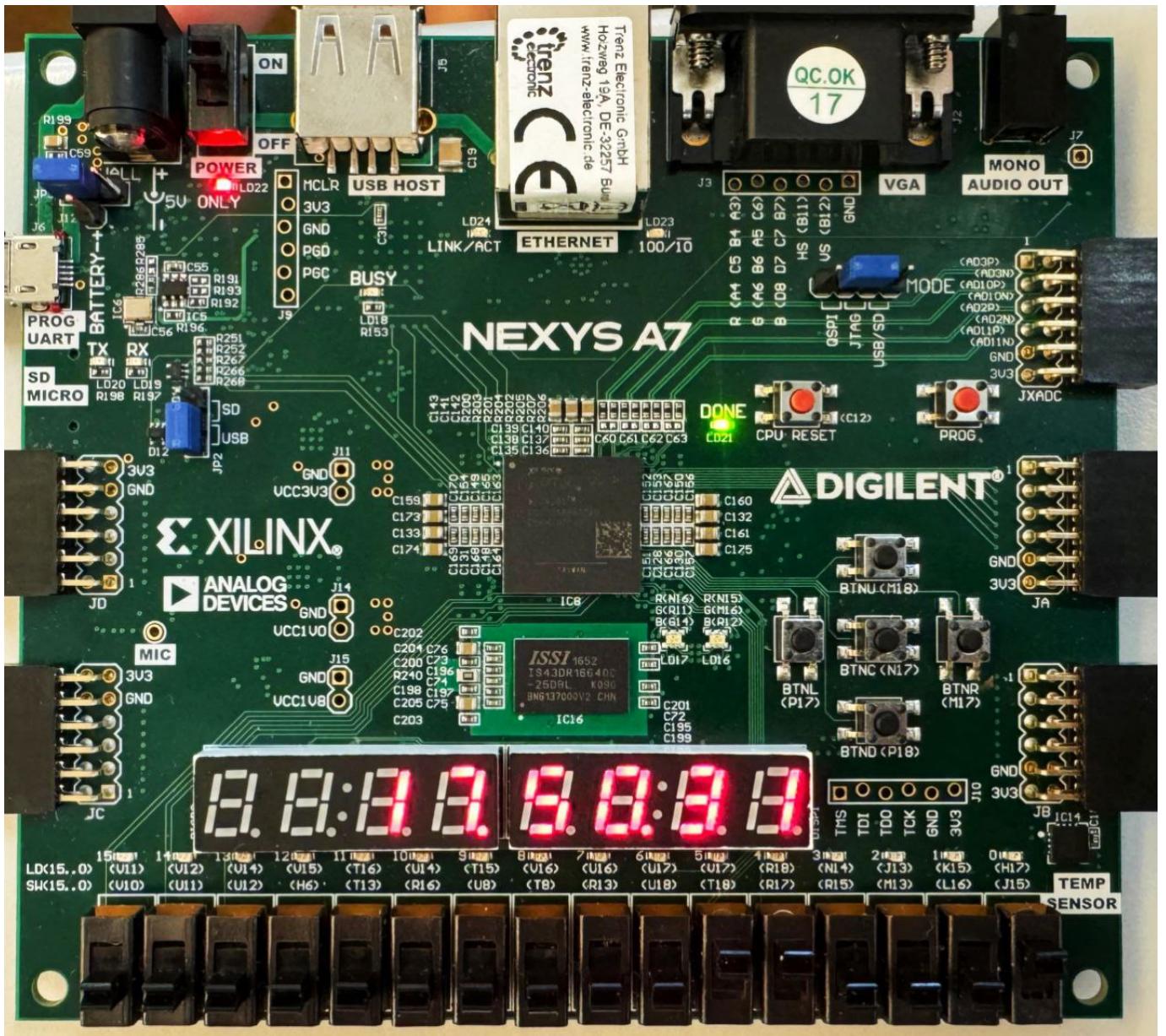


Figure 5.8: Implementazione su board

Chapter 6

Sistema PO_PC

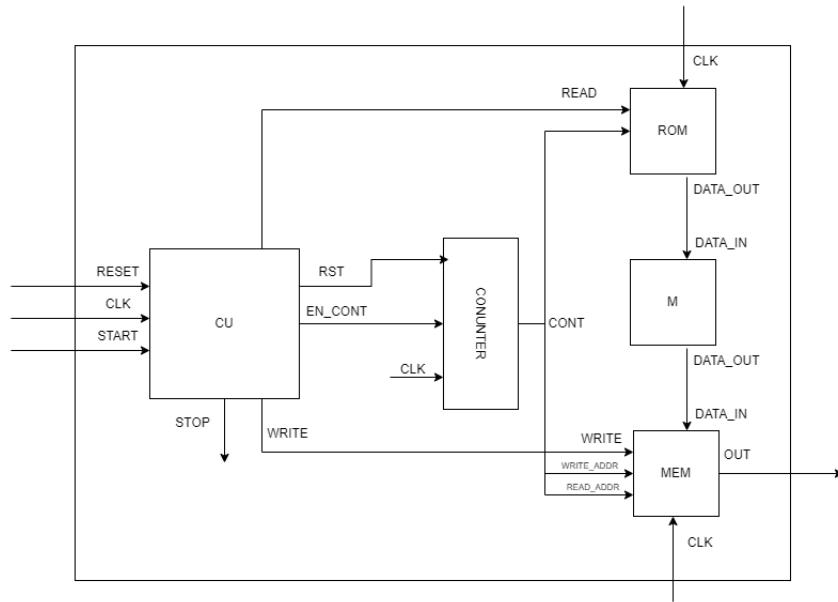
6.1 Traccia

- 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.

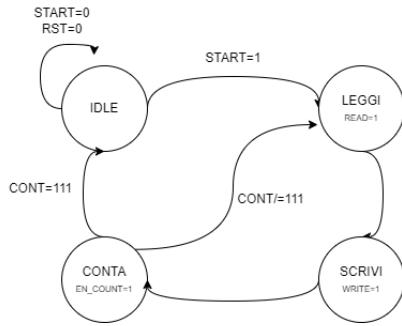
6.2 Progetto e architettura

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(
28     clk => clk,
29     start => start,
30     reset => reset,
31     count => count_temp,
32     en_cont => en_cont_temp,
33     rst_cont => rst_cont_temp,
34     read => read_temp,
35     write => write_temp,
36     stop => stop
37   );
38
39   contatore: entity work.counter_mod8 port map(
40     clk => clk,
41     reset => rst_cont_temp,
42     enable => en_cont_temp,
43     count => count_temp

```

```

44 );
45
46 ROM: entity work.rom port map(
47     clk => clk,
48     read => read_temp,
49     addr => count_temp,
50     data_out => in_m_temp
51 );
52
53 M: entity work.macchinaM port map(
54     data_in => in_m_temp,
55     data_out => out_m_temp
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); -- Stringa di
7             8 bit in ingresso

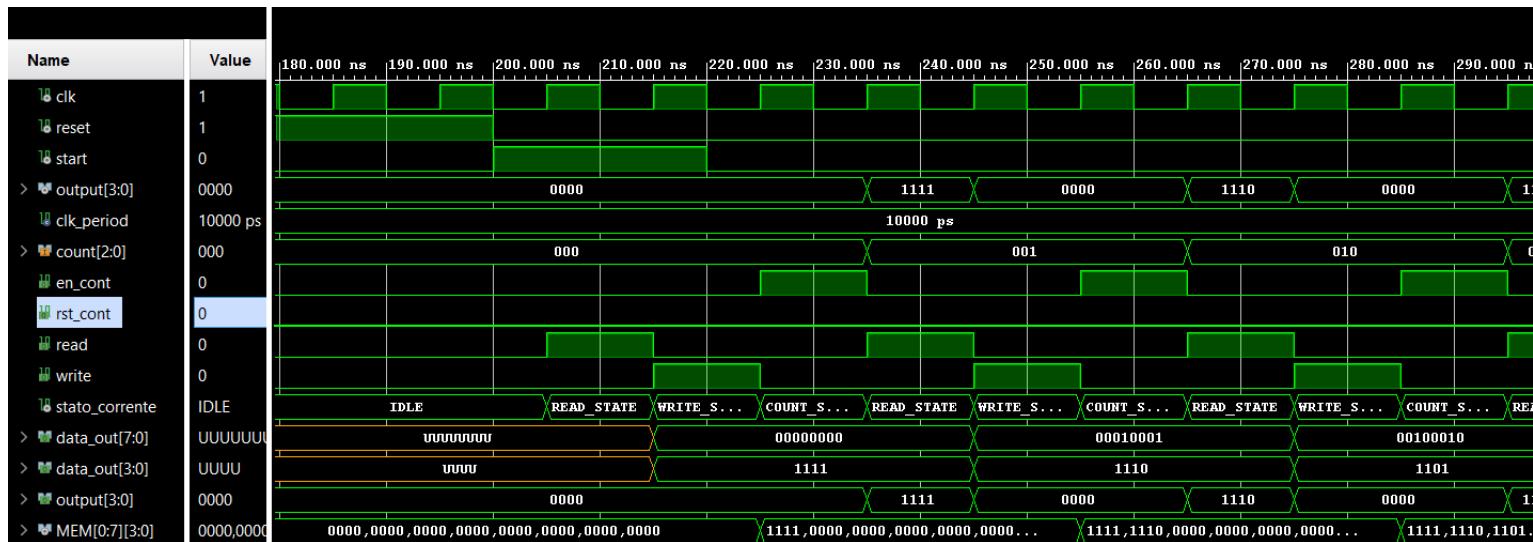
```

```

7      data_out : out STD_LOGIC_VECTOR (3 downto 0) -- Stringa
8          di 4 bit in uscita
9      );
10
11 end macchinaM;
12
13 architecture Dataflow of macchinaM is
14
15 begin
16     -- Effettua il negato bit a bit della stringa 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;
```

```

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     generic(
18       CLK_period: integer := 10;
19       btn_noise_time: integer := 10000000
20     );
21     port(
22       RST: in std_logic;
23       CLK: in std_logic;
24       BTN: in std_logic;
25       CLEARED_BTN: out std_logic -- segnale di output
26         ripulito
27     );
28   end component;
29
30   component sistemaPO_PC is
31     port(
32       clk: in std_logic;
33       reset: in std_logic;
34       start: in std_logic;
35       output: out std_logic_vector(3 downto 0);
36       stop: out std_logic
37     );
38   end component;
39
40   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 LVCMOS33 }
3           [get_ports { clock }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
4 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
5           [get_ports {clock}];
6
7 ## LEDs
8 set_property -dict { PACKAGE_PIN H17      IO_STANDARD LVCMOS33 }
9           [get_ports { Y[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
10 set_property -dict { PACKAGE_PIN K15     IO_STANDARD LVCMOS33 }

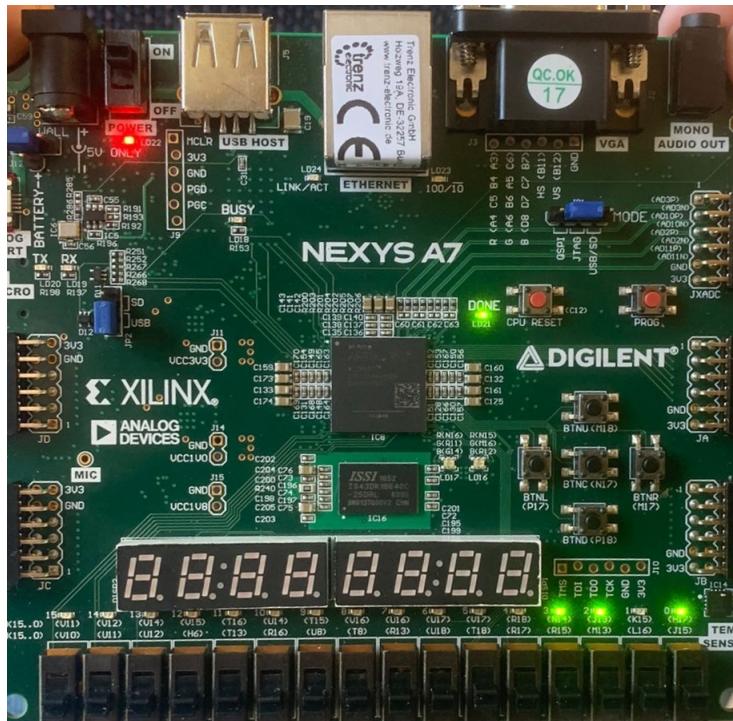
```

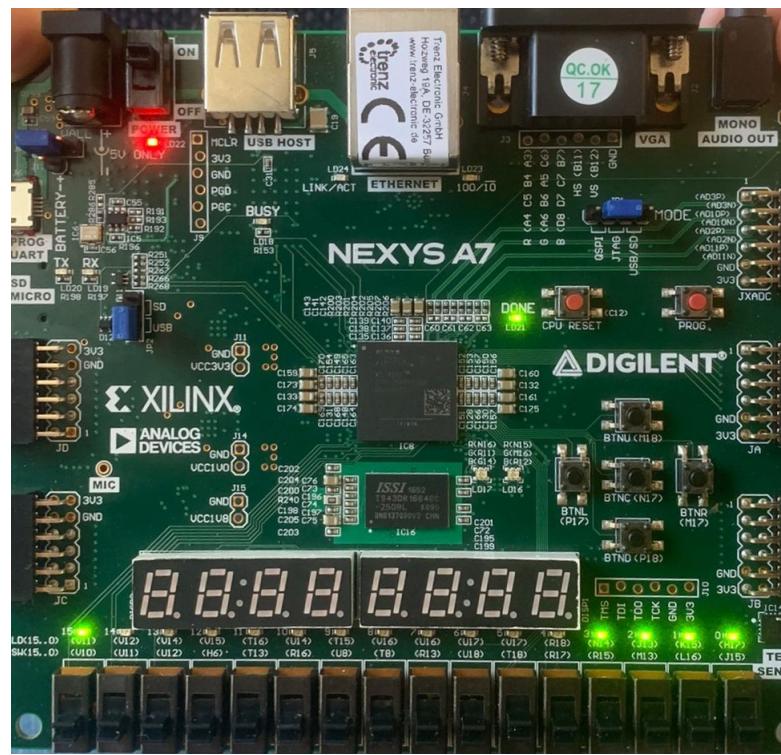
```

    [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

```

```
25 Sch=cpu_resetn  
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

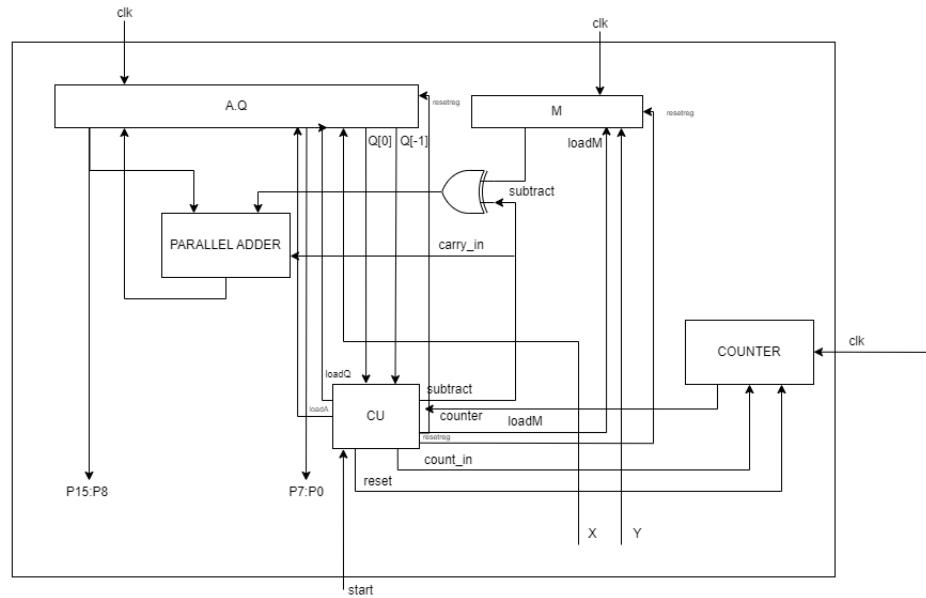
7.1 Traccia

- 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.

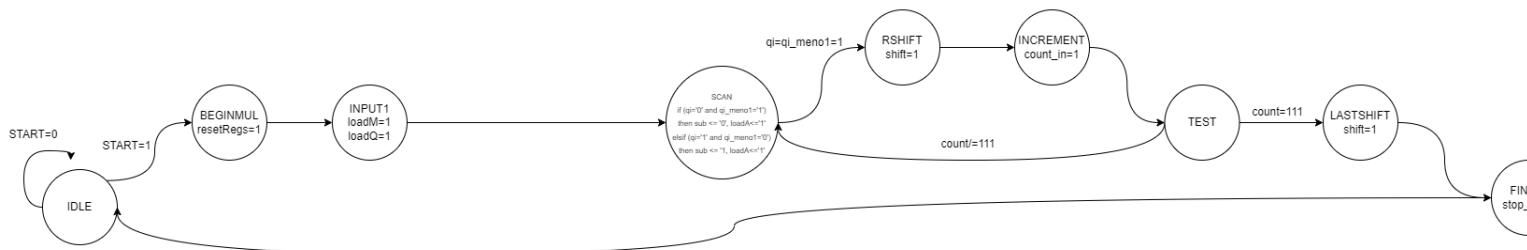


Il sistema del moltiplicatore di Booth segue un preciso flusso di esecuzione, guidato da una sequenza di stati che definiscono le varie fasi dell'operazione di moltiplicazione binaria.

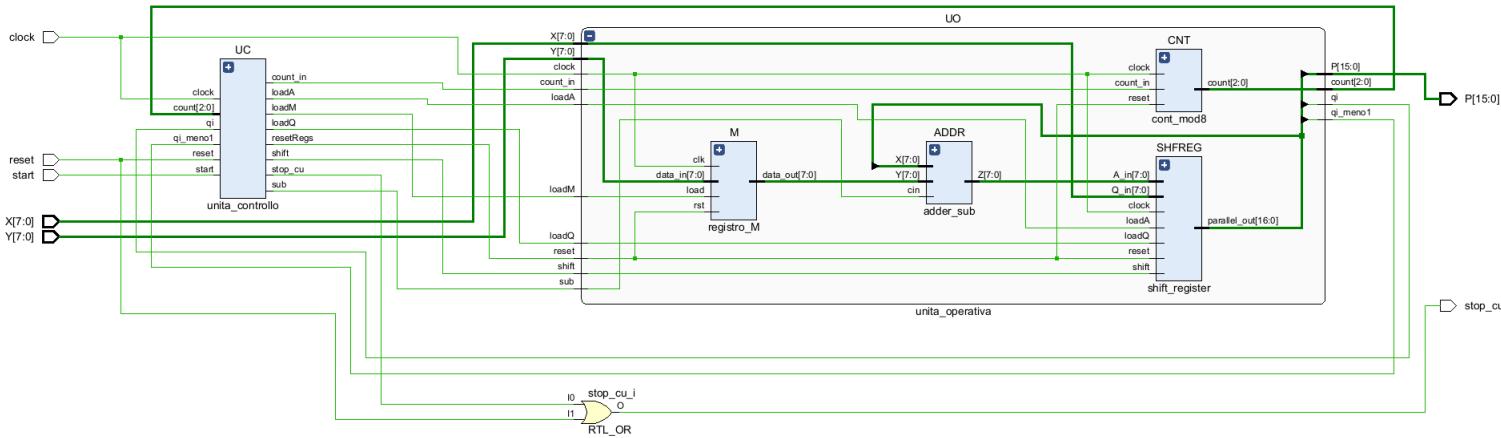
1. Stato **IDLE**: Questo è lo stato iniziale del sistema, in cui il moltiplicatore si trova in attesa del segnale di start per avviare l'operazione di moltiplicazione.
2. Stato **beginMul**: Quando il segnale di start viene rilevato, il sistema passa a questo stato. Qui, vengono azzerati lo shift register 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 moltiplicatore, mentre il registro M contiene il moltiplicando.

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
31 end component;
32
33 component unita_operativa is
34     port (
35         X, Y: in std_logic_vector (7 downto 0);
36         clock, reset: in std_logic;
37         loadQ, loadA, loadM: in std_logic;
38         shift: in std_logic;
39         sub: in std_logic;
40         count_in: in std_logic;
41         count: out std_logic_vector (2 downto 0);
42         P: out std_logic_vector(15 downto 0);
43         qi, qi_menol: out std_logic
44
45  );
46
47 end component;
48
49
50
51
52
53

```

```

54      signal tmp_count_in, tmp_stop_cu: std_logic;
55
56 begin
57
58   UC: unita_controllo port map(tmp_qi,
59     tmp_qi_menol,clock,reset,start,tmp_count,tmp_loadA,tmp_loadM,tmp_loadQ,tmp_
60   tmp_stop_cu);
61
62   UO: unita_operativa port
63     map(X,Y,clock,tmp_resetRegs,tmp_loadQ,tmp_loadA,tmp_loadM,tmp_
64     shift,tmp_s
65
66   stop_cu <= tmp_stop_cu or reset;
67
68 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_menol: 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,finish);
23
24      signal current_state, next_state: state;
25
26  begin
27
28      reg_stato: process(clock)
29          begin
30              if(clock'event and clock='1') then
31                  if(reset='1') then
32                      current_state <= idle;
33                  else
34                      current_state <= next_state;
35                  end if;
36              end if;
37          end process;
38
39
40      comb: process(current_state,start,count)
41          begin
42              loadA <= '0';
43              loadM <= '0';
44              loadQ <= '0';
45              shift <= '0';
46              sub <= '0';
47              count_in <= '0';
48              stop_cu <= '0';
49              resetRegs <= '0';
50
51
52              CASE current_state is
53                  WHEN idle =>
54                      if(start='1') then
55                          next_state <= beginMul;

```

```

52         else
53             next_state <= idle;
54         end if;
55
56         WHEN beginMul => -- Azzero i registri della UO
57             resetRegs <= '1';
58             next_state <= input;
59
60
61         WHEN input =>
62             loadM <= '1'; -- carico gli operandi
63             loadQ <= '1';
64             next_state <= scan;
65
66
67         WHEN scan =>
68             if(qi=qi_menol) then
69                 next_state <= rshift;
70             else -- scelgo cosa fare e carico A con il
71                 risultato
72                 if(qi='0' and qi_menol='1') then
73                     sub <= '0';
74                     loadA<='1';
75                 elsif (qi='1' and qi_menol='0') then
76                     sub <= '1';
77                     loadA<='1';
78                 end if;
79                 next_state <= rshift;
80             end if;
81
82         WHEN rshift =>
83             shift <= '1';
84             next_state <= increment;
85
86         WHEN increment =>
87             count_in<='1';
88             next_state <= test;
89
90         WHEN test => -- testo il conteggio
91             if(count="111") then

```

```

85           next_state<=lastshf;
86
87           else
88
89               next_state <= scan;
90
91               end if;
92
93               WHEN lastshf =>
94
95                   shift <='1';
96
97                   next_state <= finish;
98
99               WHEN finish =>
100
101
102           stop_cu <= '1';
103
104           next_state <= idle;
105
106
107       end CASE;
108
109
110   end process;
111
112
113 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
7     port(
8         X, Y: in std_logic_vector (7 downto 0);
9         clock, reset: in std_logic;
10        loadQ, loadA, loadM: in std_logic;
11        shift: in std_logic;
12        sub: in std_logic;
13        count_in: in std_logic;
14        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 abilitazione per
36     il contatore
37     count: out std_logic_vector(2 downto 0) --uscita 3 bit
38     che rappresenta il valore 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 std_logic; --
47     Segnali di clock, reset, caricamento e shift.

```

```

45         parallel_out: out std_logic_vector(16 downto 0)
46     );
47 end component;
48
49 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 end component;
58
59 --signal adder_to_acc: std_logic_vector (7 downto 0);
60 signal tmp_A: std_logic_vector (7 downto 0);
61 signal out_M: std_logic_vector (7 downto 0);
62 signal tmp_cout: std_logic;
63 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); -- OK
72
73     CNT: cont_mod8 port map(clock,reset,count_in,count); -- OK
74
75     tmp_A <= tmp_P(16 downto 9);
76
77     ADDR: adder_sub port map(tmp_A,out_M,sub,tmp_Z,tmp_cout);
78

```

```

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 abilitazione per
36         il contatore
37     count: out std_logic_vector(2 downto 0) --uscita 3 bit
38         che rappresenta il valore 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 std_logic; --
47         Segnali di clock, reset, caricamento e shift.
48     parallel_out: out std_logic_vector(16 downto 0)
49 );
50 end component;
51
52 component registro_M is
53 port(
54     clk : in std_logic;
55     rst : in std_logic;
56     load : in std_logic;
57     data_in : in std_logic_vector(7 downto 0);
58     data_out : out std_logic_vector(7 downto 0)
59 );
60

```

```

57      end component;

58

59      --signal adder_to_acc: std_logic_vector (7 downto 0);
60      signal tmp_A: std_logic_vector (7 downto 0);
61      signal out_M: std_logic_vector (7 downto 0);
62      signal tmp_cout: std_logic;
63      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); -- OK

72

73     CNT: cont_mod8 port map(clock,reset,count_in,count); -- OK

74

75     tmp_A <= tmp_P(16 downto 9);

76

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

78

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

81

82

83 end Structural;

```

7.3.5 Registro M

```

1 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

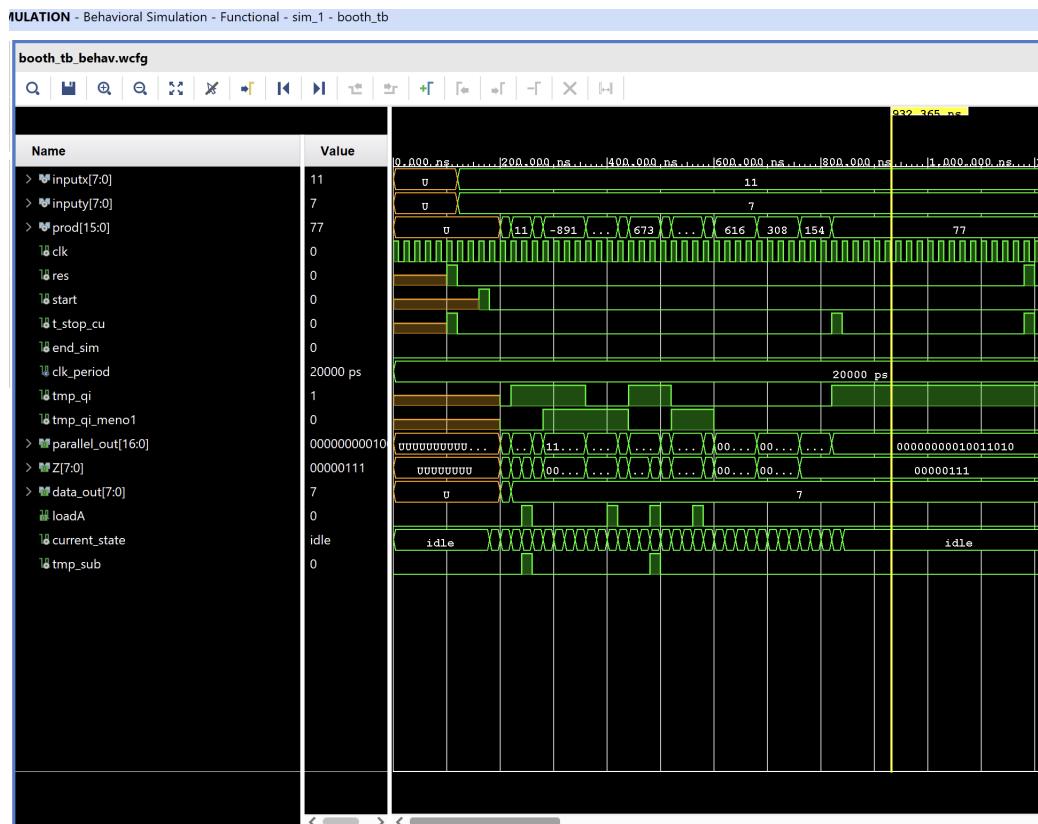
```

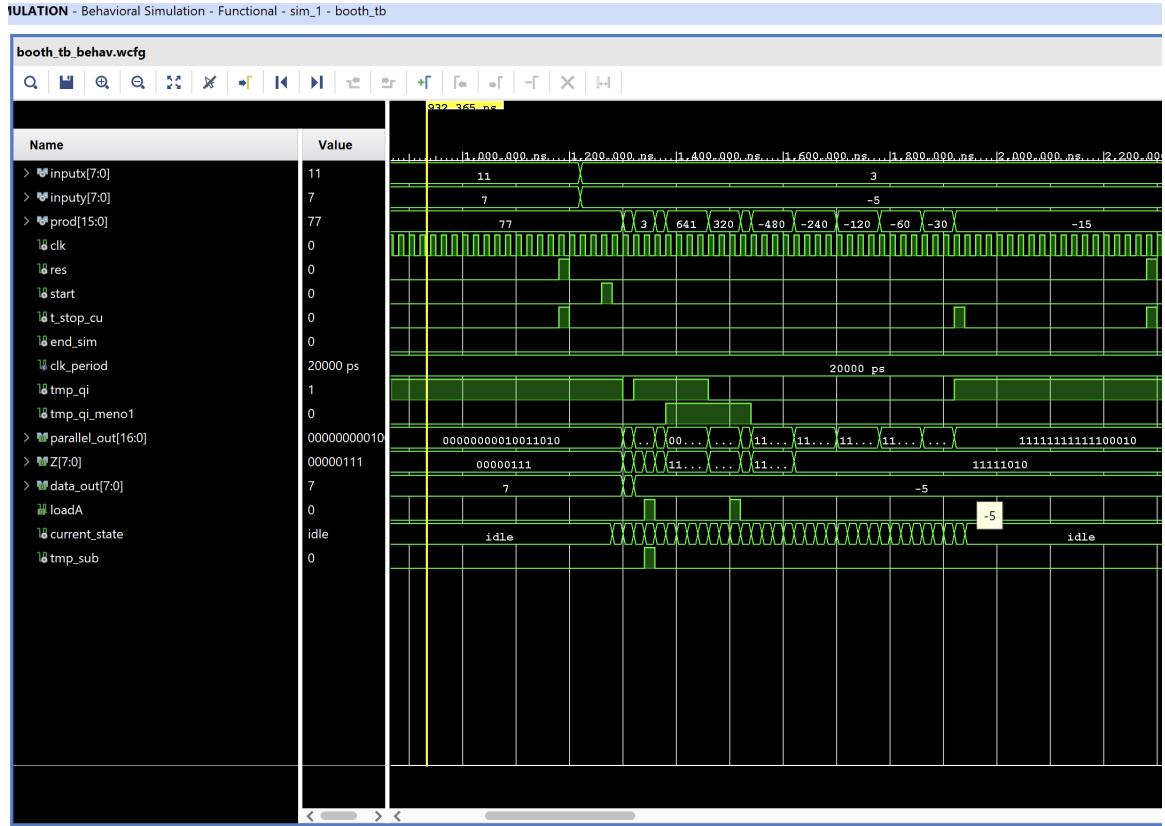


```

32      -- Se cin = '1' (sottrazione), Y viene complementato.
33      complemento_y: FOR i IN 0 TO 7 GENERATE
34          complementoy(i) <= Y(i) xor cin; -- Operazione xor per
35          complementare Y se necessario.
36
37      -- Istanza del componente ripple_carry
38      RA: ripple_carry port map(X, complementoy, cin, cout, Z);
39
40 end Structural;

```





7.4 Implementazione su Board

Nell'implementazione sulla scheda del moltiplicatore, abbiamo introdotto un button de-bouncer 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 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

6 ##Switches
7 set_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 }
22             [get_ports { Y[6] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
23 set_property -dict { PACKAGE_PIN V10    IOSTANDARD LVCMOS33 }
24             [get_ports { Y[7] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
25
26 ## LEDs
27 set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 }
28             [get_ports { P[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
29 set_property -dict { PACKAGE_PIN K15    IOSTANDARD LVCMOS33 }
30             [get_ports { P[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
31 set_property -dict { PACKAGE_PIN J13    IOSTANDARD LVCMOS33 }
32             [get_ports { P[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
33 set_property -dict { PACKAGE_PIN N14    IOSTANDARD LVCMOS33 }
34             [get_ports { P[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
35 set_property -dict { PACKAGE_PIN R18    IOSTANDARD LVCMOS33 }
36             [get_ports { P[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
37 set_property -dict { PACKAGE_PIN V17    IOSTANDARD LVCMOS33 }
38             [get_ports { P[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
39 set_property -dict { PACKAGE_PIN U17    IOSTANDARD LVCMOS33 }
40             [get_ports { P[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
41 set_property -dict { PACKAGE_PIN U16    IOSTANDARD LVCMOS33 }
42             [get_ports { P[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
43 set_property -dict { PACKAGE_PIN V16    IOSTANDARD LVCMOS33 }
44             [get_ports { P[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
45 set_property -dict { PACKAGE_PIN T15    IOSTANDARD LVCMOS33 }
46             [get_ports { P[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
47 set_property -dict { PACKAGE_PIN U14    IOSTANDARD LVCMOS33 }
48             [get_ports { P[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
49 set_property -dict { PACKAGE_PIN T16    IOSTANDARD LVCMOS33 }
50             [get_ports { P[11] }]; #IO_L15N_T2_DQS_DOUT_CS0_B_14 Sch=led[11]
51 set_property -dict { PACKAGE_PIN V15    IOSTANDARD LVCMOS33 }
52             [get_ports { P[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
53 set_property -dict { PACKAGE_PIN V14    IOSTANDARD LVCMOS33 }
54             [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=btnr
```

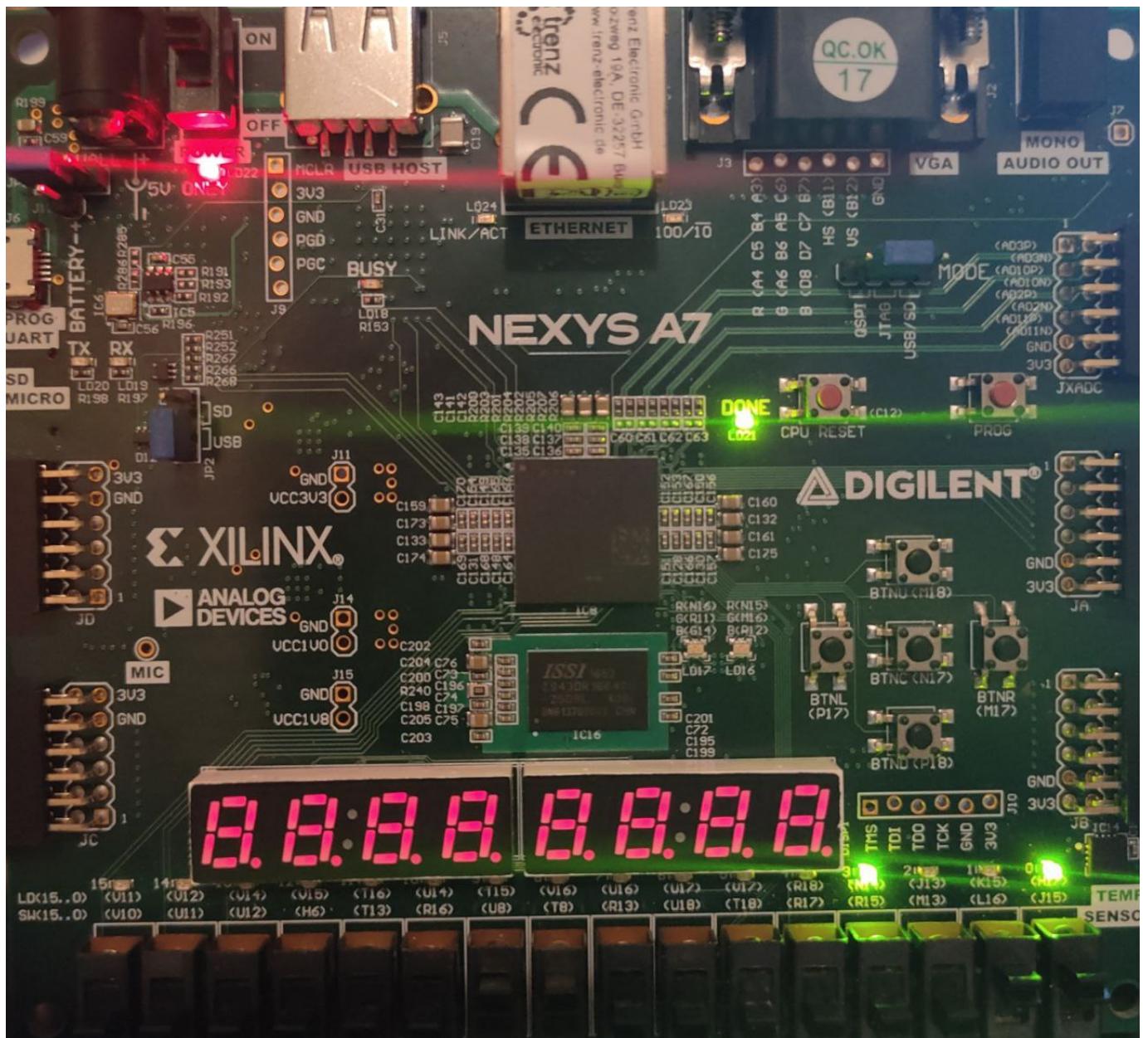


Figure 7.1: Esempio di moltiplicazione (3*3)

Chapter 8

Handshaking

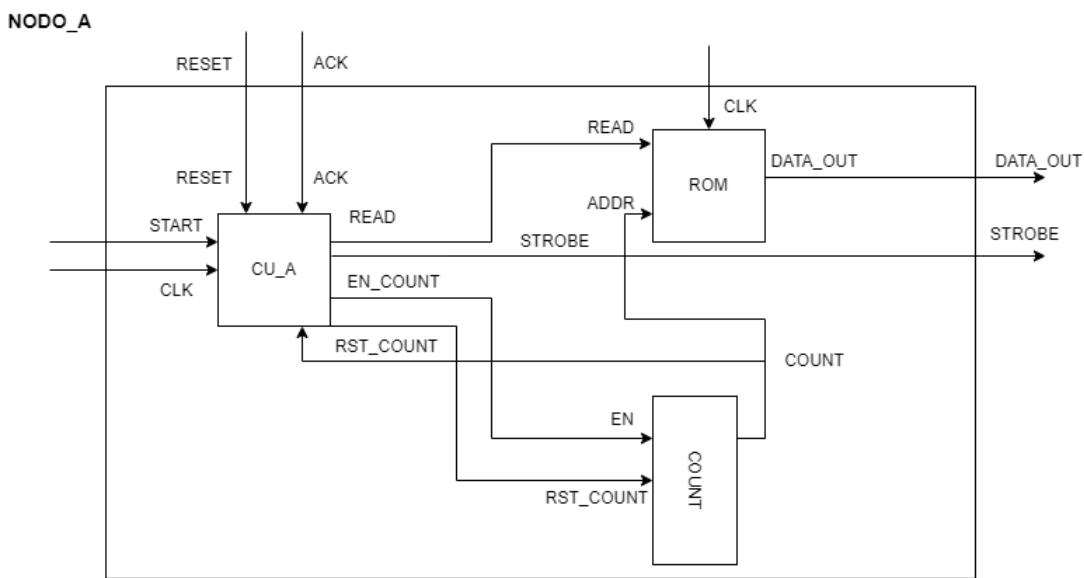
8.1 Traccia

- 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

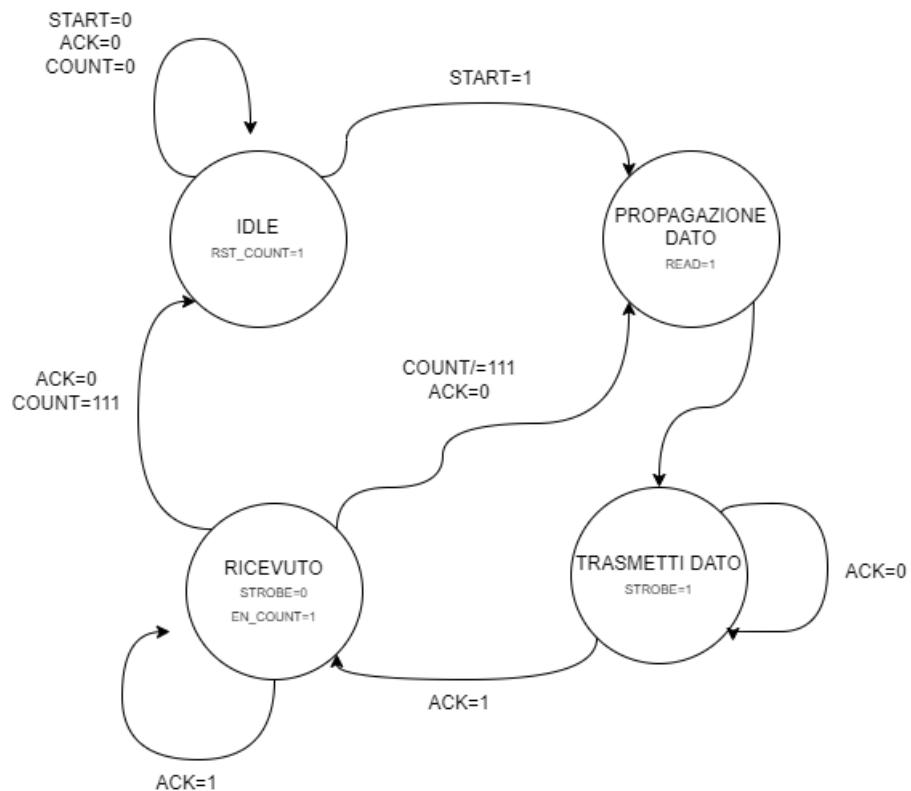
Si è progettato il design del sistema richiesto applicando un protocollo di handshaking a 2 vie per regolare la comunicazione tra i nodi A e B. Entrambi i nodi hanno una memoria interna in cui sono memorizzate N stringhe di M bit, denotate come X(i) per il nodo A e Y(i) per il nodo B, dove i varia da 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.

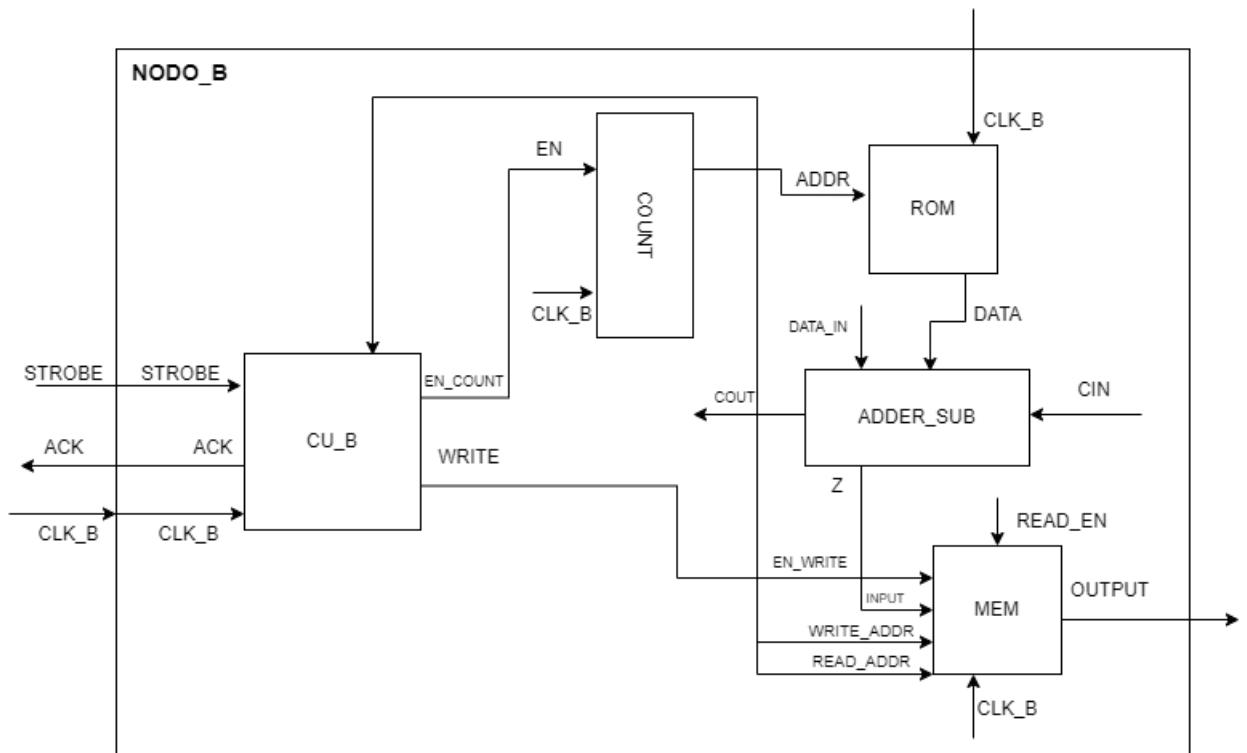


La control unit del nodo descrive il comportamento della macchina basandosi dell' automa a stati finiti. Inizialmente, fino a quando non arriva dall' esterno un segnale di START, il sistema permane nello stato "IDLE" ed il contatore resta resettato ($RST_COUNT=1$). Quando START diventa '1' il sistema 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

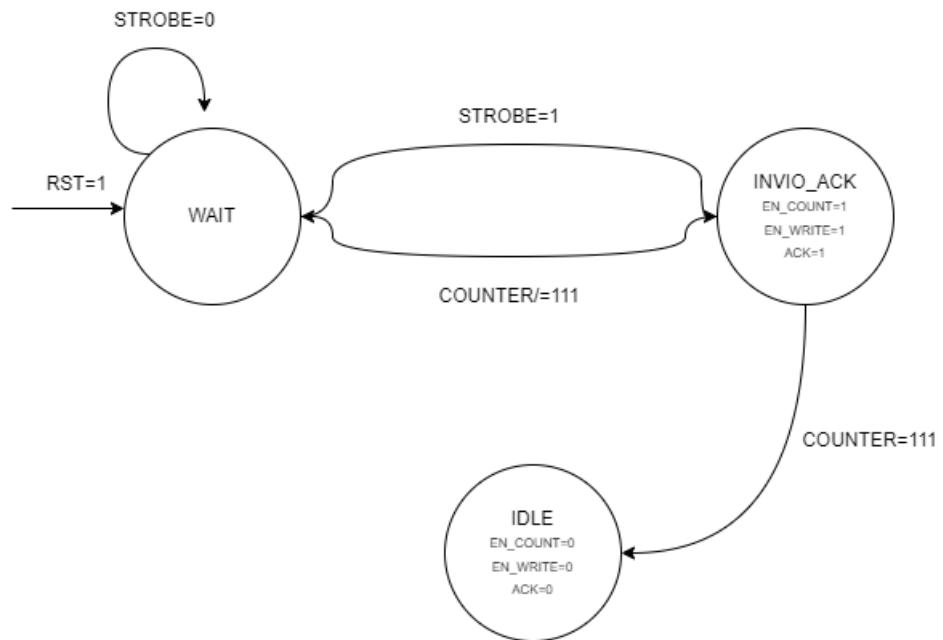
il segnale "ack" resta alto. Nel momento in cui il conteggio uscente dal contatore non ha ancora raggiunto il valore massimo, lo stato successivo del sistema sarà di nuovo "PROPAGAZIONE DATO", viceversa nel caso in cui il conteggio raggiunge il valore massimo, il sistema torna allo stato "IDLE". In entrambi 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 fintanto 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, ricevuto);
21 signal curr_state: stato := idle;
22 signal next_state: stato := idle;
23
24 begin
25
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             when waiting =>
36                 if strobe='1' then
37                     next_state <= invio_ack;
38                 else
39                     next_state <= waiting;
40                 end if;
41             when invio_ack =>
42                 en_write <= '1';
43                 en_count <= '1';
```

```

44         ack <= '1';
45
46         if(count = "111") then
47             next_state <= idle;
48         else
49             next_state <= waiting;
50         end if;
51
52     end case;
53
54 end process;
55
56 mem: process(clk)
57 begin
58     if rising_edge(clk) then
59         if reset='1' then
60             curr_state <= waiting;
61         else
62             curr_state <= next_state;
63         end if;
64     end if;
65 end process;
66
67 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.

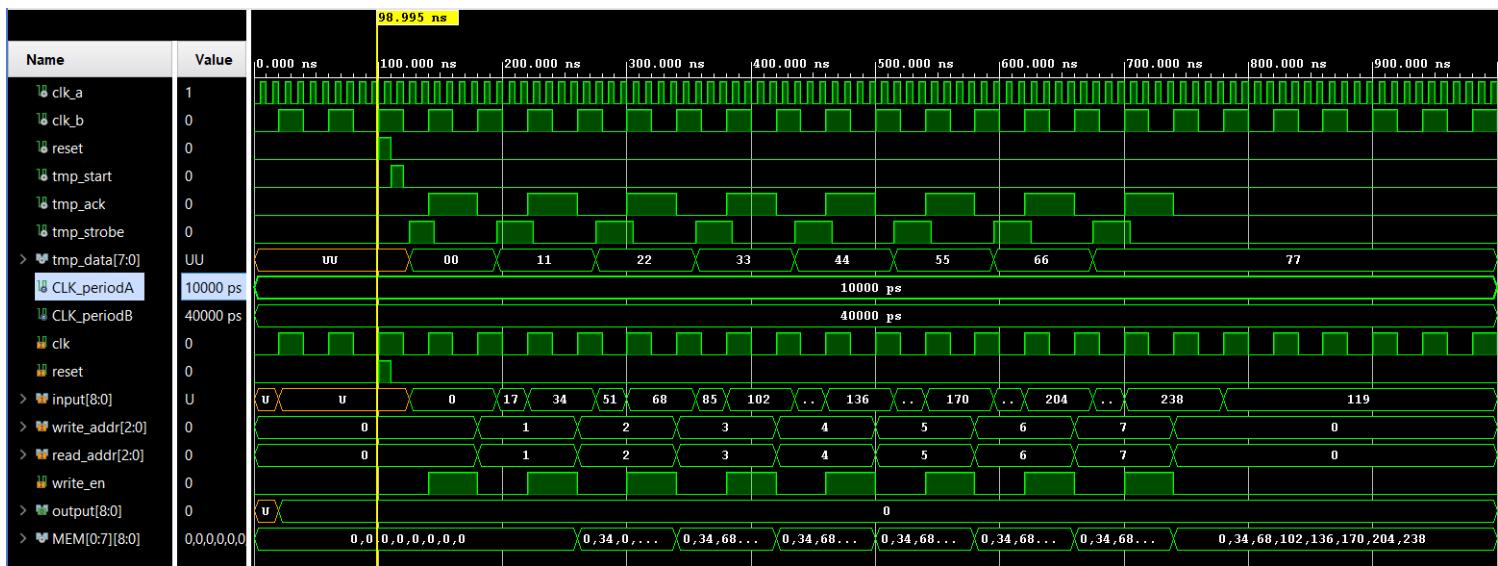


Figure 8.1: Simulazione con clock differenti

Chapter 9

Processore MIC-1

9.1 Traccia

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

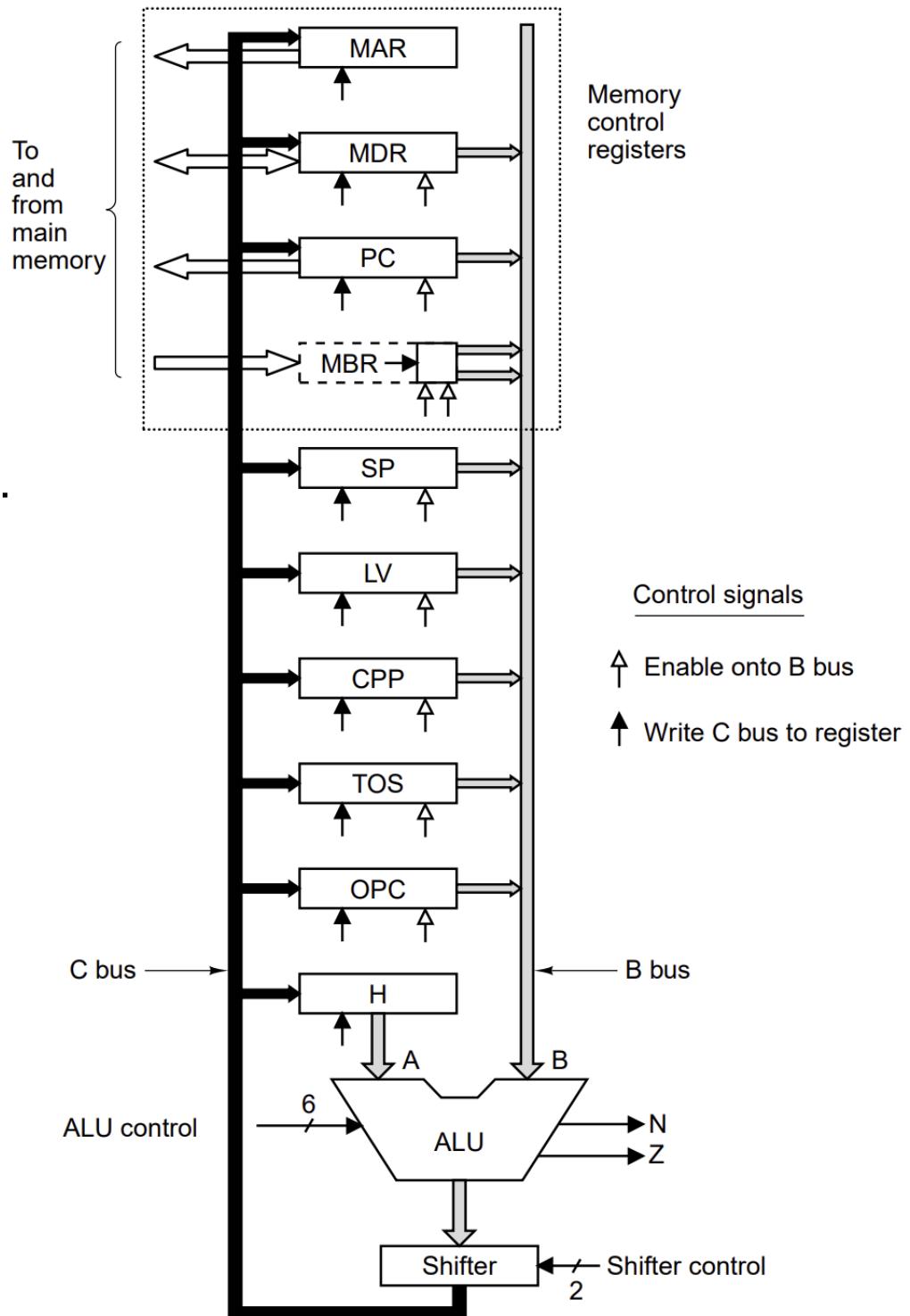
- si proceda all'analisi dell'architettura mediante simulazione e si approfondisca lo studio del suo funzionamento per due istruzioni a scelta,
- si modifichi un codice operativo a scelta, documentando tutte le modifiche effettuate

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 programma.
- 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.

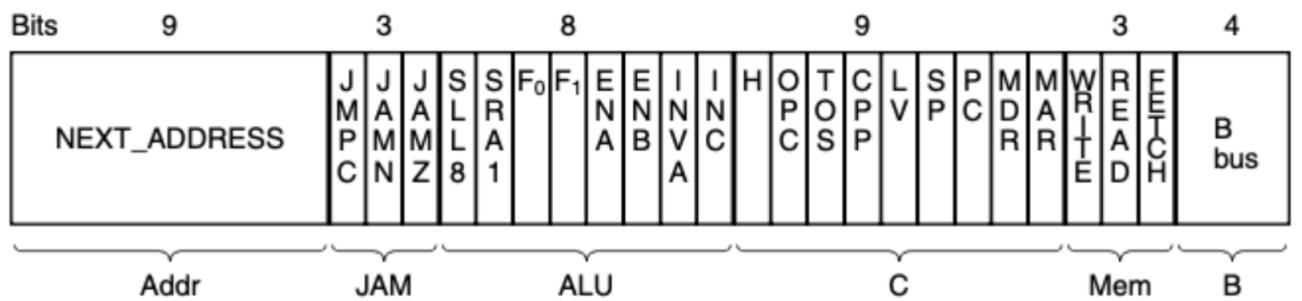


Figure 9.1: Struttura di una micro-istruzione MIC-1.

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.

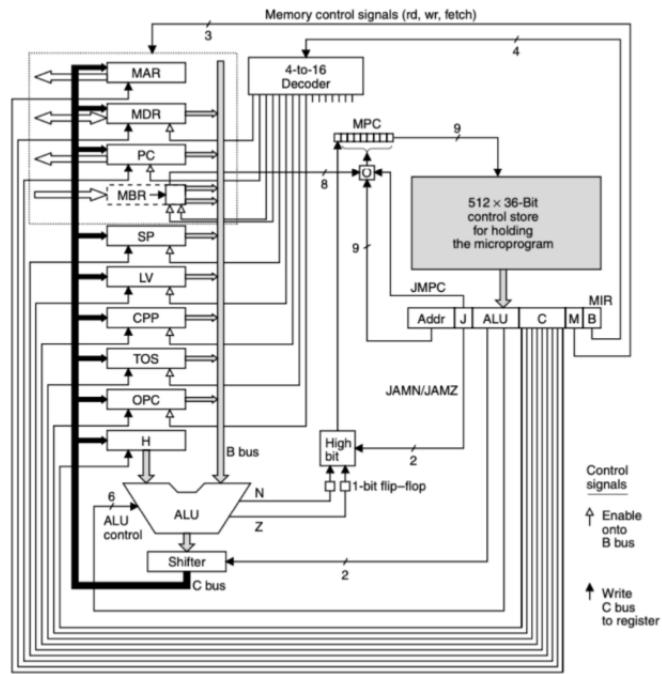


Figure 9.2: Architettura del processore MIC-1

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 BIPUSH 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     SP = MAR = SP + 1
3     PC = PC + 1; fetch
4     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 BIPUSH, 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 durante l'esecuzione dell'istruzione. Inoltre, abbiamo notato che il valore del registro Top of Stack (TOS) corrispondeva esattamente al valore desiderato, ovvero 0x0A.

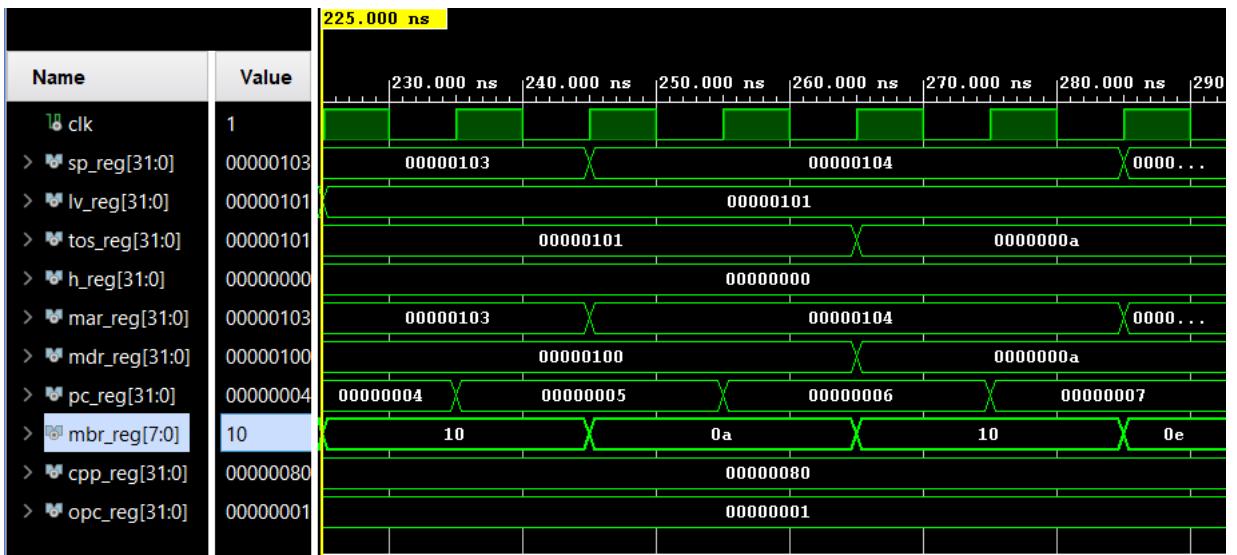


Figure 9.4: Stato dei registri del datapath durante l'esecuzione della BIPUSH.

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
6
7     MDR = TOS = MDR OR H; wr; goto main

```

Prima dell'esecuzione dell'istruzione IOR, si esegue il micropogramma “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.

- $\text{MAR} = \text{SP} = \text{SP} - 1; \text{rd}$: Questa microistruzione decremente 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 = TOS: Questa microistruzione posiziona il primo operando, il cui valore è memorizzato in TOS, nel registro di holding dell'ALU (H).
- MDR = TOS = MDR OR H; wr; 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 => "00000001000000000000000010000000",
69      1 => "0000110000100000000101000010000",
70      2 => "00000000000000000000001010011110110110", 10110110,
71      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.

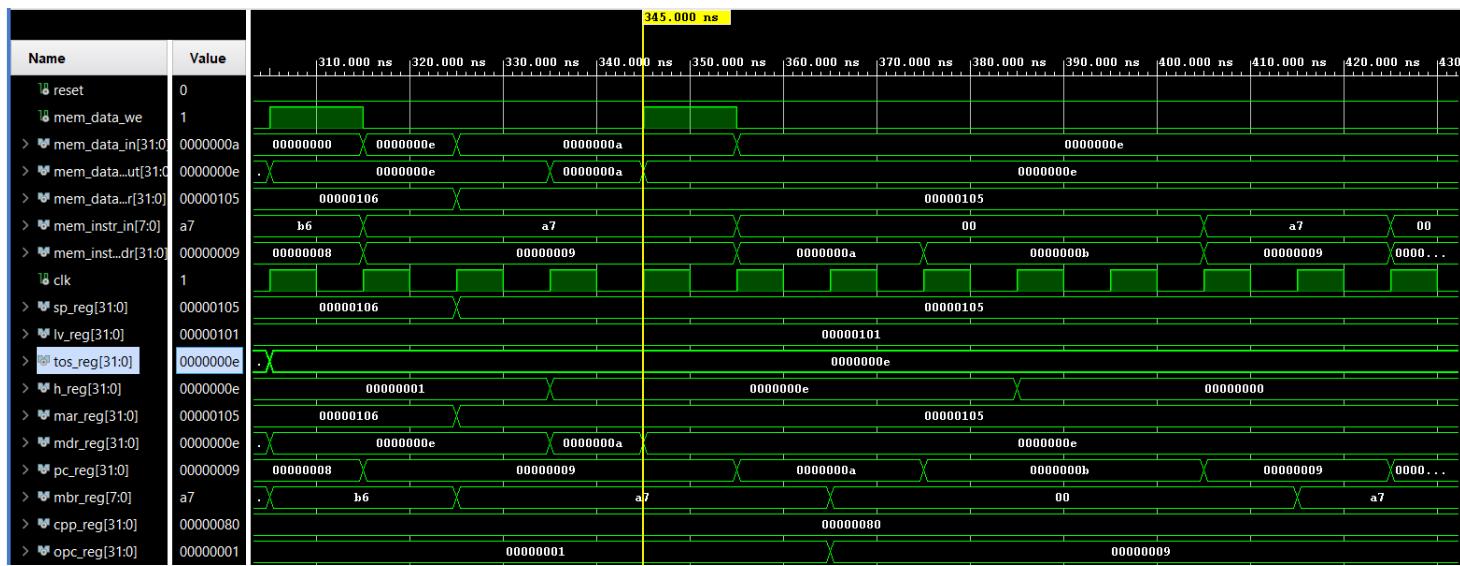


Figure 9.6: Stato dei registri del datapath durante l'esecuzione della IOR.

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.

```

64      -- RAM content
65      signal mem : dp_ar_ram_type := (
66      --BEGIN_WORDS_ENTRY
67      128 => "00000000000000000000000000000000",
68      0 => "00000010000000000000000000000000",
69      1 => '00000010000100000000000000000000',
70      2 => "1010011101101010000001100010000",
71      3 => "00000000000000000000000000000000",
72      others => (others => '0')
73      --END_WORDS_ENTRY
74      );

```

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.

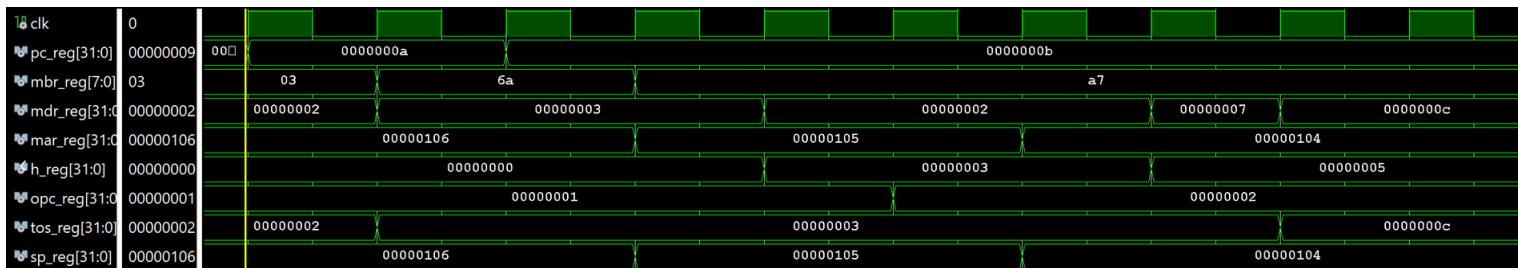


Figure 9.9: Stato dei registri del datapath durante l'esecuzione della IOR.

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:

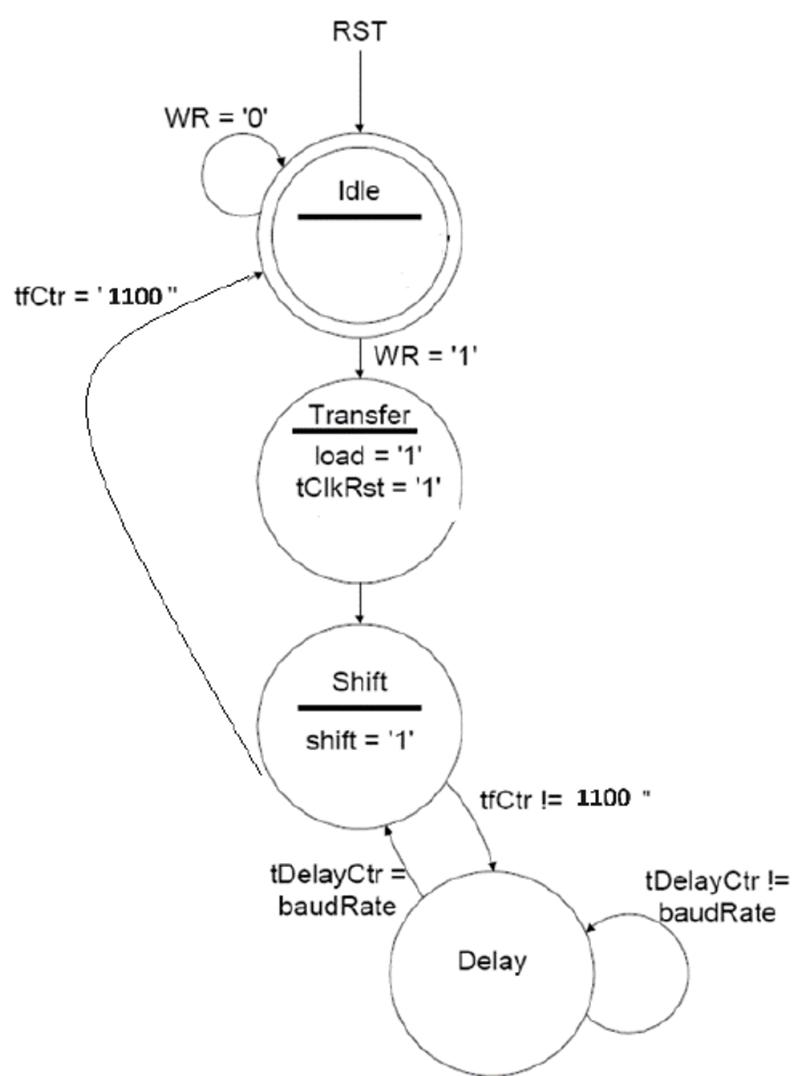
- BaudRate: velocità di trasmissione delle informazioni
- Lunghezza del frame: lunghezza della stringa di bit da trasmettere
- Endianness: ordine in cui i bit vengono trasmessi
- Sincronizzazione
- Controllo di errore

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:

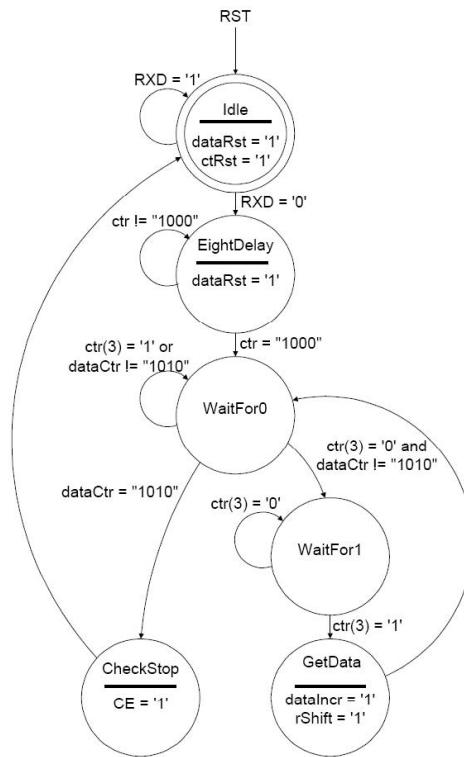
- * Il contatore fCtr tiene traccia del numero di bit trasmessi. Quando fCtr raggiunge il valore di 12, che corrisponde alla lunghezza del frame di dati più un bit aggiuntivo (tipicamente usato per garantire che ci sia un intervallo tra la fine di un frame di dati e l'inizio del successivo), la macchina 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

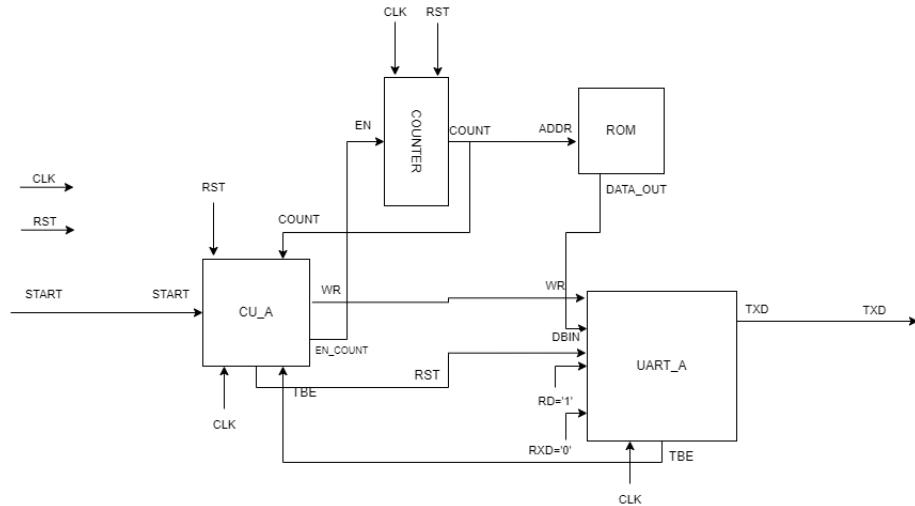
Quando è inattivo, il pin dei dati seriali in input (RXD) viene mantenuto alto

- 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 $\text{ctr}=8$, 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 $\text{dataCtr}=10$ (8 bit di dati, 1 bit di parità e 1 bit di stop), viene attivato lo stato CheckStop che abilita il controller degli errori.



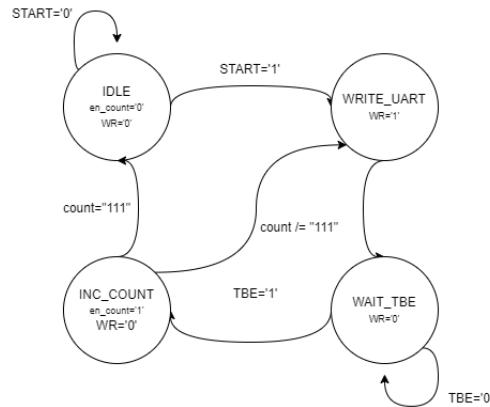
10.5 Implementazione progetto

10.5.1 Unità A

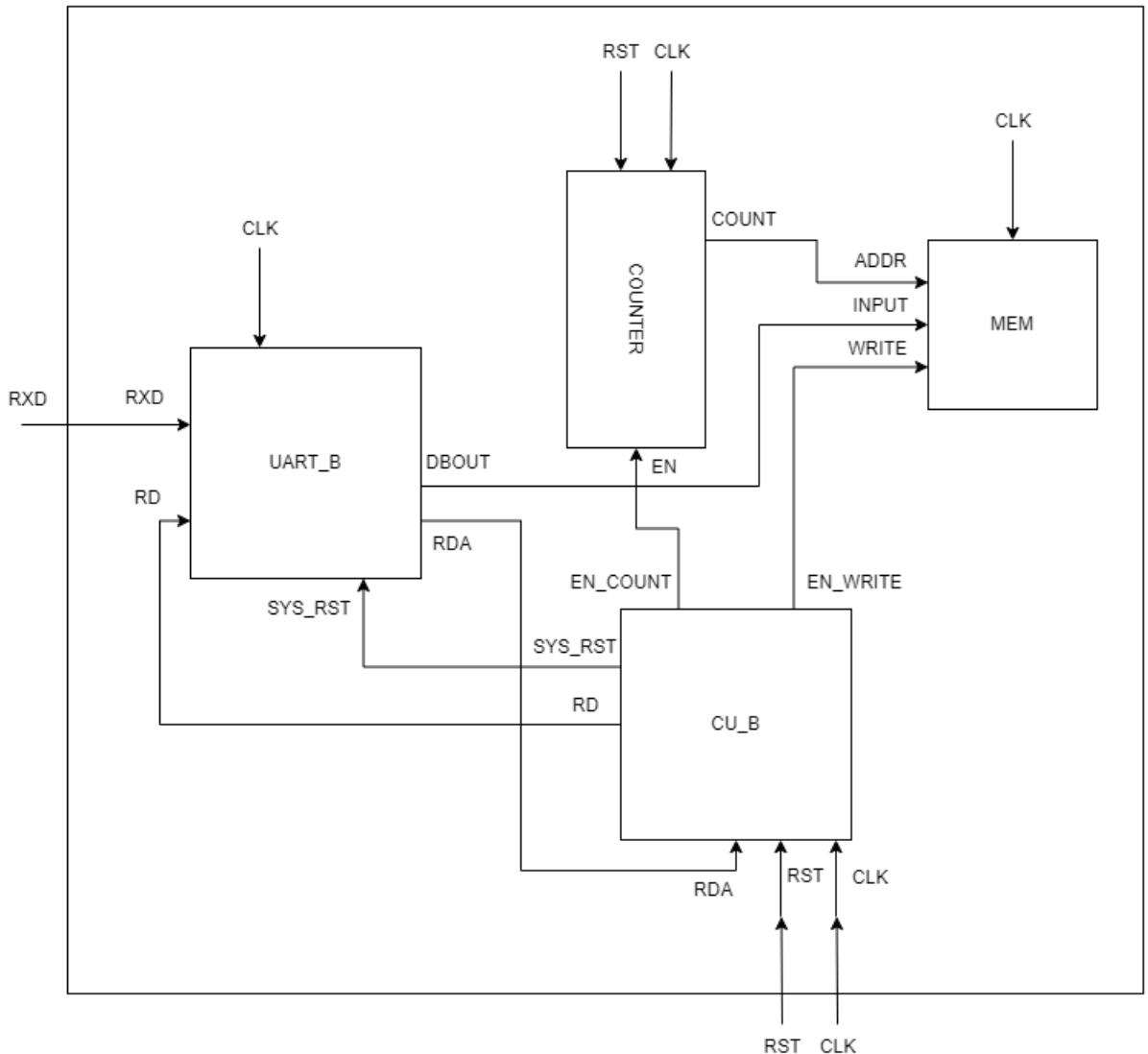


10.5.2 Automa a stati finiti - CU A

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.



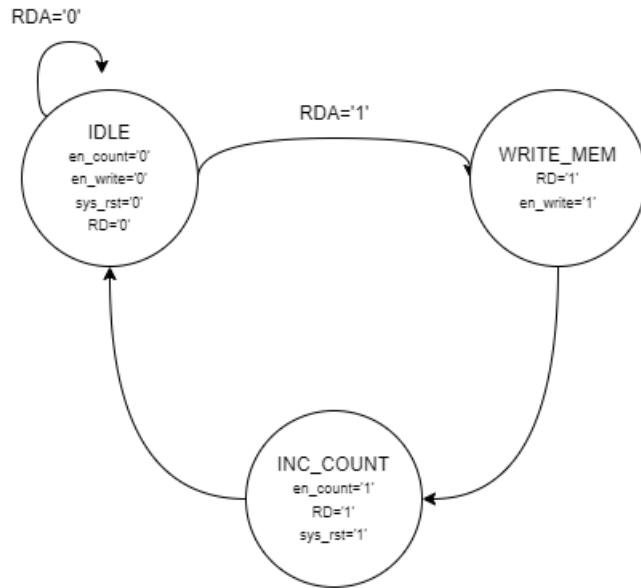
10.5.3 Unità B



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 così 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 volontà 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_Serial;
11
12 architecture Structural of Comunicazione_Serial 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             reset => reset,
30             RXD => link_UARTS
31         );
32
33 end Structural;

```

10.6.2 CU A

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity CU_A is
5     port (
6         start: in std_logic;
7         clk: in std_logic;

```

```

8      reset: in std_logic;
9      count: in std_logic_vector(2 downto 0);
10     TBE: in std_logic;
11
12     en_count: out std_logic;
13     WR: out std_logic
14   );
15 end CU_A;
16
17 architecture Behavioral of CU_A is
18
19 type stato 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          else
50              stato_successivo <= INC_COUNT;
51          end if;
52
53      when INC_COUNT =>
54          en_count <= '1';
55          WR<='0';
56          if(count = "111") then
57              stato_successivo <= IDLE;
58          else
59              stato_successivo <= WRITE_UART;
60          end if;
61
62      end case;
63  end process;
64
65  mem: process(clk)
66 begin
67     if rising_edge(clk) then
68         if reset = '1' then
69             stato_corrente <= IDLE;
70         else
71             stato_corrente <= stato_successivo;
72         end if;
73     end if;
74  end process;
75

```

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  end process;
57
58 mem: process(clk)
59 begin
60     if rising_edge(clk) then
61         if reset = '1' then
62             stato_corrente <= IDLE;
63         else
64             stato_corrente <= stato_successivo;
65         end if;
66     end if;
67 end process;
68

```

```

69
70 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             en_count => en_count_temp,

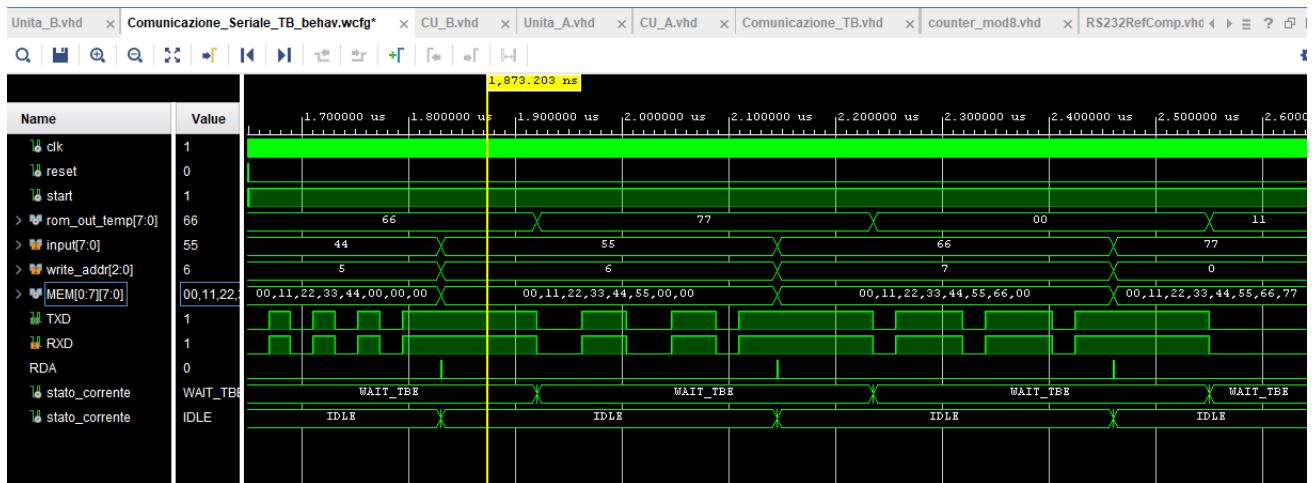
```

```

30          en_write => en_write_temp,
31          sys_rst => rst_temp,
32          RD => RD_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 MEM: entity work.mem
44     port map(
45         clk => clk,
46         input => data_mem_temp,
47         write_addr => count_temp,
48         write_en => en_write_temp
49     );
50
51 UART_B: entity work.Rs232RefComp
52     port map(
53         RXD => RXD,
54         CLK => clk,
55         DBIN => (others => '0'),
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

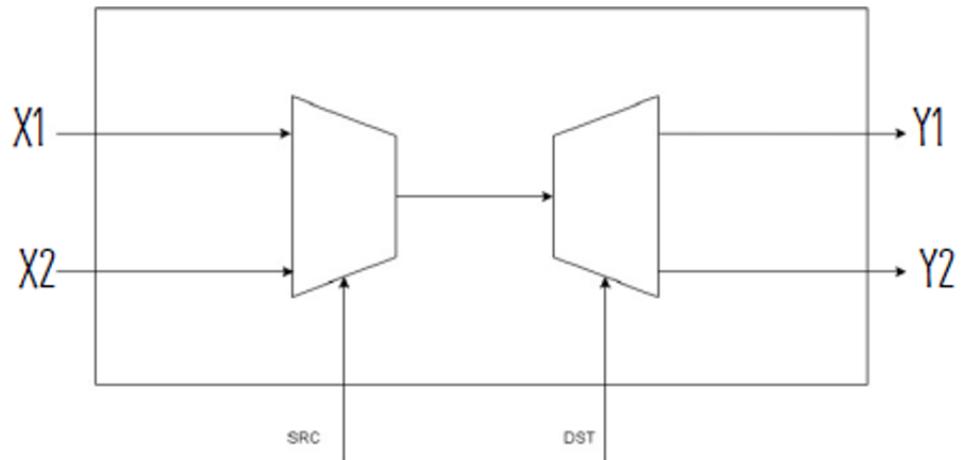
Switch Multistadio

11.1 Traccia

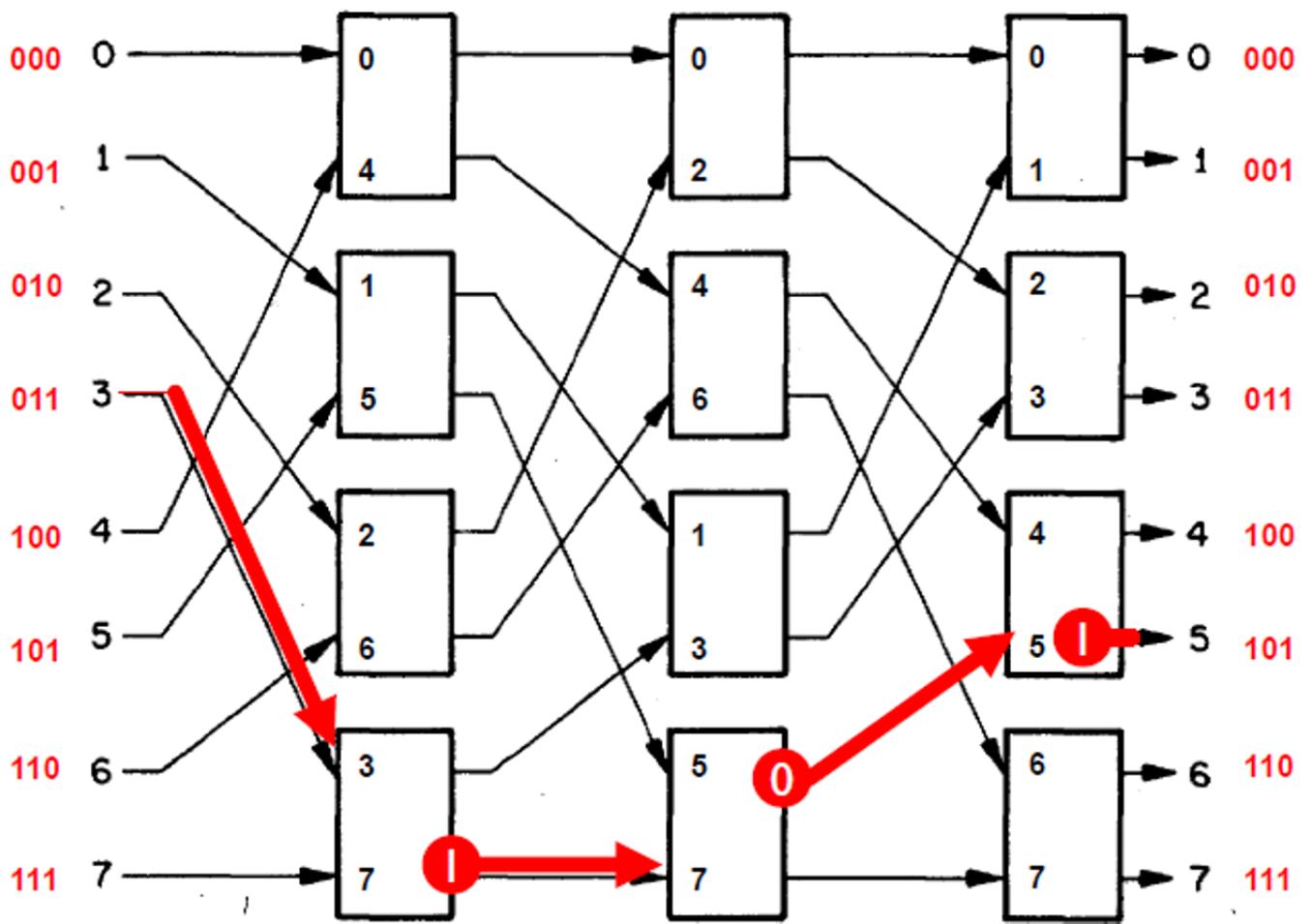
- Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network. Lo switch deve consentire lo scambio di messaggi di 2 bit ciascuno da un nodo sorgente a un nodo destinazione in una rete con 4 nodi, implementando uno schema a priorità fissa fra i nodi (es. nodo 1 più prioritario, 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 X_1 e X_2 con due destinazioni Y_1 e Y_2 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.



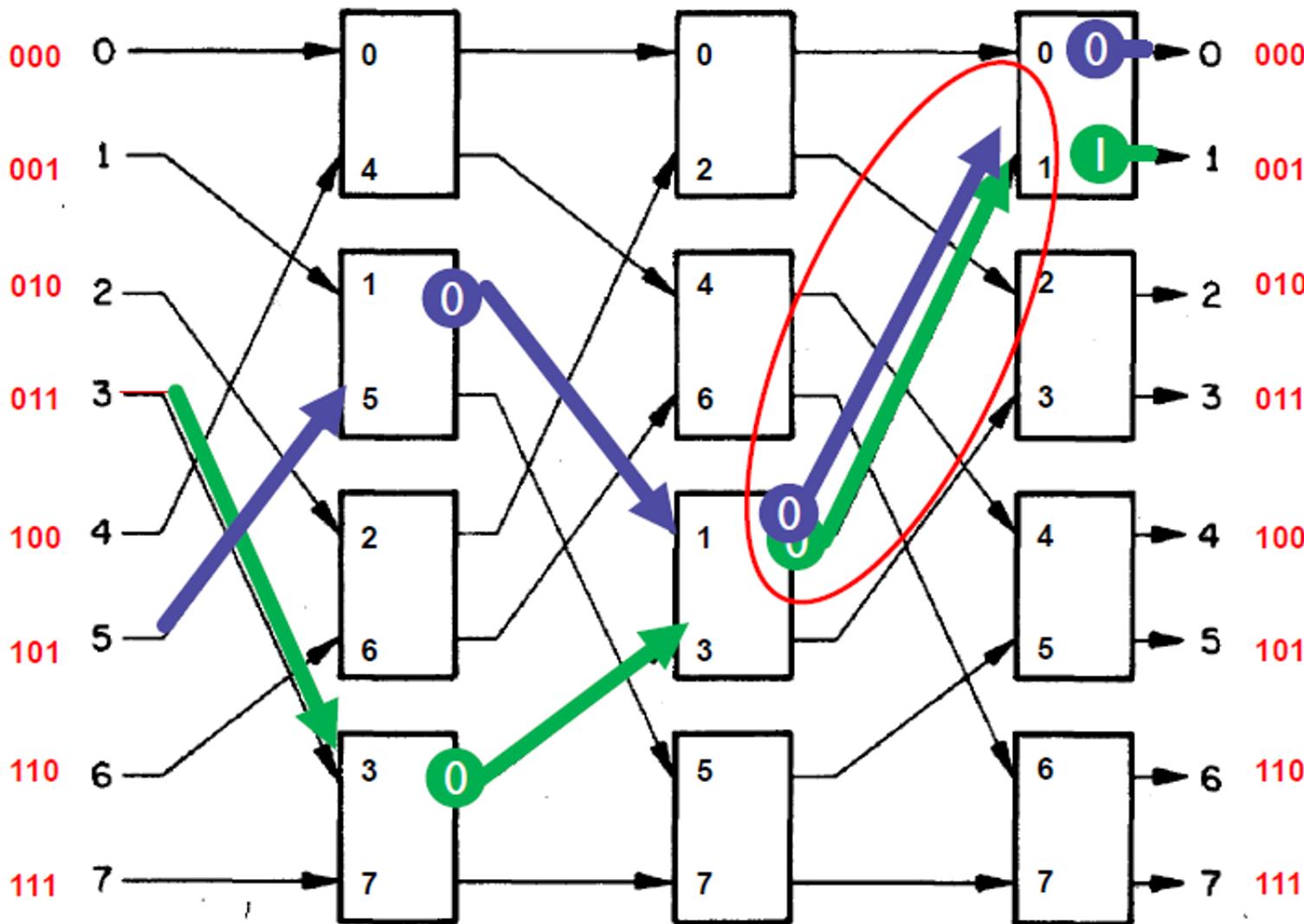
La Omega Network é un'architettura di interconnessione (basata su switch) . Si propone di far comunicare tra loro N nodi tra loro, per cui per una matrice $N \times N$ vi saranno $\log_2 N$ stadi identici che sfruttano un'interconnessione fra i nodi basata sul perfect shuffling (algoritmo derivante dal 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:

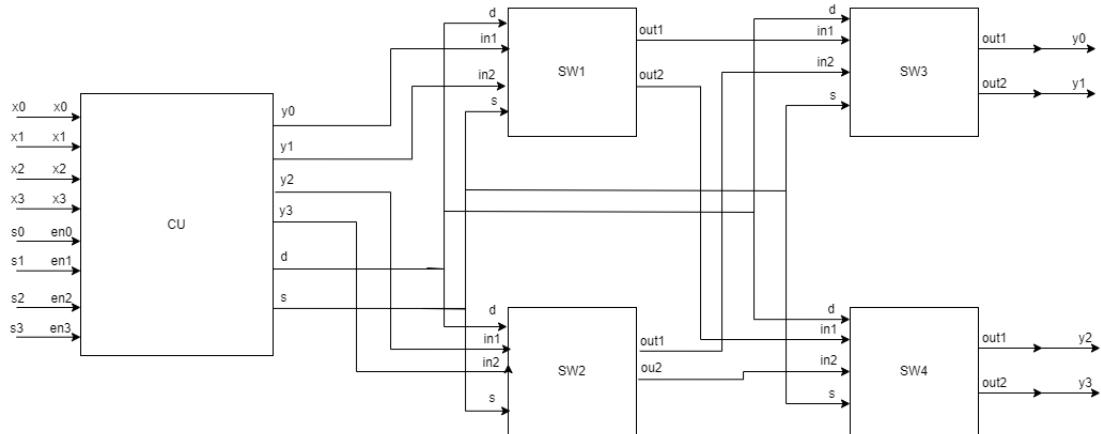
- Blocco: il nodo che non è in grado di propagare i segmenti interrompe l'avanzare del wormhole finché non si liberano le risorse necessarie (non riceve più segmenti)
- Perdita pacchetti: che non è in grado di propagare i segmenti si limita a distruggerli (rete con perdita di dati).
- Re-instradamento: 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)

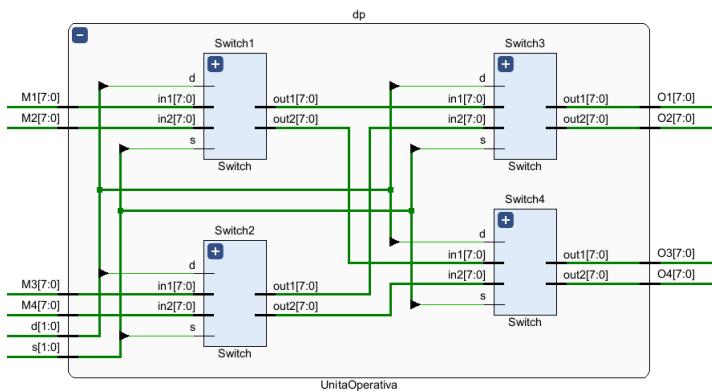
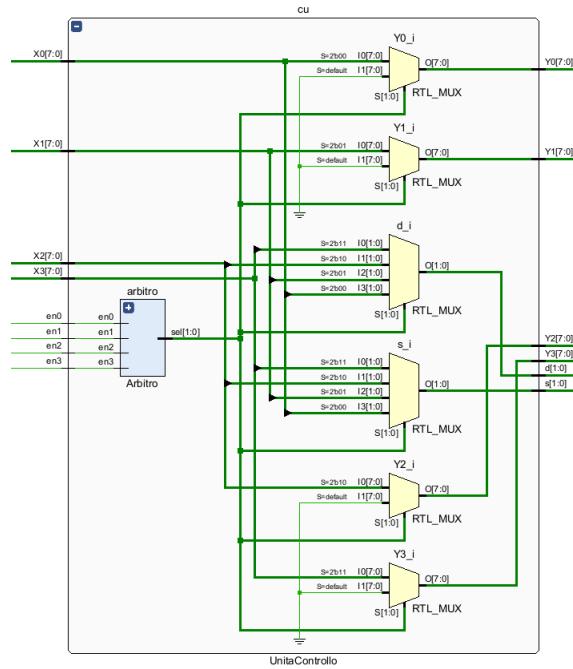
- Cut-through: consiste nell'adottare una filosofia storeforward: i segmenti che non possono essere inoltrati vengono bufferizzati finché c'è spazio

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 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         al => 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 end process;
29
30 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 UnitControllo 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 punt[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

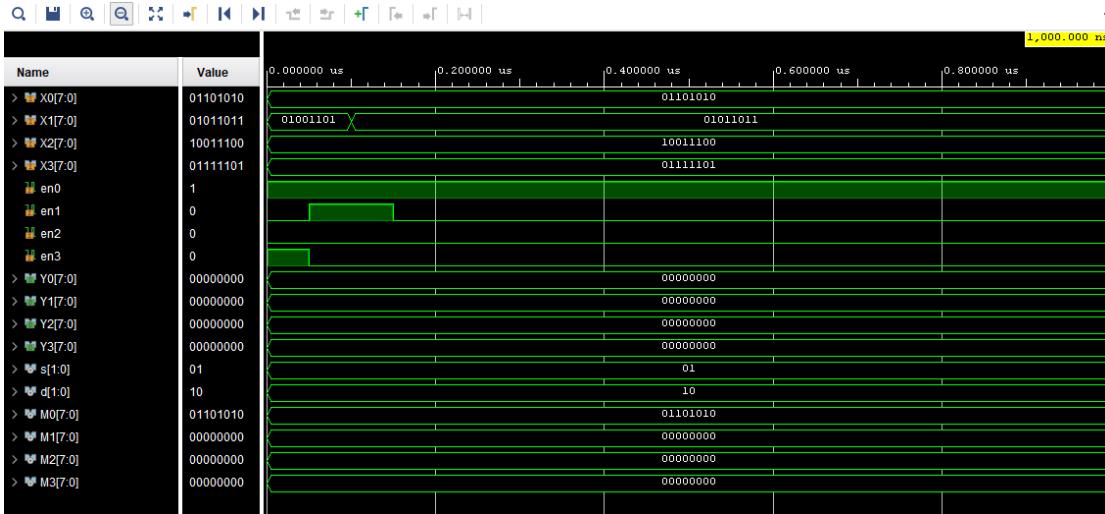


Figure 11.1: Switch multistadio

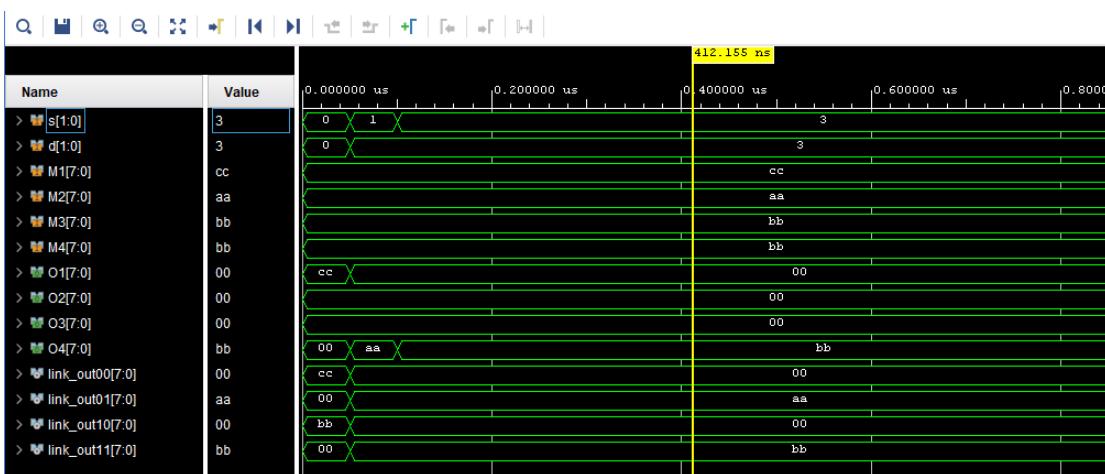


Figure 11.2: Omega Network

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 ingressi
9     S : in std_logic; --selezione
10    Y : out std_logic --uscita
11  );
12 end mux_2_1;
13
14 --Architettura livello Dataflow
15 --qui+FFFDBfme se stessi specificando la tabella di valori[U+FFFD] della
16 --funzione y
17 architecture dataflow of mux_2_1 is
18
19 begin
20   Y <= ((A(0) AND (NOT S)) OR (A(1) AND S)); --assegnazione
21   concorrente
22
23 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
9     port(
10         a: in std_logic_vector(N-1 downto 0);
11         s: in std_logic;
12         y1: out std_logic_vector(N-1 downto 0);
13         y2: out std_logic_vector(N-1 downto 0)
14     );
15
16
17
18
19 architecture Behavioral of Demux2_1 is
20
21 begin
22     process_demux : process(a,s)
23
24         begin
25             case s is
26                 when '0' => y1 <= a;
27                 when '1' => y2 <= a;
28             end case;
29         end process;
30     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 std_logic_vector(7 downto
16 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 <= memoria(conv_integer(ADDRESS));

```

```

33         end if;
34
35     end if;
36
37 end process;
38
39 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 downto 0);
17 signal ROM: rom_type := (
18     X"00",
19     X"11",
20     X"22",
21     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

```

23      div_out : process (T)
24      begin
25          if(to_integer(unsigned(T)) >= M-1)then
26              DIV<='1';
27          else
28              DIV<='0';
29          end if;
30      end process;
31
32      conteggio : process (CLK)
33      begin
34          if (rising_edge (CLK)) then
35              if(RST='1')then
36                  T <= (others =>'0');
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) +1);
42                  end if;
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 => '0');
17
18 begin
19
20     proc: process(clk)
21
22     begin
23
24         if rising_edge(clk) then
25             if (reset = '1') then
26                 temp_c <= (others => '0');
27             else
28                 if (enable = '1') then
29                     temp_c <= std_logic_vector(unsigned(temp_c) + 1);
30                 end if;
31             end if;
32         end if;
33     end process;
34
35     count <= temp_c;
36
37 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))))-1
20             downto 0);
21     DATA_INPUT: in std_logic_vector(LunghezzaWord-1 downto 0);
22     DATA_OUT: out std_logic_vector(LunghezzaWord-1 downto 0)
23 );
24
25 end MemoriaRW;
26
27
28 architecture Behavioral of MemoriaRW is
29 TYPE registri is array (NumeroLocazioni-1 downto 0) of
30     std_logic_vector(LunghezzaWord-1 downto 0);
31 signal memoria : registri;
32
33 begin
34
35     memo_behavioral: process(CLK)

```

```

30 begin
31     if(rising_edge(CLK)) then
32         if(RST = '1') then
33             for k in 0 to NumeroLocazioni-1 loop
34                 memoria(k) <= (others =>'0');
35                 DATA_OUT <= (others => '0');
36             end loop;
37         elsif(W = '1') then
38             memoria(conv_integer(ADDRESS)) <= DATA_INPUT;
39         elsif(R = '1') then
40             DATA_OUT <= memoria(conv_integer(ADDRESS));
41         end if;
42     end if;
43 end process;
44 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 => 'U');
19
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
29     end if;
30
31 end process;
32
33 output <= reg;
34
35 end Behavioral;
```

12.11 Full Adder

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity full_adder is
5
6     port(
7         a: in std_logic; -- Primo operando del Full Adder
8         b: in std_logic; -- Secondo operando del Full Adder
9         cin: in std_logic; -- Carry-In, il riporto da
10            un'eventuale addizione precedente
11
12         cout: out std_logic; -- Carry-Out, il riporto risultante
13            dall'addizione dei due operandi e del Carry-In
14
15         s: out std_logic -- Sum bit, il risultato dell'addizione
16            dei due operandi e del Carry-In
17
18     );
19
20 end full_adder;
21
22
23 architecture dataflow of full_adder is
24
25 begin
26
27     -- Assegna al port di uscita S il risultato dell'operazione
28     -- XOR tra gli operandi e il Carry-In.
29
30     -- Questo è il valore del bit di somma per l'addizione
31     -- binaria.
```

```

20      s <= (a xor b) xor cin;
21      -- Assegna al port di uscita COUT il risultato dell'operazione
22      -- OR tra:
23      -- 1. L'AND degli operandi, che genera un carry se entrambi
24      -- gli operandi sono 1.
25      -- 2. L'AND tra il Carry-In e l'OR degli operandi, che genera
26      -- un carry se uno degli operandi e il Carry-In sono 1.
27      -- Questo valore del Carry-Out per l'addizione binaria.
28      cout <= (a and b) or (cin and (a or b));
29
30
31  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); -- Ingressi: due
7         -- operandi a 8 bit da sommare.
8         c_in: in std_logic; -- Ingresso: il riporto iniziale.
9         c_out: out std_logic; -- Uscita: il riporto finale.
10        Z: out std_logic_vector(7 downto 0) -- Uscita: risultato
11        -- della somma a 8 bit.
12    );
13 end ripple_carry;
14
15
16 architecture structural of ripple_carry is
17 component full_adder is
18     port(
19         a,b: in std_logic;
20         cin: in std_logic;
21         sum: out std_logic;
22         cout: out std_logic;
23     );
24 end component;
25
26 begin
27     process(X,Y,c_in)
28     variable sum,cout: std_logic;
29     begin
30         for i in 0 to 7 loop
31             if (i = 0) then
32                 sum := X(0) xor Y(0) xor c_in;
33                 cout := X(0) and Y(0);
34             else
35                 sum := X(i) xor Y(i) xor cout;
36                 cout := X(i) and Y(i) and cout;
37             end if;
38             Z(i) <= sum;
39         end loop;
40         c_out <= cout;
41     end process;
42 end structural;

```

```

18         cout, s: out std_logic);
19 end component;
20
21 signal temp: std_logic_vector(7 downto 0);
22
23 begin
24
25 RA0: full_adder port map(X(0), Y(0), c_in, temp(0), Z(0));
26
27 RA1to6: FOR i IN 1 TO 6 GENERATE
28
29     RA: full_adder port map(X(i), Y(i), temp(i-1),
30
31     temp(i), Z(i));
32
33     END GENERATE;
34
35 RA7: full_adder port map(X(7), Y(7), temp(6), c_out, Z(7));
36
37 end structural;

```

12.13 Adder / Subtractor

```

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

```

```

10    );
11 end adder_sub;
12
13 architecture Structural of adder_sub is
14
15 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 valore di cin.
32     -- Se cin = '0' (addizione), Y rimane invariato.
33     -- Se cin = '1' (sottrazione), Y viene complementato.
34     complemento_y: FOR i IN 0 TO 7 GENERATE
35         complementoy(i) <= Y(i) xor cin; -- Operazione xor per
36         -- complementare Y se necessario.
37     END GENERATE;
38
39     -- Istanza del componente ripple_carry
40     RA: ripple_carry port map(X, complementoy, cin, cout, Z);
41
42 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 della
8             board (in nanosecondi)
9         btn_noise_time: integer := 10000000 -- durata stimata
10            dell'oscillazione del bottone
11            -- il valore di
12            default 0
13            millisecondi
14        );
15    port(
16        RST: in std_logic;
17        CLK: in std_logic;
18        BTN: in std_logic;
19        CLEARED_BTN: out std_logic -- segnale di output ripulito
20    );
21 end ButtonDebouncer;
22
23
24 architecture Behavioral of ButtonDebouncer is
25
26 type stato is (NOT_PRESSED, CHK_PRESSED, PRESSED,
27                 CHK_NOT_PRESSED);
28
29 signal BTN_state: stato := NOT_PRESSED;
30
31
32 constant max_count : integer := btn_noise_time/CLK_period; --
33             10000000/10= conto 1000000 colpi di clock
34
35 begin
36
```

```

27
28     deb: process(CLK)
29
30         variable count: integer := 0;
31
32     begin
33
34         if rising_edge(CLK) then
35
36             if (RST = '1') then
37                 BTN_state <= NOT_PRESSED;
38                 CLEARED_BTN <= '0';
39
40             else
41                 case BTN_state is
42                     when NOT_PRESSED =>
43                         if (BTN = '1') then
44                             BTN_state <= CHK_PRESSED;
45
46                     else
47                         BTN_state <= NOT_PRESSED;
48                 end if;
49
50
51             when CHK_PRESSED =>
52                 if (count = max_count-1) then --conta i
53                     cicli di clock fino a max_count-1
54                     (ovvero 10ms)
55
56                     if (BTN = '1') then --se arrivo a
57                         count max #0+FFF#h corrispondente alto
58                         vuol dire che non era un
59                         bounce, devo alzare CLEARED_BTN
60                         count := 0;
61                         CLEARED_BTN <= '1';
62                         BTN_state <= PRESSED;
63
64                     else
65                         count := 0;

```

```

56             BTN_state <= NOT_PRESSED;
57         end if;
58     else
59         count := count+1;
60         BTN_state <= CHK_PRESSED;
61     end if;
62
63     when PRESSED => --Abbasso subito
64         CLEARED_BIN per avere un singolo impulso
65         CLEARED_BTN <= '0';
66         if(BTN = '0') then
67             BTN_state <= CHK_NOT_PRESSED;
68         else
69             BTN_state <= PRESSSED;
70         end if;
71
72     when CHK_NOT_PRESSED =>
73         if(count = max_count-1) then
74             if(BTN = '0') then --se arrivo a
75                 count max #FFF#cora basso
76                 vuol dire che non era un bounce
77                 e il bottone#FFF#ato rilasciato
78                 count := 0;
79                 BTN_state <= NOT_PRESSED;
80             else
81                 count := 0;
82                 BTN_state <= PRESSSED;
83             end if;
84         else
85             count := count+1;
86             BTN_state <= CHK_NOT_PRESSED;
87         end if;
88
89     when others =>

```

```
86          BTN_state <= NOT_PRESSED;  
87      end case;  
88      end if;  
89      end if;  
90  end process;  
91  
92 end Behavioral;
```