

Elaborato Finale

Corso: Architetture di Sistemi Digitali

Gruppo 28

Nome Cognome	Matricola
Marano Stefano	M63001428
Romano Vito	M63001504
Russo Paolo	M63001426
Cimmino Marco	M63001528

21 aprile 2023

Indice

1 Multiplexer	4
1.1 Traccia	4
1.2 Descrizione Soluzione	4
1.3 Testbench	5
1.4 Sintesi	6
1.5 Codice VHDL	6
1.5.1 Punto 1 - Multiplexer 2 a 1	6
1.5.2 Punto 1 - Multiplexer 4 a 1	6
1.5.3 Punto 1 - Multiplexer 16 a 1	7
1.5.4 Punto 2 - Demux 1 a 4	9
1.5.5 Punto 2 - Bus 16 a 4	9
1.5.6 Punto 3 - Sintesi	10
2 Encoder BCD	11
2.1 Traccia	11
2.2 Descrizione Soluzione	11
2.3 Testbench	11
2.4 Sintesi	12
2.5 Codice	13
2.5.1 Punto 1 - Arbitro Priorità	13
2.5.2 Punto 1 - Encoder 10 a 4	13
2.5.3 Punto 2 - Sintesi con Switch e Led	13
2.5.4 Punto 3 - Display a 7 Segmenti	14
2.5.5 Punto 3 - Sintesi con Display	16
3 Riconoscitore di Sequenze	18
3.1 Traccia	18
3.2 Descrizione Soluzione	18
3.2.1 Punto 2	19
3.3 Testbench	20
3.4 Sintesi	20
3.5 Codice	20
3.5.1 Punto 1 - Riconoscitore	20
3.5.2 Punto 2 - Button Debouncer	23
3.5.3 Punto 2 - Rete Complessiva	24
3.5.4 Punto 2 - Mapping su scheda	25
4 Shift Register	27
4.1 Traccia	27
4.2 Descrizione Soluzione	27
4.3 Testbench	28
4.4 Codice	29
4.4.1 Approccio Comportamentale	29
4.4.2 Approccio Strutturale	31
4.4.3 Flip Flop D	33
5 Cronometro	35
5.1 Traccia	35
5.2 Descrizione Soluzione	35
5.2.1 Punto 1 - Contatori	36
5.2.2 Punto 2 - Struttura Complessiva	36
5.2.3 Punto 3 - Struttura Complessiva	36
5.3 Testbench	38
5.4 Sintesi	39
5.5 Codice VHDL	39

5.5.1	Flip Flop T	39
5.5.2	Punto 1 - Contatore Modulo 24	40
5.5.3	Punto 1 - Contatore Modulo 60	42
5.5.4	Punto 1 - Cronometro	44
5.5.5	Punto 2 - Clock Filter	46
5.5.6	Punto 2 - Convertitore	47
5.5.7	Punto 3 - Blocco Intertempo	48
5.5.8	Punto 3 - Blocco Mantieni Uscita	50
5.5.9	Punto 3 - Sistema Complessivo	51
5.5.10	Punto 3 - Mapping Scheda	53
6	Sistema di Testing	59
6.1	Traccia	59
6.2	Descrizione Soluzione	59
6.2.1	Punto 2	60
6.3	Testbench	60
6.4	Sintesi	61
6.5	Codice	61
6.5.1	Punto 1 - Macchina Combinatoria	61
6.5.2	Punto 1 - Memoria Rom	62
6.5.3	Punto 1 - Memoria Ram	63
6.5.4	Punto 1 - Contatore Modulo N	64
6.5.5	Punto 2 - Button Debouncer	65
6.5.6	Punto 2 - Sistema Complessivo	67
7	Comunicazione con Handshaking	69
7.1	Descrizione Soluzione	69
7.1.1	Struttura Nodo A	70
7.1.2	Struttura Nodo B	71
7.2	Testbench	73
7.3	Codice	73
7.3.1	Nodo A - Sistema Complessivo	73
7.3.2	Nodo A - Parte Operativa	75
7.3.3	Nodo A - Parte di Controllo	76
7.3.4	Nodo B - Sistema Complessivo	78
7.3.5	Nodo B - Parte Operativa	79
7.3.6	Nodo B - Parte di Controllo	81
7.3.7	Mem RW	83
7.3.8	Mem R	84
8	Processore	86
8.1	Analisi Architettura	86
8.1.1	Introduzione	86
8.1.2	Datapath	86
8.1.3	Unità di Controllo e Microistruzioni	87
8.2	Studio Istruzione 1 - BIPUSH	88
8.2.1	Simulazione	88
8.3	Studio Istruzione 2 - IAND	88
8.3.1	Simulazione	89
8.3.2	Modifica	89
8.4	Studio Istruzione 3 - IFLT	90
8.4.1	Simulazione	90
8.4.2	Modifica	90

9 Interfaccia Seriale	92
9.1 Introduzione	92
9.2 Descrizione Soluzione	94
9.2.1 Punto 1 - UART a Tappo	94
9.2.2 Punto 2 - UART MEM	94
9.3 Testbench	97
9.4 Sintesi	98
9.5 Codice	98
9.5.1 punto 1 - Sistema Complessivo	98
9.5.2 Punto 1 - Mapping	100
9.5.3 Punto 2 - Nodo A	102
9.5.4 Punto 2 - Nodo B	104
9.6 Punto 2 - UC A	107
9.7 Punto 2 - UC B	109
10 Switch Multistadio	112
10.1 Descrizione Soluzione	112
10.1.1 Introduzione	112
10.1.2 Perfect Shuffling	112
10.1.3 Punto 1 - Sistema Complessivo	112
10.1.4 Punto 2 - Variante Sistema Complessivo	113
10.2 Testbench	114
10.2.1 Test sistema 1	114
10.2.2 Test sistema 2	115
10.3 Codice	115
10.3.1 Parte 1 - Switch Elementare	115
10.3.2 Parte 1 - Arbitro	116
10.3.3 Parte 1 - Unità Controllo	116
10.3.4 Parte 1 - Unità Operativa	118
10.3.5 Parte 1 - Sistema Complessivo	119
10.3.6 Parte 2 - Variante Sistema Complessivo	121
11 Macchina Aritmetica	124
11.1 Descrizione Soluzione	124
11.1.1 Cifre negative	124
11.1.2 Schema della macchina	125
11.2 Testbench	126
11.3 Sintesi	127
11.4 Codice VHDL	128
11.4.1 Sistema Complessivo	128
11.4.2 Unità di Controllo	131
11.4.3 Ripple Carry Adder	133
12 Esercizio Libero	135
12.1 Descrizione Soluzione	135
12.1.1 Struttura Nodo A	135
12.1.2 Struttura Nodo B	137
12.2 Testbench	138
12.3 Codice	138
12.3.1 Nodo A	138
12.3.2 Nodo A - UC	141
12.3.3 Nodo B	143
12.3.4 Nodo B - UC	146

1 | Multiplexer

1.1 | Traccia

Punto 1 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.

Punto 2 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.

Punto 3 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 possono essere precaricati nel sistema oppure immessi anch'essi mediante switch, sviluppando in questo secondo caso un'apposita rete di controllo per l'acquisizione.

1.2 | Descrizione Soluzione

Il multiplexer è un selettori di linee, i segnali vengono raccolti e mandati in una singola linea di uscita, esistono vari tipi di multiplexer e il più semplice è il multiplexer 2:1.

Seguendo un **approccio modulare**, un multiplexer 16:1 può essere visto come una composizione di vari multiplexer 4:1 opportunamente interconnessi, a sua volta un multiplexer 4:1 può essere ottenuto interconnettendo vari multiplexer 2:1, quindi possiamo considerare quest'ultimo la cella fondamentale di questa macchina.

Il multiplexer 2:1 è una macchina combinatoria notevole avente due ingressi, una linea di selezione ed un'uscita, in base al valore dell'ingresso di selezione, una tra le linee di ingresso sarà portata sulla linea di uscita, per questo è risultato molto semplice da descrivere in maniera dataflow.

Siamo dunque partiti da una descrizione di tipo **dataflow** della cella fondamentale per poi procedere per le macchine da essa derivanti con descrizioni di tipo **structural**, in particolare: il multiplexer 16:1 è stato implementato tramite la composizione di cinque multiplexer 4:1, mentre ogni multiplexer 4:1 è stato implementato mediante la composizione di 3 multiplexer 2:1.

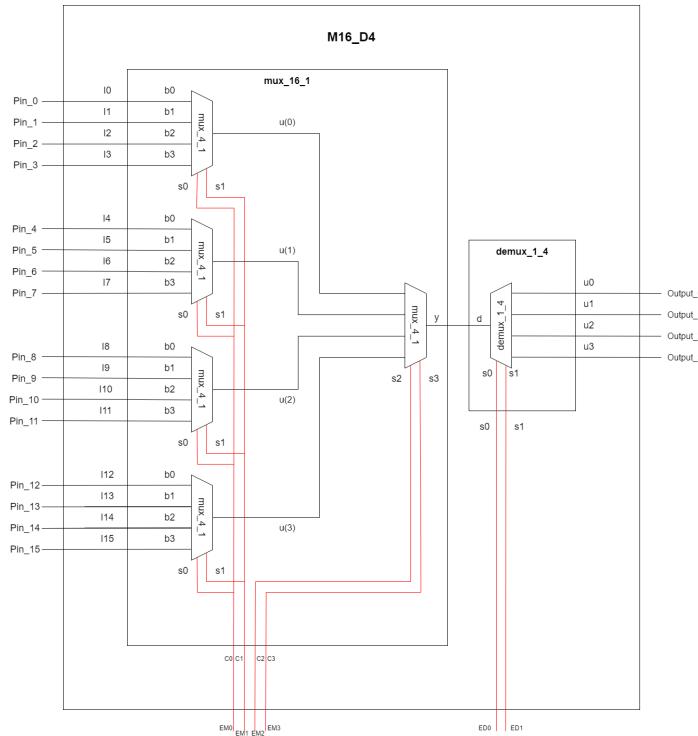
Il multiplexer 4:1 prende in ingresso 4 segnali, possiede 2 linee di selezione ed una linea di uscita, per realizzarlo attraverso composizione i 4 ingressi entrano in due multiplexer 2:1, questi ultimi producono un'uscita ciascuno che entrano in un terzo multiplexer che produce l'output finale.

In pratica stiamo dividendo l'indirizzo in ingresso al multiplexer in due parti, una più significativa $s(1)$ ed una meno significativa $s(0)$, quest'ultima va in ingresso ai primi due multiplexer come segnale di selezione, selezionando la linea del blocco, mentre il secondo serve per scegliere quale tra gli output dei due mux considerare.

Analogamente al caso del mux 4:1 abbiamo realizzato il multiplexer 16:1, i 16 ingressi entrano a gruppi in quattro multiplexer 4:1 i quali selezionano e portano in uscita un solo segnale tra quelli che ricevono, tali uscite vanno in un quinto multiplexer 4:1 che produce l'output finale a partire da essi.

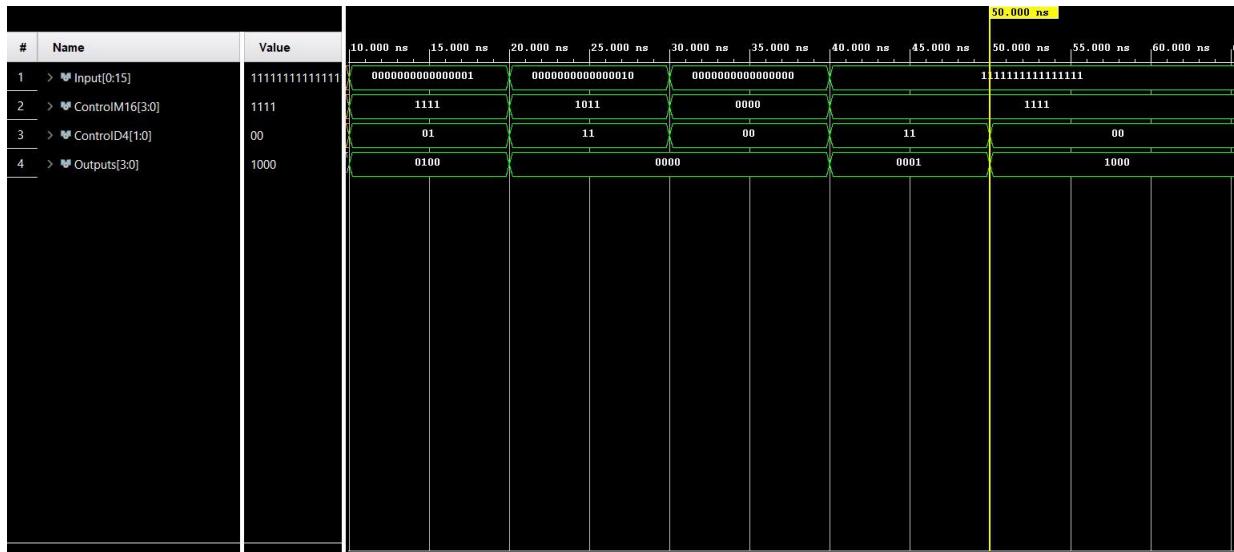
Lo schema è mostrato in figura 1.1 nel blocco mux_16_1.

Infine la rete bus prende in ingresso 16 segnali, ha 6 linee di selezione e fornisce 4 uscite, come possiamo notare entra in gioco anche un demultiplexer 1:4, il quale prende in ingresso l'uscita del multiplexer 16:1 ed ha 2 linee di selezione, quindi sulla base del valore di queste linee di selezione viene fornita su una delle 4 linee di uscita il valore di uno dei 16 ingressi.

**Figura 1.1:** Bus 16 a 4

1.3 | Testbench

Il testbench è generico in cui proviamo una serie di combinazioni di ingresso e uscita, è visibile nella figura 1.2.

**Figura 1.2:** Simulazione del comportamento generale del multiplexer

1.4 | Sintesi

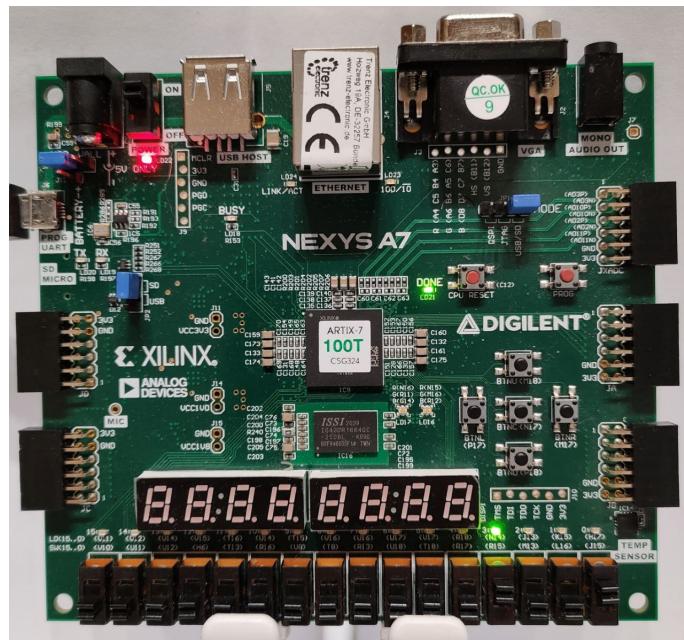


Figura 1.3: Sintesi su board dell'esercizio.

1.5 | Codice VHDL

1.5.1 | Punto 1 - Multiplexer 2 a 1

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

entity mux_2_1 is
  Port (
    a0 : in STD_LOGIC;
    a1 : in STD_LOGIC;
    s : in STD_LOGIC;
    y : out STD_LOGIC
  );
end mux_2_1;

architecture Dataflow of mux_2_1 is

begin
  y <= a0 when s = '0' else a1 when s = '1' else '-';
end Dataflow;
```

1.5.2 | Punto 1 - Multiplexer 4 a 1

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

-- Definizione dell'interfaccia del modulo mux_4_1.
entity mux_4_1 is
  port(
    b0 : in STD_LOGIC;
    b1 : in STD_LOGIC;
    b2 : in STD_LOGIC;
```

```

        b3 : in STD_LOGIC;
        s0 : in STD_LOGIC;
        s1 : in STD_LOGIC;
        y0 : out STD_LOGIC
    );
end mux_4_1;

-- Definizione architettura del modulo mux_4_1 con una descrizione strutturale.
architecture structural of mux_4_1 is
    signal u0 : STD_LOGIC := '0';
    signal u1 : STD_LOGIC := '0';

    component mux_2_1
        port(
            a0          : in STD_LOGIC;
            a1          : in STD_LOGIC;
            s           : in STD_LOGIC;
            y           : out STD_LOGIC
        );
    end component;

    begin
        mux0: mux_2_1
            Port map(
                a0          => b0,
                a1          => b1,
                s           => s0,
                y           => u0
            );
        mux1: mux_2_1
            Port map(
                a0          => b2,
                a1          => b3,
                s           => s0,
                y           => u1
            );
        mux2: mux_2_1
            Port map(
                a0          => u0,
                a1          => u1,
                s           => s1,
                y           => y0
            );
    end structural;

```

1.5.3 | Punto 1 - Multiplexer 16 a 1

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_16_1 is
    port(
        I : in STD_LOGIC_VECTOR (0 to 15);
        c : in STD_LOGIC_VECTOR (3 downto 0);
        y : out STD_LOGIC
    );
end mux_16_1;

architecture Structural of mux_16_1 is
    signal u : STD_LOGIC_VECTOR (0 to 3) := (others => '0');

```

```

component mux_4_1
  port(
    b0 : in STD_LOGIC;
    b1 : in STD_LOGIC;
    b2 : in STD_LOGIC;
    b3 : in STD_LOGIC;
    s0 : in STD_LOGIC;
    s1 : in STD_LOGIC;
    y0 : out STD_LOGIC
  );
end component;

begin
  mux0: mux_4_1
    Port map(
      b0      => I(0),
      b1      => I(1),
      b2      => I(2),
      b3      => I(3),
      s0      => c(0),
      s1      => c(1),
      y0      => u(0)
    );
  mux1: mux_4_1
    Port map(
      b0      => I(4),
      b1      => I(5),
      b2      => I(6),
      b3      => I(7),
      s0      => c(0),
      s1      => c(1),
      y0      => u(1)
    );
  mux2: mux_4_1
    Port map(
      b0      => I(8),
      b1      => I(9),
      b2      => I(10),
      b3      => I(11),
      s0      => c(0),
      s1      => c(1),
      y0      => u(2)
    );
  mux3: mux_4_1
    Port map(
      b0      => I(12),
      b1      => I(13),
      b2      => I(14),
      b3      => I(15),
      s0      => c(0),
      s1      => c(1),
      y0      => u(3)
    );
  mux4: mux_4_1
    Port map(
      b0      => u(0),
      b1      => u(1),
      b2      => u(2),
      b3      => u(3),
      s0      => c(2),
      s1      => c(3),
      y0      => y
    );

```

```
end Structural;
```

1.5.4 | Punto 2 - Demux 1 a 4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity demux_1_4 is
    Port (
        d : in std_logic;
        s : in std_logic_vector (1 downto 0);
        u : out std_logic_vector (0 to 3)
    );
end demux_1_4;

architecture Dataflow of demux_1_4 is
begin
    WITH s SELECT
        u <=      (d & "000")          WHEN "00",
                  "0" & d & "00"    WHEN "01",
                  "00" & d & "0"   WHEN "10",
                  "000" & d       WHEN "11",
                  "----"           WHEN others;
end Dataflow;
```

1.5.5 | Punto 2 - Bus 16 a 4

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

entity M16_D4 is Port (
    Pin: in std_logic_vector( 0 to 15):="0000000000000000";
    EnableM16: in std_logic_vector( 3 downto 0);
    EnableD4: in std_logic_vector( 1 downto 0);
    Output : out std_logic_vector( 3 downto 0)
);
end M16_D4;
architecture Structural of M16_D4 is
    signal outMux : std_logic;
    component mux_16_1 is port(
        I : in STD_LOGIC_VECTOR (0 to 15);
        c : in STD_LOGIC_VECTOR (3 downto 0);
        y : out STD_LOGIC
    );
    end component;
    component demux_1_4 is
        Port (
            d : in std_logic;
            s : in std_logic_vector (1 downto 0);
            u : out std_logic_vector (0 to 3)
        );
    end component;
begin
    mux_16_0: mux_16_1
        Port map(
            I => Pin,
            c => EnableM16,
            y => outMux

```

```

);
demux_4_0: demux_1_4
    Port map(
        d => outMux,
        s => EnableD4,
        u=>Output
    );
end Structural;

```

1.5.6 | Punto 3 - Sintesi

Per la sintesi è stato prodotto un file .xdc per mappare gli switch e i led con gli input e output definiti.

```

##Switches
set_property -dict { PACKAGE_PIN J15    IOSTANDARD LVCMOS33 }
[get_ports { EnableM16[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 }
[get_ports { EnableM16[1] }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13    IOSTANDARD LVCMOS33 }
[get_ports { EnableM16[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15    IOSTANDARD LVCMOS33 }
[get_ports { EnableM16[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN U11    IOSTANDARD LVCMOS33 }
[get_ports { EnableD4[1] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10    IOSTANDARD LVCMOS33 }
[get_ports { EnableD4[0] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 }
[get_ports { Output[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15    IOSTANDARD LVCMOS33 }
[get_ports { Output[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13    IOSTANDARD LVCMOS33 }
[get_ports { Output[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14    IOSTANDARD LVCMOS33 }
[get_ports { Output[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]

```

2 | Encoder BCD

2.1 | Traccia

Punto 1 Progettare, implementare in VHDL e testare mediante simulazione una rete che, data in ingresso una stringa binaria X di 10 bit X9 X8 X7 X6 X5 X4 X3 X2 X1 X0 che corrisponde alla rappresentazione decodificata di una cifra decimale (cioè, una rappresentazione in cui ogni stringa contiene un solo bit alto), fornisce in uscita la rappresentazione Y della cifra mediante codifica Binary-Coded Decimal (BCD).

Punto 2 Sintetizzare ed implementare su board il progetto dell'encoder BCD utilizzando gli switch per fornire la stringa X in ingresso, e i led per visualizzare Y.

Nel caso in cui si utilizzi una board dotata di soli 8 switch, è possibile sviluppare il progetto considerando X di soli 8 bit (la macchina sarà allora in grado di fornire in uscita la rappresentazione BCD delle cifre decimali da 0 a 7).

Punto 3 Utilizzare un display a 7 segmenti per visualizzare la cifra decimale codificata da Y (pilotare opportunamente i catodi del display per visualizzare la cifra).

2.2 | Descrizione Soluzione

La nostra soluzione prevede un arbitro che garantisce un'entrata lecita a un encoder, infatti l'arbitro è pensato come un blocco con 10 ingressi e 10 uscite con il compito di verificare se il msb sia pari a 1, se è così mette tutti i restanti bit a zero, in caso contrario controlla la seconda cifra più significativa e così via. Il blocco encoder prende in ingresso le uscite del blocco arbitro e genera in uscita la posizione del bit alto in ingresso; Per esempio se l'ingresso è "0000000100" l'uscita sarà "0011".

Infine poiché la nostra scheda presenta 16 switch, abbiamo deciso di utilizzarli per l'inserimento della stringa binaria.

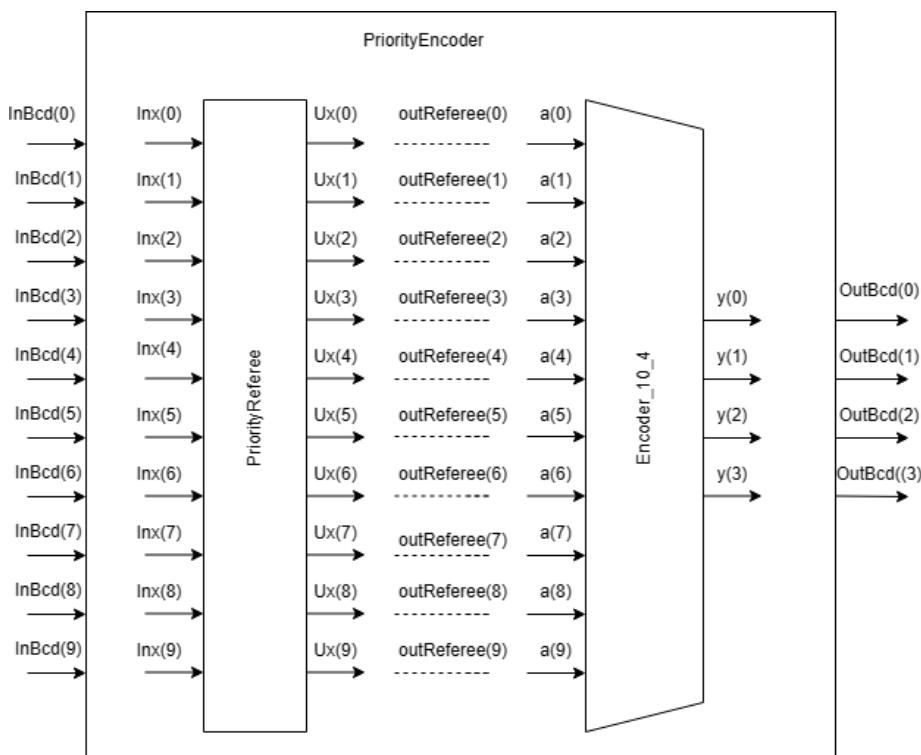
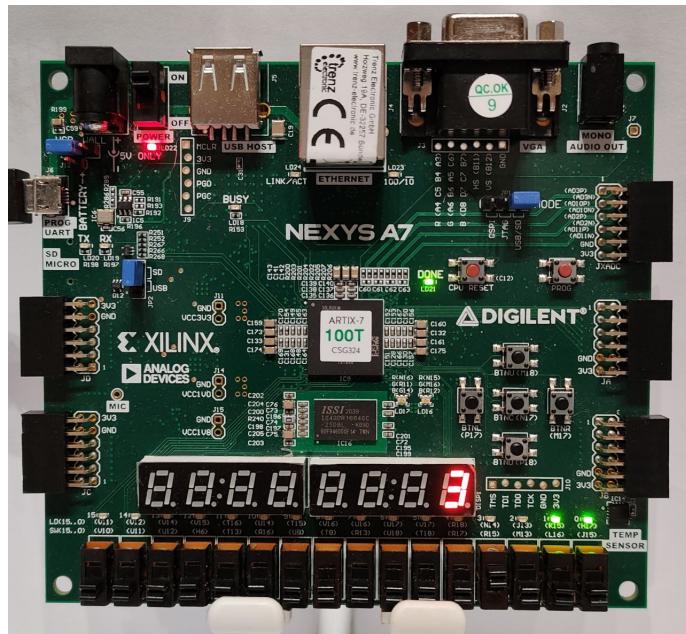


Figura 2.1: Struttura complessiva della macchina.

2.3 | Testbench

**Figura 2.2:** Testbench del BCD.**2.4 | Sintesi****Figura 2.3:** Sintesi del Encoder BCD.

2.5 | Codice

2.5.1 | Punto 1 - Arbitro Priorità

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PriorityReferee is Port (
    Inx: in std_logic_vector(9 downto 0);
    Ux: out std_logic_vector(9 downto 0)
);
end PriorityReferee;

architecture Dataflow of PriorityReferee is
begin
    Ux <= "1000000000" when Inx(9)='1' else
        "0100000000" when Inx(8)='1' else
        "0010000000" when Inx(7)='1' else
        "0001000000" when Inx(6)='1' else
        "0000100000" when Inx(5)='1' else
        "0000010000" when Inx(4)='1' else
        "0000001000" when Inx(3)='1' else
        "0000000100" when Inx(2)='1' else
        "0000000010" when Inx(1)='1' else
        "0000000001" when Inx(0)='1' else
        "-----";
end Dataflow;
```

2.5.2 | Punto 1 - Encoder 10 a 4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Encoder_10_4 is
    Port (
        a:in std_logic_vector(9 downto 0);
        y:out std_logic_vector(3 downto 0)
    );
end Encoder_10_4;

architecture Dataflow of Encoder_10_4 is

begin
    y <= "0000" when a="0000000001" else
        "0001" when a="0000000010" else
        "0010" when a="0000000100" else
        "0011" when a="0000001000" else
        "0100" when a="0000010000" else
        "0101" when a="0000100000" else
        "0110" when a="0001000000" else
        "0111" when a="0010000000" else
        "1000" when a="0100000000" else
        "1001" when a="1000000000" else
        "----";
end Dataflow;
```

2.5.3 | Punto 2 - Sintesi con Switch e Led

Anche in questo caso è stato prodotto un file .xdc per mappare opportunamente gli input e gli output.

```
##Switches
set_property -dict { PACKAGE_PIN J15    IO_STANDARD LVCMOS33 }
```

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

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

2.5.4 | Punto 3 - Display a 7 Segmenti

Questo componente sfrutta un contatore e un divisore di frequenza per gestire l'accensione dei catodi e anodi del display.

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

entity display_seven_segments is
    Generic(
        CLKIN_freq : integer := 100000000;
        CLKOUT_freq : integer := 500
    );
    Port ( CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            VALUE : in STD_LOGIC_VECTOR (31 downto 0);
            ENABLE : in STD_LOGIC_VECTOR (7 downto 0); -- decide quali cifre abilitare
            DOTS : in STD_LOGIC_VECTOR (7 downto 0); -- decide quali punti visualizzare
            ANODES : out STD_LOGIC_VECTOR (7 downto 0);
            CATHODES : out STD_LOGIC_VECTOR (7 downto 0));
end display_seven_segments;

architecture Structural of display_seven_segments is

signal counter : std_logic_vector(2 downto 0);
signal clock_filter_out : std_logic := '0';
```

```

COMPONENT counter_mod8
    PORT(
        clock : in STD_LOGIC;
        reset : in STD_LOGIC;
        enable : in STD_LOGIC;
        counter : out STD_LOGIC_VECTOR (2 downto 0)
    );
END COMPONENT;

COMPONENT cathodes_manager
    PORT(
        counter : IN std_logic_vector(2 downto 0);
        value : IN std_logic_vector(31 downto 0);
        dots : IN std_logic_vector(7 downto 0);
        cathodes : OUT std_logic_vector(7 downto 0)
    );
END COMPONENT;

COMPONENT anodes_manager
    PORT(
        counter : IN std_logic_vector(2 downto 0);
        enable_digit : IN std_logic_vector(7 downto 0);
        anodes : OUT std_logic_vector(7 downto 0)
    );
END COMPONENT;

COMPONENT clock_filter
    GENERIC(
        CLKIN_freq : integer := 100000000;
        CLKOUT_freq : integer := 500
    );
    PORT(
        clock_in : IN std_logic;
        reset : in STD_LOGIC;
        clock_out : OUT std_logic
    );
END COMPONENT;
begin
    --il clock filter genera un segnale di abilitazione per il contatore mod8 che viene usato
    --come segnale di conteggio e quindi di fatto fornisce la frequenza con cui viene modificata
    --la cifra da mostrare

    clk_filter: clock_filter GENERIC MAP(
        CLKIN_freq => CLKIN_freq,
        CLKOUT_freq => CLKOUT_freq
    )
    PORT MAP(
        clock_in => CLK,
        reset => RST,
        clock_out => clock_filter_out
    );

    counter_instance: counter_mod8 port map(
        clock => CLK,
        enable => clock_filter_out,
        reset => RST,

```

```

        counter => counter
);
--il valore di conteggio viene usato dal gestore dei catodi e degli anodi per
--selezionare l'anodo da accendere e il suo rispettivo valore
cathodes_instance: cathodes_manager port map(
    counter => counter,
    value => value,
    dots => dots,
    cathodes => cathodes
);

anodes_instance: anodes_manager port map(
    counter => counter,
    enable_digit => enable,
    anodes => anodes
);

end Structural;

```

2.5.5 | Punto 3 - Sintesi con Display

```

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

##Switches
set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 }
[get_ports { input[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 }
[get_ports { input[1] }]; #IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 }
[get_ports { input[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 }
[get_ports { input[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 }
[get_ports { input[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 }
[get_ports { input[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 }
[get_ports { input[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 }
[get_ports { input[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8       IOSTANDARD LVCMOS18 }
[get_ports { input[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8       IOSTANDARD LVCMOS18 }
[get_ports { input[9] }]; #IO_25_34 Sch=sw[9]

## LEDs
set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 }
[get_ports { OutLed[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 }
[get_ports { OutLed[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMOS33 }
[get_ports { OutLed[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 }

```

```

[get_ports { OutLed[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18    IOSTANDARD LVCMOS33 }
[get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]

##7 segment display
set_property -dict { PACKAGE_PIN T10    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[6] }]; #IO_L4P_TO_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15    IOSTANDARD LVCMOS33 }
[get_ports { cathodes_out[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17    IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18    IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9     IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14    IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14    IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14    IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2     IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13    IOSTANDARD LVCMOS33 }
[get_ports { anodes_out[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

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

```

3 | Riconoscitore di Sequenze

3.1 | Traccia

Punto 1 Progettare, implementare in VHDL e testare mediante simulazione una macchina in grado di riconoscere la sequenza 1001. La macchina prende in ingresso un segnale binario i che rappresenta il dato, un segnale CLK di temporizzazione e un segnale M di modo, che 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 4.
- se $M=1$, la macchina valuta i bit seriali uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta.

Punto 2 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 pulsanti $B1$ 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 | Descrizione Soluzione

Il comportamento della macchina è stato definito usando un automa a stati finiti, in questo caso è stata definita come macchina del Mealy ed è possibile trovarne la rappresentazione nella figura 3.1.

Da notare che la macchina è costruita per operare solo in una modalità alla volta, per cambiare modalità va resettata e modificato il bit indicante il modo.

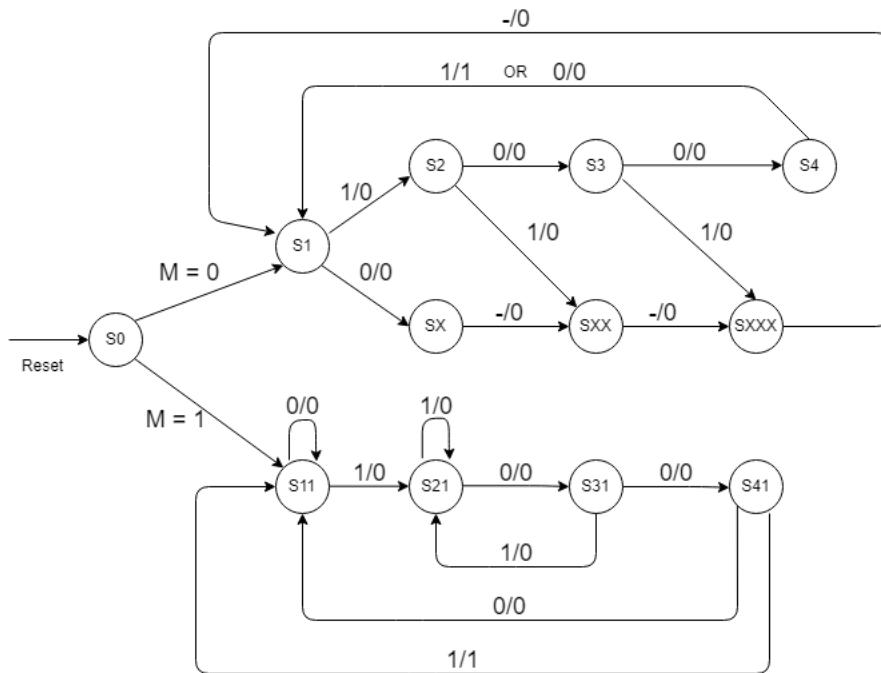


Figura 3.1: Diagramma di stato del riconoscitore di sequenze

Nell'automa per $M = 0$, la modalità in cui si considerano 4 bit alla volta, si può notare che abbiamo definito degli stati s_x , s_{xx} e così via, ma anche degli stati come s_1 e s_2 , questo serve per distinguere se sto considerando una sequenza errata o il caso in cui sto considerando la sequenza corretta.

Questo accorgimento è necessario poiché considerando 4 bit alla volta, l'automa andrà avanti anche se è entrato un bit non corretto.

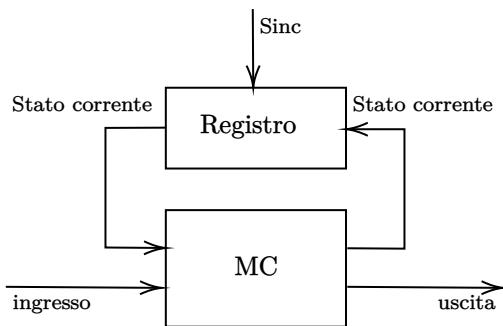


Figura 3.2: modello di Huffman.

Invece nell'automa $M = 1$, in cui valutiamo un bit alla volta, non c'è bisogno di considerare uno stato corretto e uno stato errato, infatti in questo caso rappresentiamo l'automa con soli 4 stati e l'uscita si alza se nello stato finale arriva un 1 e viene quindi riconosciuta la sequenza 1001.

Per realizzare la macchina ci siamo basati sullo schema in figura 3.2, sono stati definiti due processi: il primo definisce l'automa come macchina di mealy quindi in base allo stato in cui si trova indica quale sarà lo stato prossimo e l'uscita, quindi è a tutti gli effetti una macchina combinatoria.

Mentre il secondo definisce un registro per ricordare lo stato permettendo alla macchina di evolvere, inoltre riceve il clock in ingresso come abilitazione per scandire il comportamento della macchina.

3.2.1 | Punto 2

È stato necessario mappare gli switch per il tasto modo e introdurre un tasto per inserire i bit. Idealmente immaginando che il bottone sia attivo-alto esso passa istantaneamente dal valore 0 al valore 1 appena viene premuto, ma nella realtà sono presenti delle oscillazioni che possono avere caratteristiche e durate anche sensibilmente differenti in base alla tecnologia realizzativa del bottone.

Per l'occhio umano è impossibile notare tale dinamica, mentre per un qualsiasi sistema in grado di campionare il segnale ad alte frequenze le oscillazioni vengono rilevate.

In mancanza di una adeguata gestione di questo fenomeno si potrebbero osservare comportamenti inattesi nella logica del sistema, infatti è possibile affermare che il button bouncing costituisce un problema quando il segnale associato al bottone viene utilizzato per far evolvere lo stato di un sistema, poiché a causa dell'alta frequenza di campionamento si va col rilevare il segnale alto in più istanti successivi.

Aggiungiamo quindi nel nostro progetto il componente ButtonDebouncer, questo componente prende in input il segnale proveniente dal bottone e genera un segnale "ripulito" che presenta un impulso della durata di un colpo di clock per segnalare l'avvenuta pressione del bottone.

Il debouncer implementa un semplice automa con 2 stati, si parte da NOT_PRESSED e appena si rileva $BTN = 1$, si va in PRESSED dove si aspettano M colpi di clock in modo da "superare" l'oscillazione.

Il valore di M va valutato in base al periodo del CLK (che in questo caso è pari a 10ns) e all'intervallo di tempo in cui si ha l'oscillazione del bottone (che assumiamo pari a 6.5 ms = 6500 us = 6500000 ns), quindi M sarà pari al rapporto tra l'intervallo di tempo in cui si ha l'oscillazione del bottone e il periodo di CLK ($6500000/10$, ovvero contiamo 650000 colpi di clock).

3.3 | Testbench

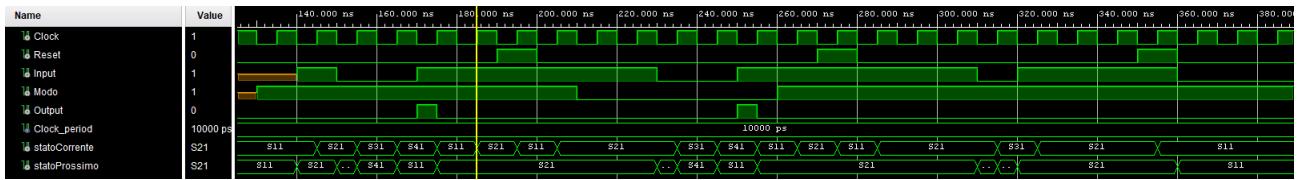


Figura 3.3: Testbench riconoscitore caso 1.

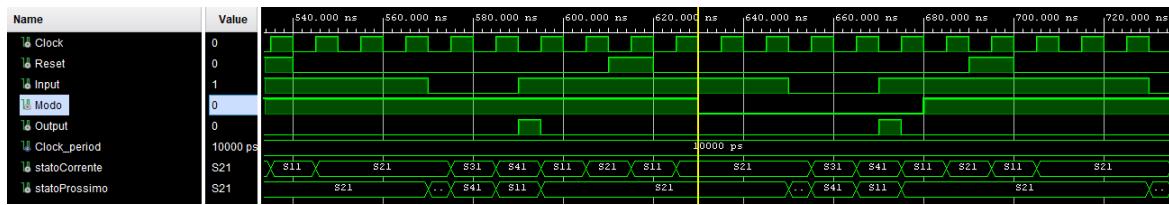


Figura 3.4: Testbench riconoscitore caso 2.

3.4 | Sintesi

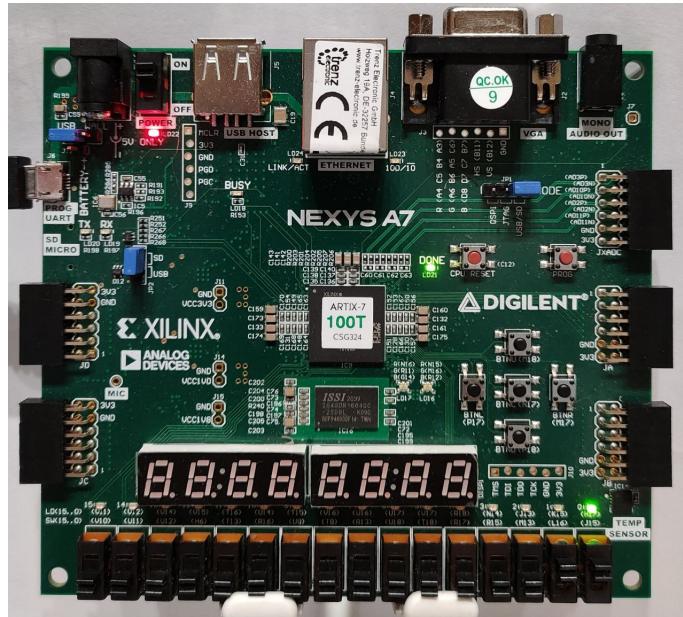


Figura 3.5: Sintesi del riconoscitore.

3.5 | Codice

3.5.1 | Punto 1 - Riconoscitore

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

entity Riconoscitore is
  Port (
    Clock : in std_logic;
    Reset : in std_logic;
```

```

InDato : in std_logic;
Modo : in std_logic;
Y: out std_logic
);
end Riconoscitore;
architecture Behavioral of Riconoscitore is

--definisco gli stati
type Stato is(S0,S1,S2,S3,S4,SX,SXX,SXXX,S11,S21,S31,S41);
signal statoCorrente : Stato :=S0;
signal statoProssimo : Stato;

begin
-- definisco il comportamento del mio sistema
statoUscita :process(statoCorrente,InDato)
begin
case statoCorrente is

--vado a descrivere il comportamento della macchina con Modo=0
when S1 =>
    if(InDato='1') then
        statoProssimo <=S2;
        y <='0';
    else
        statoProssimo <=SX;
        y <='0';
    end if;
when S2 =>
    if(InDato='0') then
        statoProssimo <=S3;
        y <='0';
    else
        statoProssimo <=SXX;
        y <='0';
    end if;

when S3 =>
    if(InDato='0') then
        statoProssimo <=S4;
        y <='0';
    else
        statoProssimo <=SXXX;
        y <='0';
    end if;

when S4 =>
    if(InDato='1') then
        statoProssimo <=S1;
        y <='1';
    else
        statoProssimo <=S1;
        y <='0';
    end if;

--gestisco gli stati che non mi portano all'uscita 1
when SX =>
    if(InDato='1' or InDato='0') then
        statoProssimo <=SXX;

```

```

        y <='0';
    end if;
when SXX =>
    if(InDato='1' or InDato='0') then
        statoProssimo <=SXXX;
        y <='0';
    end if;

when SXXX =>
    if(InDato='1' or InDato='0') then
        statoProssimo <=S1;
        y <='0';
    end if;

--vado a descrivere il comportamento della macchina con Modo=1

when S11 =>
    if(InDato='1') then
        statoProssimo <=S21;
        y <='0';
    else
        statoProssimo <=S11;
        y <='0';
    end if;
when S21 =>
    if(InDato='0') then
        statoProssimo <=S31;
        y <='0';
    else
        statoProssimo <=S21;
        y <='0';
    end if;

when S31 =>
    if(InDato='0') then
        statoProssimo <=S41;
        y <='0';
    else
        statoProssimo <=S21;
        y <='0';
    end if;

when S41 =>
    if(InDato='1') then
        statoProssimo <=S11;
        y <='1';
    else
        statoProssimo <=S11;
        y <='0';
    end if;

when others =>
    if(Modo='1') then
        statoProssimo <=S11;
    else
        statoProssimo <=S1;
    end if;

```

```

        y<='0';

    end case;
end process;

--comportamento relativo alla memorizzazione degli stati
Memoria :process(Clock)
begin
    if( Clock'event and Clock = '1' ) then
        if(Reset = '1') then
            statoCorrente <= S0;
        if(Modo='0') then
            statoCorrente <= S1;
        else
            statoCorrente <= S11;
        end if;
    else
        statoCorrente <= statoProssimo;
    end if;
end if;
end process;
end Behavioral;

```

3.5.2 | Punto 2 - Button Debouncer

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ButtonDebouncer is
    generic (
        CLK_period: integer := 10;
        -- periodo del clock della board 10 nanosecondi
        btn_noise_time: integer := 6500000
        --intervallo di tempo in cui si ha l'oscillazione del bottone
        --assumo che duri 6.5ms=6500microsec=6500000ns
    );
    Port (
        RST : in STD_LOGIC;
        CLK : in STD_LOGIC;
        BTN : in STD_LOGIC;
        CLEARED_BTN : out STD_LOGIC);
end ButtonDebouncer;

architecture Behavioral of ButtonDebouncer is

type stato is (NOT_PRESSED, PRESSED);
signal BTN_state : stato := NOT_PRESSED;

constant max_count : integer := btn_noise_time/CLK_period;
-- 6500000/10= conto 650000 colpi di clock

begin
deb: process (CLK)
variable count: integer := 0;
begin
    if rising_edge(CLK) then
        if( RST = '1') then
            BTN_state <= NOT_PRESSED;
            CLEARED_BTN <= '0';

```

```

        else
            case BTN_state is
                when NOT_PRESSED =>  CLEARED_BTN<= '0';
                    if( BTN = '1' ) then
                        BTN_state <= PRESSED;
                    else
                        BTN_state <= NOT_PRESSED;
                    end if;
                when PRESSED =>
                    if(count = max_count -1) then
                        count:=0;
                        CLEARED_BTN <= '1';
                        BTN_state <= NOT_PRESSED;
                    else
                        count:= count+1;
                        BTN_state <= PRESSED;
                    end if;
                when others =>
                    BTN_state <= NOT_PRESSED;
            end case;
        end if;
    end if;
end process;
end Behavioral;

```

3.5.3 | Punto 2 - Rete Complessiva

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SistemaComplessivo is
    Port (
        Clk: in std_logic;
        Rst: in std_logic;
        Btn_Ingresso: in std_logic;
        Ingresso: in std_logic;
        Modo: in std_logic;
        Btn_Modo: in std_logic;
        Uscita: out std_logic
    );
end SistemaComplessivo;

architecture Structural of SistemaComplessivo is

    signal Btn_Modo_cl: std_logic;
    signal Btn_Ingresso_cl: std_logic;

    component ButtonDebouncer is
        generic (
            CLK_period: integer := 10;
            -- periodo del clock della board 10 nanosecondi
            btn_noise_time: integer := 6500000
            --intervallo di tempo in cui si ha l'oscillazione del bottone
        );
        Port (
            RST : in STD_LOGIC;
            CLK : in STD_LOGIC;
            BTN : in STD_LOGIC;
            CLEARED_BTN : out STD_LOGIC
        );
    end component;

```

```

    );
end component;

component Riconoscitore
  Port (
    Clock : in std_logic;
    Reset : in std_logic;
    Btn_InData: in std_logic;
    InData : in std_logic;
    Btn_Modo: in std_logic;
    Modo : in std_logic;
    Y: out std_logic
  );
end component;

begin
  ModoClear: ButtonDebouncer
    generic map (
      CLK_period => 10,
      btn_noise_time => 550000000
    )
    Port map (
      RST => Rst,
      CLK => Clk,
      BTN => Btn_Modo,
      CLEARED_BTN => Btn_Modo_cl
    );

  IngressoClear: ButtonDebouncer
    generic map (
      CLK_period => 10,
      btn_noise_time => 550000000
    )
    Port map (
      RST => Rst,
      CLK => Clk,
      BTN => Btn_Ingresso,
      CLEARED_BTN => Btn_Ingresso_cl
    );
  RiconoscitoreSeq: Riconoscitore
    port map(
      Clock => Clk,
      Reset => Rst,
      InData => Ingresso,
      Btn_InData => Btn_Ingresso_cl,
      Modo => Modo,
      Btn_Modo => Btn_Modo_cl,
      Y => Uscita
    );
end Structural;

```

3.5.4 | Punto 2 - Mapping su scheda

```

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

```

```
##Switches
set_property -dict { PACKAGE_PIN J15    IO_STANDARD LVCMOS33 }
[get_ports { Ingresso }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16    IO_STANDARD LVCMOS33 }
[get_ports { Modo }]; #IO_L3N_TO_DQS_EMCLK_14 Sch=sw[1]

## LEDs
set_property -dict { PACKAGE_PIN H17    IO_STANDARD LVCMOS33 } [get_ports { Uscita }];
#IO_L18P_T2_A24_15 Sch=led[0]

##Buttons
set_property -dict { PACKAGE_PIN N17    IO_STANDARD LVCMOS33 }
[get_ports { Rst }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN P17    IO_STANDARD LVCMOS33 }
[get_ports { Btn_Modo }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17    IO_STANDARD LVCMOS33 }
[get_ports { Btn_Ingresso }];#IO_L10N_T1_D15_14 Sch=btnr
```

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. Il componente deve essere realizzato utilizzando sia:

- Un approccio **comportamentale**
- Un approccio **strutturale**

Nota: il numero di bit del registro X e i valori che può assumere il parametro Y possono essere scelti dallo studente (ad es. X=8 e Y=1,2).

4.2 | Descrizione Soluzione

Per realizzare il registro a scorrimento in maniera comportamentale abbiamo utilizzato un process e siamo andati a descrivere con dei semplici if tutte le caratteristiche del comportamento della macchina.

Per realizzare il registro a scorrimento in maniera strutturale abbiamo deciso che il segnale di abilitazione è fornito in ingresso a tutti i componenti in parallelo, mentre il dato si propaga da sinistra verso destra. Inoltre supponendo di partire da una situazione iniziale in cui i FF sono di tipo D è importante sottolineare che debbano essere sensibili solo ad una variazione, ad esempio quella del fronte di discesa del clock: se il sistema fosse di tipo latch non funzionerebbe sempre, perché dato l'ingresso pari ad 1 il primo flip-flop acquisirebbe 1 e dopo un certo intervallo temporale tutti gli altri flip-flop starebbero memorizzando 1. Nel nostro caso possiamo shiftare sia da destra verso sinistra che viceversa, inoltre è possibile decidere se effettuare uno shift di una o due posizioni, per la nostra soluzione abbiamo inizialmente considerato uno shift register composto da 4 FF; Lo schema è presente nella figura 4.1.

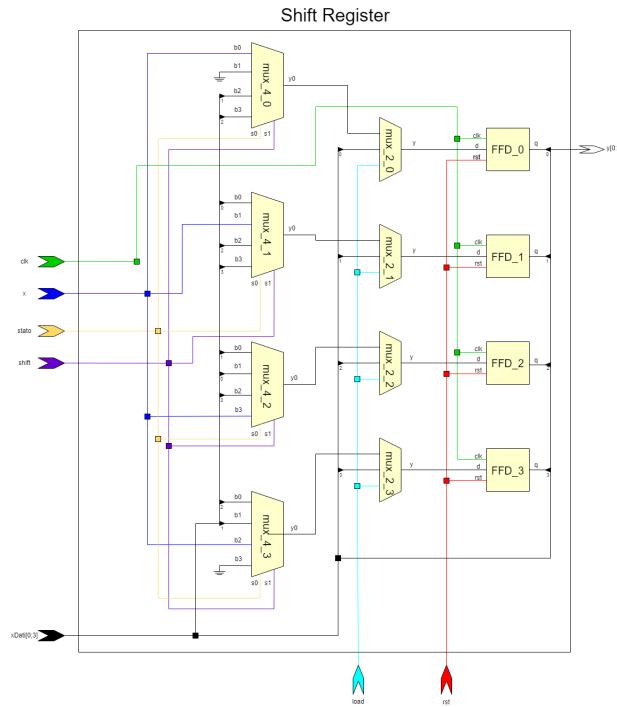


Figura 4.1: Schematico del shift register strutturale.

Il sistema complessivo riceve in ingresso il segnale x, il quale rappresenta un nuovo valore che deve essere inserito all'interno del registro, mentre xDati rappresenta un ingresso parallelo, quindi è possibile costruire un generatore di segnali.

Il segnale di LOAD permette di scegliere se caricare il valore del segnale xDati oppure no, mentre i segnali Stato e Shift ci permettono di selezionare la modalità di funzionamento del registro, in particolare:

entrambi i segnali possono assumere valore 0 oppure 1 e, in base alla combinazione di valori, abbiamo le seguenti possibili situazioni:

- Stato = 0, Shift = 0: shift a destra di una posizione;
- Stato = 1, Shift = 0: shift a destra di due posizioni;
- Stato = 0, Shift = 1: shift a sinistra di una posizione;
- Stato = 1, Shift = 1: shift a sinistra di due posizioni;

Analizziamo la struttura interna del registro: vediamo la presenza di un primo multiplexer 4:1 che riceve in ingresso 4 segnali e in base ai valori di Shift ed Stato produce un'unica uscita, la quale andrà in ingresso ad un secondo multiplexer 2:1, insieme al bit più significativo del segnale DATO.

Questo multiplexer 2:1 produrrà l'uscita sulla base del valore del segnale di LOAD, la quale andrà in ingresso al primo flip-flop.

A questo punto, il flip-flop manda l'uscita verso il multiplexer 4:1 successivo se stiamo considerando lo shift di una sola posizione verso destra, oppure la manda in ingresso al multiplexer successivo al secondo flip-flop se stiamo considerando uno shift verso destra di due posizioni.

Un analogo discorso vale per i restanti componenti e invece vale un discorso duale se stiamo considerando lo shift verso sinistra.

4.3 | Testbench

Sono stati effettuati vari test, identici sia per lo shift register pensato in comportamentale che in strutturale.

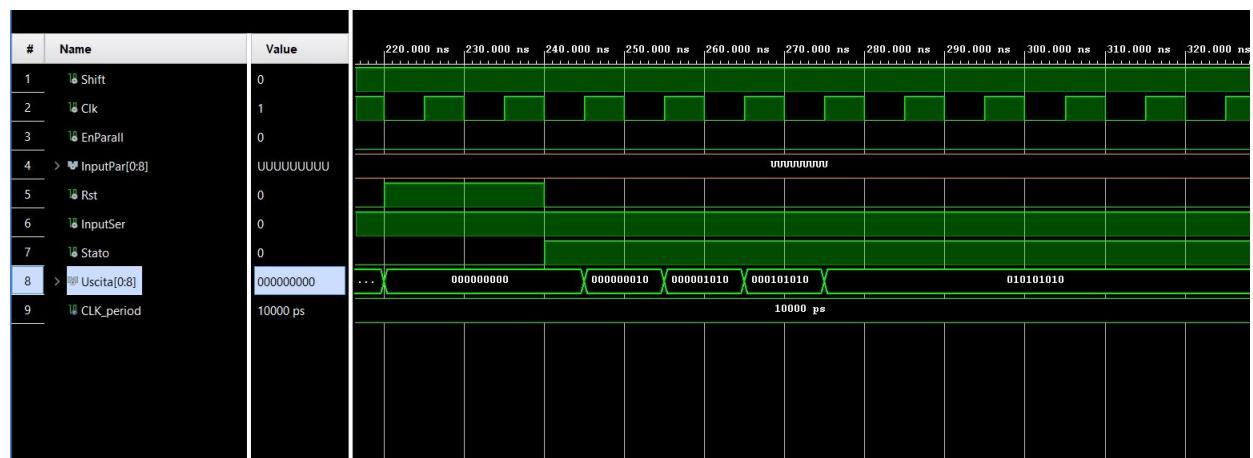


Figura 4.2: Test per lo shift da destra verso sinistra di 2 posizioni.



Figura 4.3: Test per lo shift da sinistra verso destra di una posizione.



Figura 4.4: Test per lo shift da destra verso sinistra di una posizione.



Figura 4.5: Test per lo shift da sinistra verso destra di 2 posizioni.

4.4 | Codice

4.4.1 | Approccio Comportamentale

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ShiftRegister is
    generic (N : integer := 9);
    port(
        x : in std_logic;
        --dato in input seriale
        xDati: in std_logic_vector (0 to N-1);
        --dato in input parallelo
        load: in std_logic;
        --abilitazione ingresso in parallelo
        clk : in std_logic;
        rst : in std_logic;
        stato : in std_logic;
        -- se è 0-->shift a destra;
        --se è 1-->shift due posizioni alla volta;
        shift: in std_logic;
        -- se è 0-->shift a destra;
        -- se è 1-->shift a sinistra;
        y: out std_logic_vector (0 to N-1)
    );
end ShiftRegister;

```

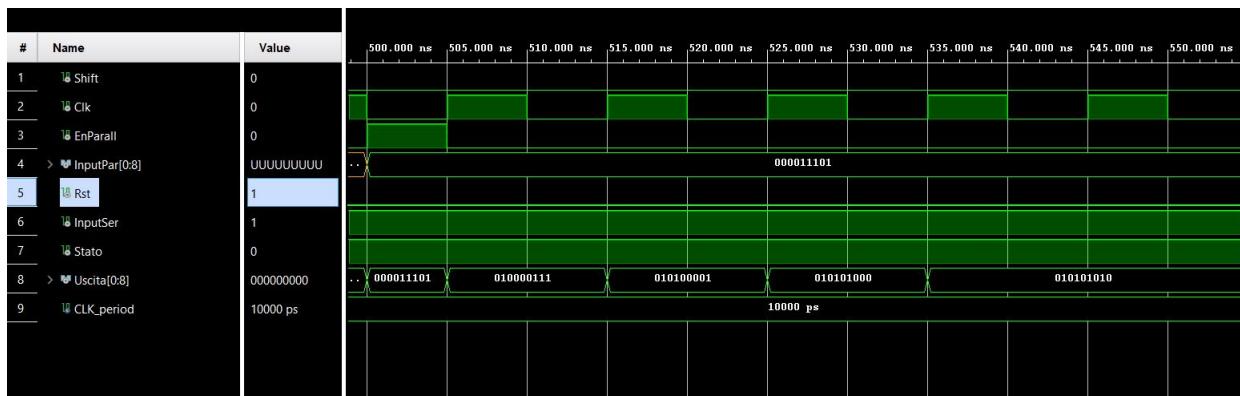


Figura 4.6: Test per il caricamento parallelo con shift ogni 2 posizioni.

```

architecture behavioral of ShiftRegister is
    signal T : std_logic_vector (0 to N-1);
begin
ShiftReg : process(clk, rst)
begin
    if (rst = '1') then
        T <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        if (load = '1') then
            T <=xDati;
        else
            --SHIFT A Sinistra una posizione per volta
            if (shift = '1' and stato = '0') then
                T(N-1) <= x; -- se N=4 ----> T(3)=x;
                T(0 to N-2) <= T(1 to N-1);
                -- se N=4----> pongo T(0) <= T(1),T(1) <= T(2),T(2) <= T(3)
            --SHIFT A Sinistra due posizioni per volta
            elsif (shift = '1' and stato = '1') then
                -- T(N-1) <='0'; --pongo l'ultimo bit N-1 a 0
                T(N-2) <= x;
                -- se N=4 ----> T(2)=x, pertanto il secondo bit ad x in maniera tale da avere --x0
                T(0 to N-3) <= T(2 to N-1);
                -- se N=4----> pongo T(0) <= T(2),T(1) <= T(3)
            --SHIFT A Destra una posizione per volta
            elsif (shift = '0' and stato = '0') then
                T(1 to N-1) <= T(0 to N-2);
                -- se N=4----> pongo T(1) <= T(0),T(2) <= T(1),T(3) <= T(2)
                T(0) <= x;
            --SHIFT A Destra due posizioni per volta
            elsif (shift = '0' and stato = '1') then
                T(2 to N-1) <= T(0 to N-3);
                -- se N=4----> pongo T(2) <= T(0),T(3) <= T(1)
                T(1) <= x;
                --T(0) <= '0';
                --con queste due assegnazione pongo il secondo bit ad x in maniera tale da avere 0x
            end if;
        end if;
    end process;

    y <= T;
end behavioral;

```

4.4.2 | Approccio Strutturale

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ShiftRegister is
    generic (N : integer := 4);
    port(
        x : in std_logic;
        --dato in input seriale
        xDati: in std_logic_vector (0 to N-1);
        --dato in input parallelo
        load: in std_logic;
        -- abilitazione ingresso in parallelo
        clk : in std_logic;
        rst : in std_logic;
        stato : in std_logic;
        -- se è 0-->shifta una posizione alla volta;
        --se è 1-->shifta due posizioni alla volta;
        shift: in std_logic;
        -- se è 0-->shifta a destra;
        -- se è 1-->shifta a sinistra;
        y: out std_logic_vector (0 to N-1)
    );
end ShiftRegister;

architecture Structural of ShiftRegister is
    signal outMux4:std_logic_vector(0 to N-1);
    signal outMux2:std_logic_vector(0 to N-1);
    signal outFFd:std_logic_vector(0 to N-1);
    component FlipFlopD
        port (
            clk: in std_logic;
            rst: in std_logic;
            d: in std_logic;
            q: out std_logic
        );
    end component;
    component mux_4_1 is
        port(
            b0 : in STD_LOGIC;
            b1 : in STD_LOGIC;
            b2 : in STD_LOGIC;
            b3 : in STD_LOGIC;
            s0 : in STD_LOGIC;
            s1 : in STD_LOGIC;
            y0 : out STD_LOGIC
        );
    end component;
    component mux_2_1 is
        Port (
            a0 : in STD_LOGIC;
            a1 : in STD_LOGIC;
            s : in STD_LOGIC;
            y : out STD_LOGIC
        );
    end component;

begin
    --COMP_0

```

```

Mux4_0:mux_4_1
Port Map(
    b0 =>x,
    b1 =>'0',
    b2 =>outFFd(1),
    b3 =>outFFd(2),
    s0 =>stato,
    s1 =>shift,
    y0 =>outMux4(0)
);
Mux2_0:mux_2_1
Port Map(
    a0 =>outMux4(0),
    a1 =>xDati(0),
    s => load,
    y =>outMux2(0)
);
FFD_0:FlipFlopD
Port Map(
    clk=>clk,
    rst =>rst,
    d =>outMux2(0),
    q =>outFFd(0)
);
--COMP_1
Mux4_1:mux_4_1
Port Map(
    b0 =>outFFd(0),
    b1 =>x,
    b2 =>outFFd(2),
    b3 =>outFFd(3),
    s0 =>stato,
    s1 =>shift,
    y0 =>outMux4(1)
);
Mux2_1:mux_2_1
Port Map(
    a0 =>outMux4(1),
    a1 =>xDati(1),
    s => load,
    y =>outMux2(1)
);
FFD_1:FlipFlopD
Port Map(
    clk=>clk,
    rst =>rst,
    d =>outMux2(1),
    q =>outFFd(1)
);
--COMP_2
Mux4_2:mux_4_1
Port Map(
    b0 =>outFFd(1),
    b1 =>outFFd(0),
    b2 =>outFFd(3),
    b3 =>x,

```

```

        s0 =>stato,
        s1 =>shift,
        y0 =>outMux4(2)
    );
Mux2_2:mux_2_1
Port Map(
    a0 =>outMux4(2),
    a1 =>xDati(2),
    s => load,
    y =>outMux2(2)

);
FFD_2:FlipFlopD
Port Map(
    clk=>clk,
    rst =>rst,
    d =>outMux2(2),
    q =>outFFd(2)
);

--COMP_3
Mux4_3:mux_4_1
Port Map(
    b0 =>outFFd(2),
    b1 =>outFFd(1),
    b2 =>x,
    b3 =>'0',
    s0 =>stato,
    s1 =>shift,
    y0 =>outMux4(3)
);
Mux2_3:mux_2_1
Port Map(
    a0 =>outMux4(3),
    a1 =>xDati(3),
    s => load,
    y =>outMux2(3)

);
FFD_3:FlipFlopD
Port Map(
    clk=>clk,
    rst =>rst,
    d =>outMux2(3),
    q =>outFFd(3)
);

y<=outFFd;

end Structural;

```

4.4.3 | Flip Flop D

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FlipFlopD is
    port (
        clk: in std_logic;

```

```
        rst: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end FlipFlopD;

architecture Behavioral of FlipFlopD is
signal qinternal: std_logic :='0';
begin
    process(clk, rst)
    begin
        if(rst='1') then
            qinternal<='0';
        elsif (clk 'event and clk='1') then
            qinternal<=d;
        end if;
    end process;
    q<= qinternal;
end Behavioral;
```

5 | Cronometro

5.1 | Traccia

Punto 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.

Punto 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).

Punto 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).

5.2 | Descrizione Soluzione

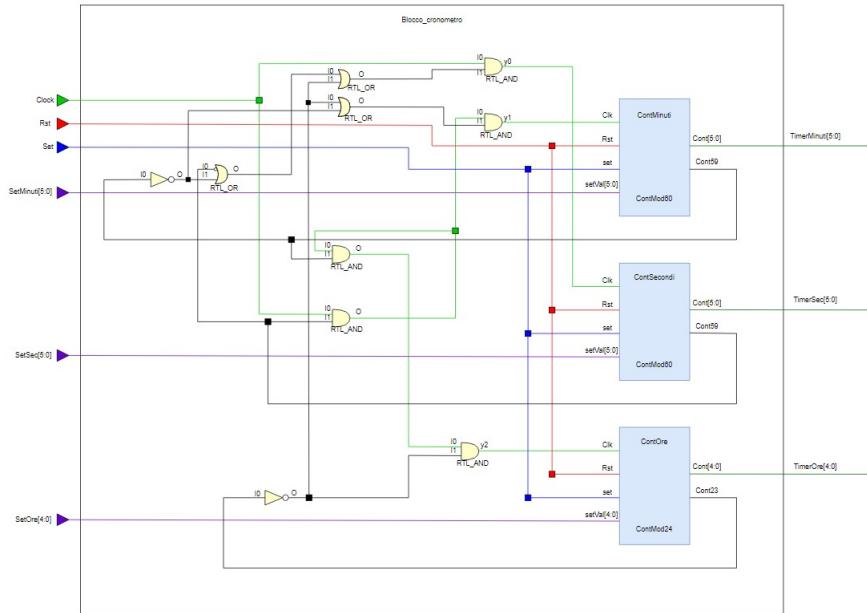


Figura 5.1: Schema completo del cronometro

In figura 5.1 è presente lo schema che descrive la soluzione al primo punto della traccia.

La struttura complessiva del cronometro è ottenuta per composizione di tre contatori: il contatore dei secondi, il contatore dei minuti ed il contatore delle ore; possiamo notare una serie di segnali di ingresso:

- Il segnale **clock** viene mappato con l'ingresso di conteggio di ogni contatore in parallelo sfruttando le porte logiche, questo permette di avere meno ritardo da propagazione e quindi un cronometro più preciso.

- Il segnale **reset** che viene mappato con i tre contatori per permettere l'azzeramento del cronometro.
- I segnali **SetSec**, **SetMinuti**, **SetOre** per impostare il valore iniziale del cronometro.
- Un segnale **set** dato in parallelo per abilitare l'impostazione di un orario sul cronometro.
- Ciascun contatore produce un'uscita che viene mappata opportunamente e così viene rappresentato l'orario.

5.2.1 | Punto 1 - Contatori

Sono stati realizzati due contatori per il progetto: un contatore modulo 24 utilizzato per le ore, visibile nella figura 5.2, ed un contatore modulo 60 per i minuti e i secondi, la cui struttura è nella figura 5.3. Inoltre abbiamo deciso che essendo un cronometro all'orario 23:59:59 si blocca e ha bisogno di un reset o di introdurre un nuovo orario.

Per realizzare questo comportamento entrambe le tipologie di contatori presentano un segnale di uscita utilizzato per indicare che si è arrivati all'ultimo valore disponibile, ad esempio per il mod60 abbiamo il segnale cont59, quindi è bastato elaborare delle condizioni logiche tra questi segnali.

Come si può vedere dagli schematici in figura i contatori sono stati realizzati in maniera strutturale partendo dai flip-flop T e dando in parallelo il clock a tutti, quindi in pratica i due contatori differiscono solo per il numero di flip-flop.

C'è da specificare che quest'ultimo è stato realizzato in maniera comportamentale e non prevede unicamente l'ingresso detto T e il clock, ma multipli ingressi in modo da realizzare più efficacemente e semplicemente la funzione di conteggio, reset e precaricamento.

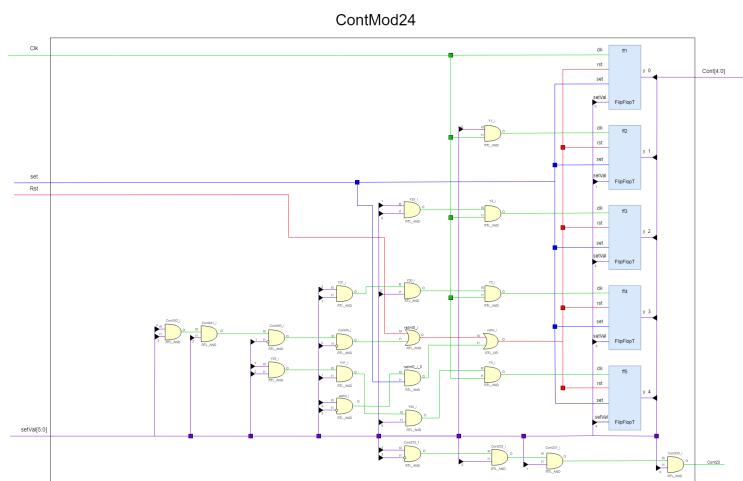


Figura 5.2: Contatore modulo 24.

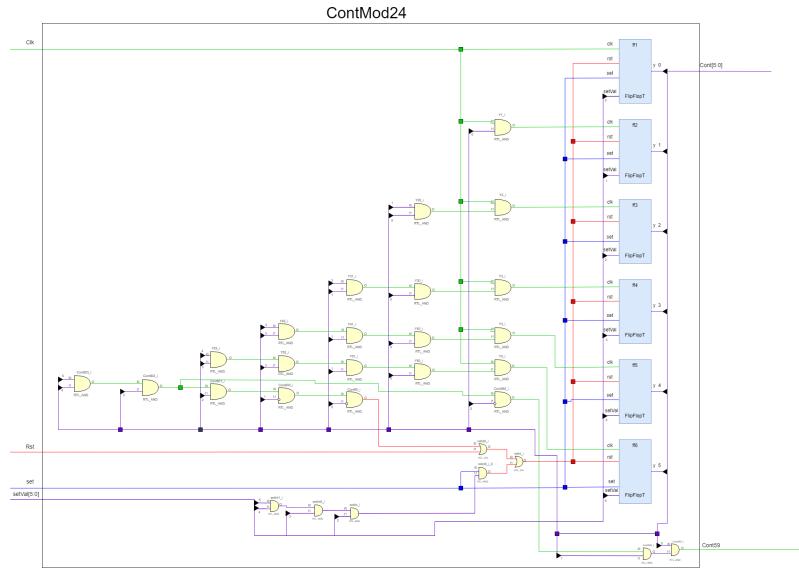
5.2.2 | Punto 2 - Struttura Complessiva

Poiché non abbiamo a disposizione 17 switch per settare il tempo in ore, minuti e secondi, abbiamo deciso di mappare gli switch in modo da poter impostare ore e minuti lasciando quindi i secondi a zero.

È stato implementato un convertitore per avere un'uscita in decimale per il display a 7 segmenti poiché la macchina fornitava un'uscita esadecimale. quindi abbiamo dovuto codificare 6 diverse cifre.

5.2.3 | Punto 3 - Struttura Complessiva

Per implementare la cattura degli intertempi abbiamo aggiunto due componenti, il blocco intertempo ed il blocco mantieni uscita; Questi blocchi saranno mappati con opportuni tasti della scheda in modo da poter selezionare 3 intertempi.

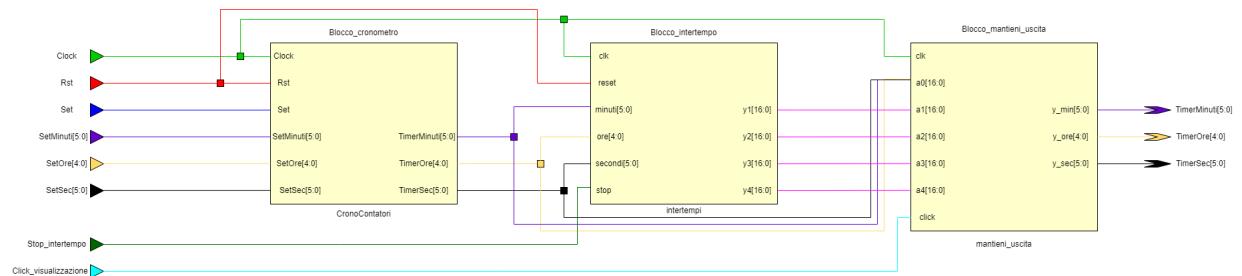
**Figura 5.3:** Contatore modulo 60.

Blocco intertempo Esso prende in ingresso l’uscita del blocco cronometro, ovvero il tempo corrente, e tramite il segnale stop_intertempo viene salvato di volta in volta un intertempo, si è deciso per un numero massimo di intertempi pari a 4 e se si salvano più intertempi di quelli previsti il sistema andrà a sovrascrivere i precedenti intertempi partendo dal primo; Le uscite del blocco intertempo sono i 4 possibili intertempi salvati.

Abbiamo predisposto anche un segnale di reset generale in modo tale da andare a resettare sia il blocco cronometro sia il blocco intertempi.

Blocco mantieni uscita Esso prende in ingresso sia il tempo corrente generato dal blocco cronometro sia i 4 intertempi posti come uscite del blocco intertempo.

Questo componente si occupa di gestire l’uscita da visualizzare tramite la pressione del bottone indicato come Click_visualizzazione, ad ogni pressione cambiamo l’uscita di questo blocco che parte dal tempo corrente per passare ai 4 intertempi salvati.

**Figura 5.4:** Struttura del cronometro al punto 3.

5.3 | Testbench

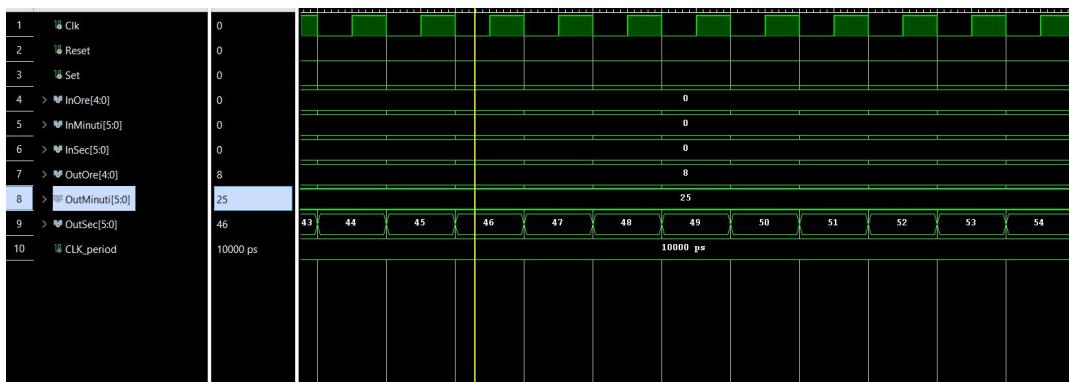


Figura 5.5: Scorrere del tempo.

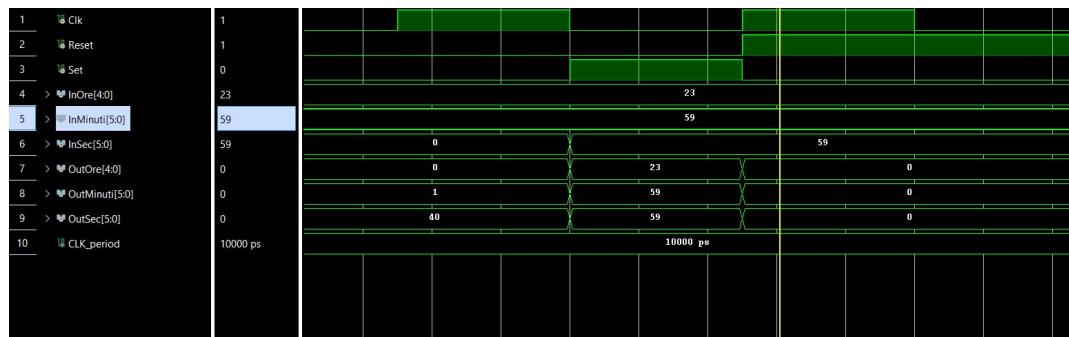


Figura 5.6: Blocco del cronometro all'orario 23:59:59.

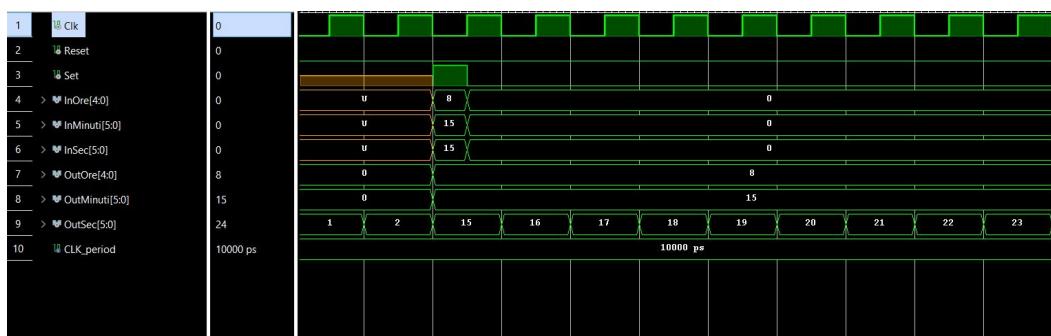


Figura 5.7: Settaggio di un orario sul Cronometro.

5.4 | Sintesi

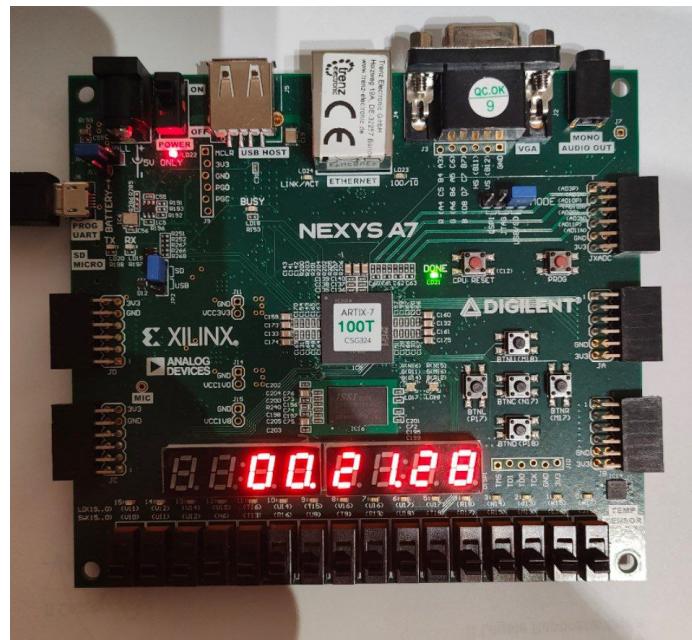


Figura 5.8: Cronometro su board.

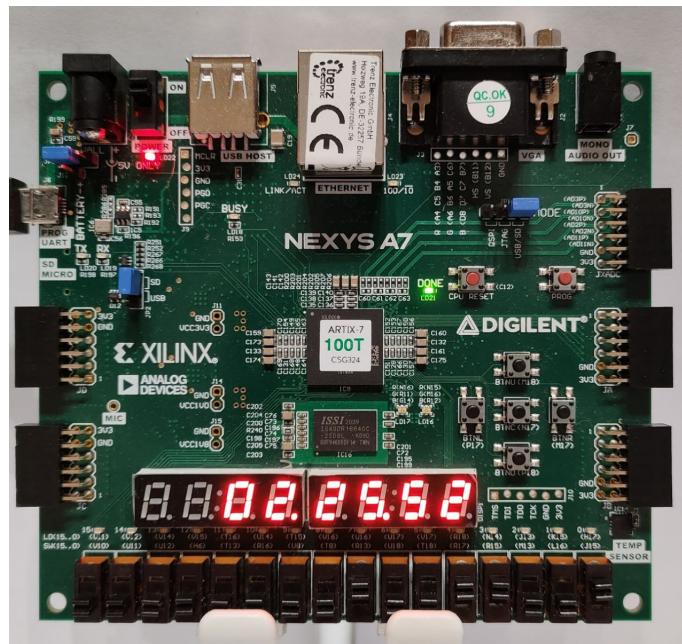


Figura 5.9: Altro esempio di cronometro su board.

5.5 | Codice VHDL

5.5.1 | Flip Flop T

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

entity FlipFlopT is
  Port (
```

```

        clk :in std_logic;
        rst :in std_logic;
        set :in std_logic;
        setVal :in std_logic;
        y :out std_logic
    );
end FlipFlopT;

architecture rtl of FlipFlopT is
    signal ty: std_logic;
begin
    fft:process(clk,rst,set)
    begin
        if(rst ='1') then
            ty <= '0';
        elsif(set ='1')then
            ty <=setVal;
        elsif(clk'event and clk='0')then
            ty <= not ty;
        end if;
    end process;
    y <= ty;
end rtl;

```

5.5.2 | Punto 1 - Contatore Modulo 24

--CONTATORE MOD24 parallelo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ContMod24 is
    Port (
        Clk : in std_logic;
        Rst :in std_logic;
        set :in std_logic;
        setVal : in std_logic_vector(4 downto 0);
        Cont : out std_logic_vector(4 downto 0);
        Cont23: inout std_logic
    );
end ContMod24;

architecture Structural of ContMod24 is
    component FlipFlopT is
        Port (
            clk :in std_logic;
            rst :in std_logic;
            set :in std_logic;
            setVal :in std_logic;
            y :out std_logic
        );
    end component;
    Signal S1: std_logic;
    Signal S2: std_logic;
    Signal S3: std_logic;
    Signal S4: std_logic;
    Signal S5: std_logic;

    Signal Y1: std_logic;

```

```

Signal Y2: std_logic;
Signal Y3: std_logic;
Signal Y4: std_logic;

Signal rstInt: std_logic;
Signal setInt: std_logic;
Signal Cont24 : std_logic;

begin
    rstInt <= Rst or Cont24 or (setInt and set) ;
    ff1: flipflopT
        port map(
            Clk ,
            rstInt,
            set,
            setVal(0),
            S1
        );

    Y1 <= S1 and Clk;
    ff2: flipflopT
        port map(
            Y1 ,
            rstInt,
            set,
            setVal(1),
            S2
        );

    Y2 <= S2 and S1 and Clk;
    ff3: flipflopT
        port map(
            Y2,
            rstInt,
            set,
            setVal(2),
            S3
        );
    Y3 <= S3 and S2 and S1 and Clk;
    ff4: flipflopT
        port map(
            Y3,
            rstInt,
            set,
            setVal(3),
            S4
        );
    Y4 <= S4 and S3 and S2 and S1 and Clk;
    ff5: flipflopT
        port map(
            Y4,
            rstInt,
            set,
            setVal(4),
            S5
        );

    Cont24 <= S5 and S4 and not(S3) and not(S2) and not(S1);

```

```

        setInt <= setVal(4) and setVal(3);
        Cont <= S5 & S4 & S3 & S2 & S1;
        Cont23 <= S5 and not(S4) and S3 and S2 and S1;

end Structural;
```

5.5.3 | Punto 1 - Contatore Modulo 60

--CONTATORE MOD60 Parallello

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ContMod60 is
    Port (
        Clk : in std_logic;
        Rst : in std_logic;
        set : in std_logic;
        setVal : in std_logic_vector(5 downto 0);
        Cont : inout std_logic_vector(5 downto 0);
        Cont59 : inout std_logic
    );
end ContMod60;

architecture Structural of ContMod60 is
    component FlipFlopT is
        Port (
            clk : in std_logic;
            rst : in std_logic;
            set : in std_logic;
            setVal : in std_logic;
            y : out std_logic
        );
    end component;
    Signal S1: std_logic;
    Signal S2: std_logic;
    Signal S3: std_logic;
    Signal S4: std_logic;
    Signal S5: std_logic;
    Signal S6: std_logic;

    Signal Y1: std_logic;
    Signal Y2: std_logic;
    Signal Y3: std_logic;
    Signal Y4: std_logic;
    Signal Y5: std_logic;
    Signal Y6: std_logic;

    Signal rstInt: std_logic;
    Signal setInt: std_logic;
    Signal Cont60 : std_logic;

begin
    rstInt <= Rst or Cont60 or (setInt and set) ;
    ff1: flipflopT
        port map(
            Clk ,
            rstInt,
            set,
```

```

        setVal(0),
        S1
    );

Y1 <= S1 and Clk;
ff2: flipflopT
port map(
    Y1 ,
    rstInt,
    set,
    setVal(1),
    S2
);

Y2 <= S2 and S1 and Clk;
ff3: flipflopT
port map(
    Y2,
    rstInt,
    set,
    setVal(2),
    S3
);
Y3 <= S3 and S2 and S1 and Clk;
ff4: flipflopT
port map(
    Y3,
    rstInt,
    set,
    setVal(3),
    S4
);
Y4 <= S4 and S3 and S2 and S1 and Clk;
ff5: flipflopT
port map(
    Y4,
    rstInt,
    set,
    setVal(4),
    S5
);
Y5 <= S5 and S4 and S3 and S2 and S1 and Clk;
ff6: flipflopT
port map(
    Y5,
    rstInt,
    set,
    setVal(5),
    S6
);
Cont60 <= S6 and S5 and S4 and S3 and not(S2) and not(S1);
setInt <= setVal(5) and setVal(4) and setVal(3) and setVal(2);
Cont59 <= S6 and S5 and S4 and not(S3) and S2 and S1;
Cont <= S6 & S5 & S4 & S3 & S2 & S1;

end Structural;

```

5.5.4 | Punto 1 - Cronometro

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CronoContatori is
Port (
    Clock : in std_logic;
    Rst :in std_logic;
    Set :in std_logic;
    SetOre : in std_logic_vector(4 downto 0);
    SetMinuti : in std_logic_vector(5 downto 0);
    SetSec : in std_logic_vector(5 downto 0);
    TimerOre : inout std_logic_vector(4 downto 0);
    TimerMinuti : inout std_logic_vector(5 downto 0);
    TimerSec : inout std_logic_vector(5 downto 0)

);
end CronoContatori;

architecture structural of CronoContatori is
    signal rstMnt : std_logic;
    signal rstOre : std_logic;
    signal Stop: std_logic;
    Signal Y0: std_logic;
    Signal Y1: std_logic;
    Signal Y2: std_logic;

    component ContMod24 is
        Port (
            Clk : in std_logic;
            Rst :in std_logic;
            set :in std_logic;
            setVal : in std_logic_vector(4 downto 0);
            Cont : out std_logic_vector(4 downto 0);
            Cont23 : inout std_logic

);
    end component;

    component ContMod60
        Port (
            Clk : in std_logic;
            Rst :in std_logic;
            set :in std_logic;
            setVal : in std_logic_vector(5 downto 0);
            Cont : inout std_logic_vector(5 downto 0);
            Cont59 : inout std_logic
);
    end component;
begin
Y0 <= Clock and (not(rstMnt) or not(rstOre) or not(stop));
ContSecondi: ContMod60
    Port map(
        Y0 ,
        Rst,
        set,
        SetSec,

```

```

        TimerSec,
        rstMnt
    );
Y1 <= (Clock and rstMnt)and (not(rstOre) or not(Stop));
ContMinuti: ContMod60
    Port map(
        Y1 ,
        Rst,
        set,
        SetMinuti,
        TimerMinuti,
        rstOre
    );
Y2 <= (Clock and rstMnt and rstOre) and not(Stop);
ContOre: ContMod24
    Port map(
        Y2 ,
        Rst,
        set,
        SetOre,
        TimerOre,
        Stop
    );

end structural;library IEEE;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CronoContatori is
Port (
    Clock : in std_logic;
    Rst :in std_logic;
    Set :in std_logic;
    SetOre : in std_logic_vector(4 downto 0);
    SetMinuti : in std_logic_vector(5 downto 0);
    SetSec : in std_logic_vector(5 downto 0);
    TimerOre :  inout std_logic_vector(4 downto 0);
    TimerMinuti :  inout std_logic_vector(5 downto 0);
    TimerSec :  inout std_logic_vector(5 downto 0)

);
end CronoContatori;

architecture structural of CronoContatori is
    signal rstMnt : std_logic;
    signal rstOre : std_logic;
    signal Stop: std_logic;
    Signal Y0: std_logic;
    Signal Y1: std_logic;
    Signal Y2: std_logic;

component ContMod24 is
    Port (
        Clk : in std_logic;
        Rst :in std_logic;
        set :in std_logic;
        setVal : in std_logic_vector(4 downto 0);

```

```

        Cont : out std_logic_vector(4 downto 0);
        Cont23 : inout std_logic

    );
end component;

component ContMod60
    Port (
        Clk : in std_logic;
        Rst : in std_logic;
        set : in std_logic;
        setVal : in std_logic_vector(5 downto 0);
        Cont : inout std_logic_vector(5 downto 0);
        Cont59 : inout std_logic
    );
end component;
begin
Y0 <= Clock and (not(rstMnt) or not(rstOre) or not(stop));

ContSecondi: ContMod60
    Port map(
        Y0 ,
        Rst,
        set,
        SetSec,
        TimerSec,
        rstMnt
    );
Y1 <= (Clock and rstMnt)and (not(rstOre) or not(Stop));
ContMinuti: ContMod60
    Port map(
        Y1 ,
        Rst,
        set,
        SetMinuti,
        TimerMinuti,
        rstOre
    );
Y2 <= (Clock and rstMnt and rstOre) and not(Stop);
ContOre: ContMod24
    Port map(
        Y2 ,
        Rst,
        set,
        SetOre,
        TimerOre,
        Stop
    );
end structural;

```

5.5.5 | Punto 2 - Clock Filter

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clock_filter is
    generic(
        CLKIN_freq : integer := 100000000; --clock board 450MHz

```

```

        CLKOUT_freq : integer := 1           --frequenza desiderata 1Hz
    );
Port (
    clock_in : in STD_LOGIC;
    reset : in STD_LOGIC;
    clock_out : out STD_LOGIC -- attenzione: non è un vero clock ma un impulso che sarà usato
);
end clock_filter;

architecture Behavioral of clock_filter is
signal clockfx : std_logic := '0';
constant count_max_value : integer := CLKIN_freq/(CLKOUT_freq);

begin
clock_out <= clockfx;
count_for_division: process(clock_in)
variable counter : integer range 0 to count_max_value := 0;
begin

    if rising_edge(clock_in) then
        if( reset = '1') then
            counter := 0;
            clockfx <= '0';
        else
            if counter = count_max_value then
                clockfx <= '1';
                counter := 0;
            else
                clockfx <= '0';
                counter := counter + 1;
            end if;
        end if;
    end if;
end process;

end Behavioral;

```

5.5.6 | Punto 2 - Convertitore

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity converter is
    port(
        in_sec,in_min: in std_logic_vector(5 downto 0);
        in_ore : in std_logic_vector(4 downto 0);
        out_conv: out std_logic_vector(23 downto 0)
    );
end converter;

architecture Behavioral of converter is
signal cifra1,cifra2,cifra3,cifra4,cifra5,cifra6: std_logic_vector(3 downto 0);

begin
cifra1 <=
    "0000" when (in_sec="000000" or in_sec="001010" or in_sec="010100" or in_sec="011100"
    "0001" when (in_sec="000001" or in_sec="001011" or in_sec="010101" or in_sec="011111" or in_sec="101100"
    "0010" when (in_sec="000010" or in_sec="001100" or in_sec="010110" or in_sec="100000" or in_sec="101000"
    "0011" when (in_sec="000011" or in_sec="001101" or in_sec="010111" or in_sec="100001" or in_sec="101001"
    "0100" when (in_sec="000100" or in_sec="001110" or in_sec="011000" or in_sec="100010" or in_sec="101010");

```

```

0101" when (in_sec="000101" or in_sec="001111" or in_sec="011001" or in_sec="100011" or in_sec="101100" or in_sec="100100" or in_sec="011010" or in_sec="011011" or in_sec="100101" or in_sec="101110" or in_sec="100110" or in_sec="110000" or in_sec="110011" or in_sec="110100" or in_sec="110111");
0110" when (in_sec="000110" or in_sec="010000" or in_sec="011010" or in_sec="100100" or in_sec="101100" or in_sec="100110" or in_sec="011011" or in_sec="100101" or in_sec="101111" or in_sec="100111" or in_sec="110001" or in_sec="110010" or in_sec="110101" or in_sec="110110");
0111" when (in_sec="000111" or in_sec="010001" or in_sec="011011" or in_sec="100101" or in_sec="101101" or in_sec="100111" or in_sec="011010" or in_sec="100100" or in_sec="101110" or in_sec="100110" or in_sec="110000" or in_sec="110011" or in_sec="110100" or in_sec="110111");
1000" when (in_sec="001000" or in_sec="010010" or in_sec="011100" or in_sec="100110" or in_sec="101100" or in_sec="100111" or in_sec="011010" or in_sec="100100" or in_sec="101110" or in_sec="100110" or in_sec="110000" or in_sec="110011" or in_sec="110100" or in_sec="110111");
1001" when (in_sec="001001" or in_sec="010011" or in_sec="011101" or in_sec="100111" or in_sec="101101" or in_sec="100111" or in_sec="011011" or in_sec="100101" or in_sec="101111" or in_sec="100111" or in_sec="110001" or in_sec="110011" or in_sec="110101" or in_sec="110111");
-----;

cifra2 <=      "0000" when (in_sec="000000" or in_sec="000001" or in_sec="000010" or in_sec="000011" or in_sec="000100" or in_sec="000101" or in_sec="000110" or in_sec="000111" or in_sec="001000" or in_sec="001001" or in_sec="001010" or in_sec="001011" or in_sec="001100" or in_sec="001101" or in_sec="001110" or in_sec="001111" or in_sec="010000" or in_sec="010001" or in_sec="010010" or in_sec="010011" or in_sec="010100" or in_sec="010101" or in_sec="010110" or in_sec="010111" or in_sec="011000" or in_sec="011001" or in_sec="011010" or in_sec="011011" or in_sec="011100" or in_sec="011101" or in_sec="011110" or in_sec="011111");
-----;

cifra3 <=      "0000" when (in_min="000000" or in_min="001010" or in_min="010100" or in_min="011111" or in_min="101100" or in_min="100000" or in_min="100001" or in_min="100010" or in_min="100011" or in_min="101010" or in_min="101011" or in_min="101101" or in_min="101110" or in_min="101111" or in_min="110000" or in_min="110001" or in_min="110010" or in_min="110011" or in_min="110100" or in_min="110101" or in_min="110110" or in_min="110111" or in_min="111000" or in_min="111001" or in_min="111010" or in_min="111011" or in_min="111100" or in_min="111101" or in_min="111110" or in_min="111111");
-----;

cifra4 <=      "0000" when (in_min="000000" or in_min="000001" or in_min="000010" or in_min="000011" or in_min="000100" or in_min="000101" or in_min="000110" or in_min="000111" or in_min="001000" or in_min="001001" or in_min="001010" or in_min="001011" or in_min="001100" or in_min="001101" or in_min="001110" or in_min="001111" or in_min="010000" or in_min="010001" or in_min="010010" or in_min="010011" or in_min="010100" or in_min="010101" or in_min="010110" or in_min="010111" or in_min="011000" or in_min="011001" or in_min="011010" or in_min="011011" or in_min="011100" or in_min="011101" or in_min="011110" or in_min="011111" or in_min="100000" or in_min="100001" or in_min="100010" or in_min="100011" or in_min="100100" or in_min="100101" or in_min="100110" or in_min="100111" or in_min="101000" or in_min="101001" or in_min="101010" or in_min="101011" or in_min="101100" or in_min="101101" or in_min="101110" or in_min="101111" or in_min="110000" or in_min="110001" or in_min="110010" or in_min="110011" or in_min="110100" or in_min="110101" or in_min="110110" or in_min="110111" or in_min="111000" or in_min="111001" or in_min="111010" or in_min="111011" or in_min="111100" or in_min="111101" or in_min="111110" or in_min="111111");
-----;

cifra5 <=      "0000" when (in_ore="00000" or in_ore="01010" or in_ore="10100") else
-----;
0001" when (in_ore="00001" or in_ore="01011" or in_ore="10101") else
-----;
0010" when (in_ore="00010" or in_ore="01100" or in_ore="10110") else
-----;
0011" when (in_ore="00011" or in_ore="01101" or in_ore="10111") else
-----;
0100" when (in_ore="00100" or in_ore="01110") else
-----;
0101" when (in_ore="00101" or in_ore="01111") else
-----;
0110" when (in_ore="00110" or in_ore="10000") else
-----;
0111" when (in_ore="00111" or in_ore="10001") else
-----;
1000" when (in_ore="01000" or in_ore="10010") else
-----;
1001" when (in_ore="01001" or in_ore="10011") else
-----;

cifra6 <=      "0000" when (in_ore="00000" or in_ore="00001" or in_ore="00010" or in_ore="00011" or in_ore="01010" or in_ore="01011" or in_ore="01100" or in_ore="01101" or in_ore="01110" or in_ore="01111") else
-----;

out_conv <= (cifra6 & cifra5 & cifra4 & cifra3 & cifra2 & cifra1);

end Behavioral;
```

5.5.7 | Punto 3 - Blocco Intertempo

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

```

entity intertempi is
  Port ( secondi : in std_logic_vector(5 downto 0);
         minuti : in std_logic_vector(5 downto 0);
         ore : in std_logic_vector(4 downto 0);
         clk : in std_logic;
         reset : in std_logic;
         stop : in std_logic;
         y1 : out std_logic_vector(16 downto 0);
         y2 : out std_logic_vector(16 downto 0);
         y3 : out std_logic_vector(16 downto 0);
         y4 : out std_logic_vector(16 downto 0)
      );
end intertempi;

architecture Behavioral of intertempi is

  signal int_1 : std_logic_vector (16 downto 0);
  signal int_2 : std_logic_vector (16 downto 0);
  signal int_3 : std_logic_vector (16 downto 0);
  signal int_4 : std_logic_vector (16 downto 0);

begin

  intertempo : process(clk, reset, int_1, int_2, int_3, int_4)
  variable cont_intertempi : integer range 0 to 3 := 0;
  begin

    y1 <= int_1;
    y2 <= int_2;
    y3 <= int_3;
    y4 <= int_4;

    if(reset = '1') then
      cont_intertempi := 0;
      int_1 <= "00000000000000000000";
      int_2 <= "00000000000000000000";
      int_3 <= "00000000000000000000";
      int_4 <= "00000000000000000000";

    elsif ( clk' event and clk='1') then
      if(stop='1') then
        if (cont_intertempi = 0) then
          int_1 (5 downto 0) <= secondi;
          int_1 (11 downto 6) <= minuti;
          int_1 (16 downto 12) <= ore;
          cont_intertempi := cont_intertempi +1;
        elsif (cont_intertempi = 1) then
          int_2 (5 downto 0) <= secondi;
          int_2 (11 downto 6) <= minuti;
          int_2 (16 downto 12) <= ore;
          cont_intertempi := cont_intertempi +1;
        elsif (cont_intertempi = 2) then
          int_3 (5 downto 0) <= secondi;
          int_3 (11 downto 6) <= minuti;
          int_3 (16 downto 12) <= ore;
          cont_intertempi := cont_intertempi +1;
        end if;
      end if;
    end if;
  end process;
end Behavioral;

```

```

        elsif (cont_intertempi = 3) then
            int_4 (5 downto 0) <= secondi;
            int_4 (11 downto 6) <= minuti;
            int_4 (16 downto 12) <= ore;
            cont_intertempi := 0;
        end if;
    end if;
end if;

end process;

end Behavioral;

```

5.5.8 | Punto 3 - Blocco Mantieni Uscita

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mantieni_uscita is
    Port ( a0 : in std_logic_vector(16 downto 0);
           a1 : in std_logic_vector(16 downto 0);
           a2 : in std_logic_vector(16 downto 0);
           a3 : in std_logic_vector(16 downto 0);
           a4 : in std_logic_vector(16 downto 0);
           click : in std_logic;
           clk : in std_logic;
           y_sec : out std_logic_vector(5 downto 0);
           y_min : out std_logic_vector(5 downto 0);
           y_ore : out std_logic_vector(4 downto 0)
    );
end mantieni_uscita;

architecture Behavioral of mantieni_uscita is
    signal uscita : std_logic_vector (16 downto 0);
begin
    intertempo : process(clk, uscita, a0, a1, a2, a3, a4)
    variable cont_visualizza : integer range 0 to 5 := 0;

    begin
        y_sec <= uscita (5 downto 0);
        y_min <= uscita (11 downto 6);
        y_ore <= uscita (16 downto 12);

        if(cont_visualizza = 0) then
            uscita <= a0;
        end if;
        if (clk' event and clk='1') then
            if(click = '1') then

                cont_visualizza := cont_visualizza +1;
                if (cont_visualizza = 1) then
                    uscita <= a1;
                elsif (cont_visualizza = 2) then
                    uscita <= a2;
                elsif (cont_visualizza = 3) then
                    uscita <= a3;
                elsif (cont_visualizza = 4) then
                    uscita <= a4;
            end if;
        end if;
    end process;

```

```

        elsif (cont_visualizza = 5) then
            uscita <= a0;
            cont_visualizza := 0;
        end if;
    end if;
end if;
end process;

end Behavioral;

```

5.5.9 | Punto 3 - Sistema Complessivo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Cronometro_on_board is

    port(
        clock: in std_logic;
        reset: in std_logic;
        input_set: in std_logic_vector(10 downto 0);
        pulsante_set : in std_logic;
        pulsante_visualizzazione: in std_logic;
        pulsante_stop_intertempo: in std_logic;
        anodes_out : out STD_LOGIC_VECTOR (7 downto 0);
        cathodes_out : out STD_LOGIC_VECTOR (7 downto 0)
    );
end Cronometro_on_board;

architecture Structural of Cronometro_on_board is
    component cronometro_finale is
        port (
            Clock : in std_logic;
            Rst : in std_logic;
            Set : in std_logic;
            Click_visualizzazione : in std_logic;
            --pulsante per cambiare le modalità di visualizzazione del display
            Stop_intertempo : in std_logic;
            --pulsante per salvare un intertempo
            SetOre : in std_logic_vector(4 downto 0);
            SetMinuti : in std_logic_vector(5 downto 0);
            SetSec : in std_logic_vector(5 downto 0);
            TimerOre : inout std_logic_vector(4 downto 0);
            TimerMinuti : inout std_logic_vector(5 downto 0);
            TimerSec : inout std_logic_vector(5 downto 0)
        );
    end component;

    component ButtonDebouncer is
        generic (
            CLK_period: integer := 10;  -- periodo del clock in nanosec
            btn_noise_time: integer := 100000000 --durata dell'oscillazione in nanosec
        );
        Port ( RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                BTN : in STD_LOGIC;
                CLEARED_BTN : out STD_LOGIC);
    end component;

```

```

component converter is
  port(
    in_sec,in_min: in std_logic_vector(5 downto 0);
    in_ore : in std_logic_vector(4 downto 0);
    out_conv: out std_logic_vector(23 downto 0)
  );
end component;

component display_seven_segments is
Generic(
  CLKIN_freq : integer := 10000000;
  CLKOUT_freq : integer := 500
);
Port ( clock : in STD_LOGIC;
       reset : in STD_LOGIC;
       value32_in : in STD_LOGIC_VECTOR (31 downto 0);
       enable : in STD_LOGIC_VECTOR (7 downto 0);
       dots : in STD_LOGIC_VECTOR (7 downto 0);
       anodes : out STD_LOGIC_VECTOR (7 downto 0);
       cathodes : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal out_temp: std_logic_vector(16 downto 0);
signal out_disp: std_logic_vector(31 downto 0);

signal pulsante_visualizzazione_pulito: std_logic;
signal pulsante_stop_intertempo_pulito: std_logic;
signal pulsante_set_pulito: std_logic;
signal out_converter: std_logic_vector(23 downto 0);

begin
  Bottone_visualizzazione: ButtonDebouncer
  Generic map(
    CLK_period => 10,
    btn_noise_time => 100000000
  )
  port map(
    RST => reset,
    CLK => clock,
    BTN => pulsante_visualizzazione,
    CLEARED_BTN => pulsante_visualizzazione_pulito
  );
  Bottone_stop_intertempo: ButtonDebouncer
  Generic map(
    CLK_period => 10,
    btn_noise_time => 100000000
  )
  port map(
    RST => reset,
    CLK => clock,
    BTN => pulsante_stop_intertempo,
    CLEARED_BTN => pulsante_stop_intertempo_pulito
  );
  Bottone_set: ButtonDebouncer

```

```

Generic map(
    CLK_period => 10,
    btn_noise_time => 100000000
)
port map(
    RST => reset,
    CLK => clock,
    BTN => pulsante_set,
    CLEARED_BTN => pulsante_set_pulito
);

Cronometro: cronometro_finale
port map(
    Clock => clock,
    Rst => reset,
    Set => pulsante_set_pulito,
    Click_visualizzazione => pulsante_visualizzazione_pulito,
    Stop_intertempo => pulsante_stop_intertempo_pulito,
    SetOre => input_set(10 downto 6),
    SetMinuti => input_set(5 downto 0),
    SetSec => "000000",
    TimerOre => out_temp(16 downto 12),
    TimerMinuti => out_temp(11 downto 6),
    TimerSec => out_temp(5 downto 0)
);

Convertitore: converter
port map(
    in_sec => out_temp(5 downto 0),
    in_min => out_temp(11 downto 6),
    in_ore => out_temp(16 downto 12),
    out_conv => out_converter
);

out_disp <= "00000000" & out_converter;

Disp: display_seven_segments
Generic map(
    CLKIN_freq => 100000000,
    CLKOUT_freq => 500
)
port map
    ( clock => clock,
    reset => reset,
    value32_in => out_disp,
    enable => "00111111", -- decide quali cifre abilitare
    dots => "00010100", -- decide quali punti visualizzare
    anodes => anodes_out,
    cathodes => cathodes_out
);

end Structural;

```

5.5.10 | Punto 3 - Mapping Scheda

```

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

```

```

#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform [0 5] [get_ports [clock]];

##Switches
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { input_set[0] }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { input_set[1] }];
#IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { input_set[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { input_set[3] }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { input_set[4] }];
#IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { input_set[5] }];
#IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { input_set[6] }];
#IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { input_set[7] }];
#IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { input_set[8] }];
#IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { input_set[9] }];
#IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { input_set[10] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW[11] }];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { SW[12] }];
#IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { SW[13] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A0
#set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW[15] }];
#IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
#set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { LED[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
#set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { LED[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
#set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { LED[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { LED[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { LED[4] }];
#IO_L7P_T1_D09_14 Sch=led[4]
#set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { LED[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { LED[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { LED[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { LED[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { LED[9] }];
#IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { LED[10] }];

```

```

#IO_L22P_T3_A05_D21_14 Sch=led[10] IOSTANDARD LVCMOS33 [get_ports [ LED[11] ]]; #IO_L15N_T2_D
#set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 [get_ports [ LED[12] ]];
#set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 [get_ports [ LED[13] ]];
#IO_L16P_T2_CSI_B_14 Sch=led[12] IOSTANDARD LVCMOS33 [get_ports [ LED[14] ]];
#set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 [get_ports [ LED[15] ]]; #IO_L21N_T3_D
#IO_L22N_T3_A04_D20_14 Sch=led[13] IOSTANDARD LVCMOS33 [get_ports [ LED[16] ]];
#set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 [get_ports [ LED[17] ]];
#IO_L20N_T3_A07_D23_14 Sch=led[14] IOSTANDARD LVCMOS33 [get_ports [ LED[18] ]];
#set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 [get_ports [ LED[19] ]]; #IO_L21N_T3_D

## RGB LEDs
#set_property -dict { PACKAGE_PIN R12 IOSTANDARD LVCMOS33 [get_ports [ LED16_B ]];
#IO_L5P_T0_D06_14 Sch=led16_b IOSTANDARD LVCMOS33 [get_ports [ LED16_G ]];
#set_property -dict { PACKAGE_PIN M16 IOSTANDARD LVCMOS33 [get_ports [ LED16_R ]];
#IO_L10P_T1_D14_14 Sch=led16_g IOSTANDARD LVCMOS33 [get_ports [ LED17_B ]]; #IO_L15N_T2_D
#set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 [get_ports [ LED17_G ]];
#IO_L11P_T1_SRCC_14 Sch=led16_r IOSTANDARD LVCMOS33 [get_ports [ LED17_R ]];
#set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 [get_ports [ LED17_B ]]; #IO_L15N_T2_D
#set_property -dict { PACKAGE_PIN R11 IOSTANDARD LVCMOS33 [get_ports [ LED17_G ]];
#IO_O_14 Sch=led17_g IOSTANDARD LVCMOS33 [get_ports [ LED17_R ]];
#set_property -dict { PACKAGE_PIN N16 IOSTANDARD LVCMOS33 [get_ports [ LED17_R ]];
#IO_L11N_T1_SRCC_14 Sch=led17_r

##7 segment display
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[0] ]];
#IO_L24N_T3_A00_D16_14 Sch=ca IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[1] ]];
set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[2] ]];
#IO_25_14 Sch=cb IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[3] ]];
set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[4] ]];
#IO_25_15 Sch=cc IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[5] ]];
set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[6] ]];
#IO_L17P_T2_A26_15 Sch=cd IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[7] ]];
set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[8] ]];
#IO_L13P_T2_MRCC_14 Sch=ce IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[9] ]];
set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[10] ]];
#IO_L19P_T3_A10_D26_14 Sch=cf IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[11] ]];
set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[12] ]];
#IO_L4P_T0_D04_14 Sch=cg IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[13] ]];
set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 [get_ports [ cathodes_out[14] ]];
#IO_L19N_T3_A21_VREF_15 Sch=dp IOSTANDARD LVCMOS33 [get_ports [ anodes_out[0] ]];
set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[1] ]];
#IO_L23P_T3_FOE_B_15 Sch=an[0] IOSTANDARD LVCMOS33 [get_ports [ anodes_out[2] ]];
set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[3] ]];
#IO_L23N_T3_FWE_B_15 Sch=an[1] IOSTANDARD LVCMOS33 [get_ports [ anodes_out[4] ]];
set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[5] ]];
#IO_L24P_T3_A01_D17_14 Sch=an[2] IOSTANDARD LVCMOS33 [get_ports [ anodes_out[6] ]];
set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[7] ]];
#IO_L19P_T3_A22_15 Sch=an[3] IOSTANDARD LVCMOS33 [get_ports [ anodes_out[8] ]];
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[9] ]];
#IO_L8N_T1_D12_14 Sch=an[4] IOSTANDARD LVCMOS33 [get_ports [ anodes_out[10] ]];
set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[11] ]];
#IO_L14P_T2_SRCC_14 Sch=an[5] IOSTANDARD LVCMOS33 [get_ports [ anodes_out[12] ]];
set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[13] ]];
#IO_L23P_T3_35 Sch=an[6] IOSTANDARD LVCMOS33 [get_ports [ anodes_out[14] ]];
set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 [get_ports [ anodes_out[15] ]];
#IO_L23N_T3_A02_D18_14 Sch=an[7]

```

```

##Buttons
#set_property -dict { PACKAGE_PIN C12 IOSTANDARD LVCMOS33 } [get_ports [ CPU_RESETN ]]; #IO_L3P_TO_0
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports [ pulsante_stop_intertempo ]];
#set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVCMOS33 } [get_ports [ BTNU ]]; #IO_L4N_TO_D05_1
set_property -dict { PACKAGE_PIN P17 IOSTANDARD LVCMOS33 } [get_ports [ reset ]]; #IO_L12P_T1_MRCC
set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports [ pulsante_set ]]; #IO_L10N_T1
set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33 } [get_ports [ pulsante_visualizzazione ]];

##Pmod Headers
##Pmod Header JA
#set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports [ JA[1] ]]; #IO_L20N_T3_A19
#set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports [ JA[2] ]]; #IO_L21N_T3_DQS
#set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports [ JA[3] ]]; #IO_L21P_T3_DQS
#set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports [ JA[4] ]]; #IO_L18N_T2_A23
#set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 } [get_ports [ JA[7] ]]; #IO_L16N_T2_A27
#set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 } [get_ports [ JA[8] ]]; #IO_L16P_T2_A28
#set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33 } [get_ports [ JA[9] ]]; #IO_L22N_T3_A16
#set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports [ JA[10] ]]; #IO_L22P_T3_A17

##Pmod Header JB
#set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33 } [get_ports [ JB[1] ]]; #IO_L1P_TO_ADOP
#set_property -dict { PACKAGE_PIN F16 IOSTANDARD LVCMOS33 } [get_ports [ JB[2] ]]; #IO_L14N_T2_SRC
#set_property -dict { PACKAGE_PIN G16 IOSTANDARD LVCMOS33 } [get_ports [ JB[3] ]]; #IO_L13N_T2_MRCC
#set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33 } [get_ports [ JB[4] ]]; #IO_L15P_T2_DQS
#set_property -dict { PACKAGE_PIN E16 IOSTANDARD LVCMOS33 } [get_ports [ JB[7] ]]; #IO_L11N_T1_SRC
#set_property -dict { PACKAGE_PIN F13 IOSTANDARD LVCMOS33 } [get_ports [ JB[8] ]]; #IO_L5P_TO_AD9P
#set_property -dict { PACKAGE_PIN G13 IOSTANDARD LVCMOS33 } [get_ports [ JB[9] ]]; #IO_O_15 Sch=jb
#set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports [ JB[10] ]]; #IO_L13P_T2_MRA

##Pmod Header JC
#set_property -dict { PACKAGE_PIN K1 IOSTANDARD LVCMOS33 } [get_ports [ JC[1] ]]; #IO_L23N_T3_35 S
#set_property -dict { PACKAGE_PIN F6 IOSTANDARD LVCMOS33 } [get_ports [ JC[2] ]]; #IO_L19N_T3_VREF
#set_property -dict { PACKAGE_PIN J2 IOSTANDARD LVCMOS33 } [get_ports [ JC[3] ]]; #IO_L22N_T3_35 S
#set_property -dict { PACKAGE_PIN G6 IOSTANDARD LVCMOS33 } [get_ports [ JC[4] ]]; #IO_L19P_T3_35 S
#set_property -dict { PACKAGE_PIN E7 IOSTANDARD LVCMOS33 } [get_ports [ JC[7] ]]; #IO_L6P_TO_35 S
#set_property -dict { PACKAGE_PIN J3 IOSTANDARD LVCMOS33 } [get_ports [ JC[8] ]]; #IO_L22P_T3_35 S
#set_property -dict { PACKAGE_PIN J4 IOSTANDARD LVCMOS33 } [get_ports [ JC[9] ]]; #IO_L21P_T3_DQS
#set_property -dict { PACKAGE_PIN E6 IOSTANDARD LVCMOS33 } [get_ports [ JC[10] ]]; #IO_L5P_TO_AD13

##Pmod Header JD
#set_property -dict { PACKAGE_PIN H4 IOSTANDARD LVCMOS33 } [get_ports [ JD[1] ]]; #IO_L21N_T3_DQS
#set_property -dict { PACKAGE_PIN H1 IOSTANDARD LVCMOS33 } [get_ports [ JD[2] ]]; #IO_L17P_T2_35 S
#set_property -dict { PACKAGE_PIN G1 IOSTANDARD LVCMOS33 } [get_ports [ JD[3] ]]; #IO_L17N_T2_35 S
#set_property -dict { PACKAGE_PIN G3 IOSTANDARD LVCMOS33 } [get_ports [ JD[4] ]]; #IO_L20N_T3_35 S
#set_property -dict { PACKAGE_PIN H2 IOSTANDARD LVCMOS33 } [get_ports [ JD[7] ]]; #IO_L15P_T2_DQS
#set_property -dict { PACKAGE_PIN G4 IOSTANDARD LVCMOS33 } [get_ports [ JD[8] ]]; #IO_L20P_T3_35 S
#set_property -dict { PACKAGE_PIN G2 IOSTANDARD LVCMOS33 } [get_ports [ JD[9] ]]; #IO_L15N_T2_DQS
#set_property -dict { PACKAGE_PIN F3 IOSTANDARD LVCMOS33 } [get_ports [ JD[10] ]]; #IO_L13N_T2_MRCC

##Pmod Header JXADC
#set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports [ XA_N[1] ]]; #IO_L9N_T1_DQ0
#set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33 } [get_ports [ XA_P[1] ]]; #IO_L9P_T1_DQ0
#set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33 } [get_ports [ XA_N[2] ]]; #IO_L8N_T1_ADC
#set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33 } [get_ports [ XA_P[2] ]]; #IO_L8P_T1_ADC
#set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33 } [get_ports [ XA_N[3] ]]; #IO_L7N_T1_ADC
#set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33 } [get_ports [ XA_P[3] ]]; #IO_L7P_T1_ADC
#set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports [ XA_N[4] ]]; #IO_L10N_T1_AN

```

```

##set_property -dict { PACKAGE_PIN B18 } IOSTANDARD LVCMOS33 } [get_ports { XA_P[4] }]; #IO_L10P_T1_A

##VGA Connector
##set_property -dict { PACKAGE_PIN A3 } IOSTANDARD LVCMOS33 } [get_ports { VGA_R[0] }]; #IO_L8N_T1_A
##set_property -dict { PACKAGE_PIN B4 } IOSTANDARD LVCMOS33 } [get_ports { VGA_R[1] }]; #IO_L7N_T1_A
##set_property -dict { PACKAGE_PIN C5 } IOSTANDARD LVCMOS33 } [get_ports { VGA_R[2] }]; #IO_L1N_T0_A
##set_property -dict { PACKAGE_PIN A4 } IOSTANDARD LVCMOS33 } [get_ports { VGA_R[3] }]; #IO_L8P_T1_A
##set_property -dict { PACKAGE_PIN C6 } IOSTANDARD LVCMOS33 } [get_ports { VGA_G[0] }]; #IO_L1P_T0_A
##set_property -dict { PACKAGE_PIN A5 } IOSTANDARD LVCMOS33 } [get_ports { VGA_G[1] }]; #IO_L3N_T0_D
##set_property -dict { PACKAGE_PIN B6 } IOSTANDARD LVCMOS33 } [get_ports { VGA_G[2] }]; #IO_L2N_T0_A
##set_property -dict { PACKAGE_PIN A6 } IOSTANDARD LVCMOS33 } [get_ports { VGA_G[3] }]; #IO_L3P_T0_D
##set_property -dict { PACKAGE_PIN B7 } IOSTANDARD LVCMOS33 } [get_ports { VGA_B[0] }]; #IO_L2P_T0_A
##set_property -dict { PACKAGE_PIN C7 } IOSTANDARD LVCMOS33 } [get_ports { VGA_B[1] }]; #IO_L4N_T0_3
##set_property -dict { PACKAGE_PIN D7 } IOSTANDARD LVCMOS33 } [get_ports { VGA_B[2] }]; #IO_L6N_T0_V
##set_property -dict { PACKAGE_PIN D8 } IOSTANDARD LVCMOS33 } [get_ports { VGA_B[3] }]; #IO_L4P_T0_3
##set_property -dict { PACKAGE_PIN B11 } IOSTANDARD LVCMOS33 } [get_ports { VGA_HS }]; #IO_L4P_T0_15
##set_property -dict { PACKAGE_PIN B12 } IOSTANDARD LVCMOS33 } [get_ports { VGA_VS }]; #IO_L3N_T0_DQS

##Micro SD Connector
##set_property -dict { PACKAGE_PIN E2 } IOSTANDARD LVCMOS33 } [get_ports { SD_RESET }]; #IO_L14P_T2_S
##set_property -dict { PACKAGE_PIN A1 } IOSTANDARD LVCMOS33 } [get_ports { SD_CD }]; #IO_L9N_T1_DQS_A
##set_property -dict { PACKAGE_PIN B1 } IOSTANDARD LVCMOS33 } [get_ports { SD_SCK }]; #IO_L9P_T1_DQS_A
##set_property -dict { PACKAGE_PIN C1 } IOSTANDARD LVCMOS33 } [get_ports { SD_CMD }]; #IO_L16N_T2_35
##set_property -dict { PACKAGE_PIN C2 } IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[0] }]; #IO_L16P_T2_35
##set_property -dict { PACKAGE_PIN E1 } IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[1] }]; #IO_L18N_T2_35
##set_property -dict { PACKAGE_PIN F1 } IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[2] }]; #IO_L18P_T2_35
##set_property -dict { PACKAGE_PIN D2 } IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[3] }]; #IO_L14N_T2_35

##Accelerometer
##set_property -dict { PACKAGE_PIN E15 } IOSTANDARD LVCMOS33 } [get_ports { ACL_MISO }]; #IO_L11P_T1_S
##set_property -dict { PACKAGE_PIN F14 } IOSTANDARD LVCMOS33 } [get_ports { ACL_MOSI }]; #IO_L5N_T0_A
##set_property -dict { PACKAGE_PIN F15 } IOSTANDARD LVCMOS33 } [get_ports { ACL_SCLK }]; #IO_L14P_T2_S
##set_property -dict { PACKAGE_PIN D15 } IOSTANDARD LVCMOS33 } [get_ports { ACL_CSN }]; #IO_L12P_T1_M
##set_property -dict { PACKAGE_PIN B13 } IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[1] }]; #IO_L2P_T0_A
##set_property -dict { PACKAGE_PIN C16 } IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[2] }]; #IO_L20P_T2_S

##Temperature Sensor
##set_property -dict { PACKAGE_PIN C14 } IOSTANDARD LVCMOS33 } [get_ports { TMP_SCL }]; #IO_L1N_T0_ADC
##set_property -dict { PACKAGE_PIN C15 } IOSTANDARD LVCMOS33 } [get_ports { TMP_SDA }]; #IO_L12N_T1_M
##set_property -dict { PACKAGE_PIN D13 } IOSTANDARD LVCMOS33 } [get_ports { TMP_INT }]; #IO_L6N_T0_VRA
##set_property -dict { PACKAGE_PIN B14 } IOSTANDARD LVCMOS33 } [get_ports { TMP_CT }]; #IO_L2N_T0_AD8

##Omnidirectional Microphone
##set_property -dict { PACKAGE_PIN J5 } IOSTANDARD LVCMOS33 } [get_ports { M_CLK }]; #IO_25_35 Sch=m
##set_property -dict { PACKAGE_PIN H5 } IOSTANDARD LVCMOS33 } [get_ports { M_DATA }]; #IO_L24N_T3_35
##set_property -dict { PACKAGE_PIN F5 } IOSTANDARD LVCMOS33 } [get_ports { M_LRSEL }]; #IO_0_35 Sch=r

##PWM Audio Amplifier
##set_property -dict { PACKAGE_PIN A11 } IOSTANDARD LVCMOS33 } [get_ports { AUD_PWM }]; #IO_L4N_T0_15
##set_property -dict { PACKAGE_PIN D12 } IOSTANDARD LVCMOS33 } [get_ports { AUD_SD }]; #IO_L6P_T0_15

##USB-RS232 Interface
##set_property -dict { PACKAGE_PIN C4 } IOSTANDARD LVCMOS33 } [get_ports { UART_RXD_IN }]; #IO_L7P_T1
##set_property -dict { PACKAGE_PIN D4 } IOSTANDARD LVCMOS33 } [get_ports { UART_RXD_OUT }]; #IO_L11N_T1
##set_property -dict { PACKAGE_PIN D3 } IOSTANDARD LVCMOS33 } [get_ports { UART_CTS }]; #IO_L12N_T1_I
##set_property -dict { PACKAGE_PIN E5 } IOSTANDARD LVCMOS33 } [get_ports { UART_RTS }]; #IO_L5N_T0_A

##USB HID (PS/2)

```

```

##set_property -dict { PACKAGE_PIN F4 } IOSTANDARD LVCMOS33 } [get_ports { PS2_CLK }]; #IO_L13P_T2_M
##set_property -dict { PACKAGE_PIN B2 } IOSTANDARD LVCMOS33 } [get_ports { PS2_DATA }]; #IO_L10N_T1_M

##SMSC Ethernet PHY
##set_property -dict { PACKAGE_PIN C9 } IOSTANDARD LVCMOS33 } [get_ports { ETH_MDC }]; #IO_L11P_T1_S
##set_property -dict { PACKAGE_PIN A9 } IOSTANDARD LVCMOS33 } [get_ports { ETH_MDIO }]; #IO_L14N_T2_S
##set_property -dict { PACKAGE_PIN B3 } IOSTANDARD LVCMOS33 } [get_ports { ETH_RSTN }]; #IO_L10P_T1_A
##set_property -dict { PACKAGE_PIN D9 } IOSTANDARD LVCMOS33 } [get_ports { ETH_CRSDEV }]; #IO_L6N_T0_A
##set_property -dict { PACKAGE_PIN C10 } IOSTANDARD LVCMOS33 } [get_ports { ETH_RXERR }]; #IO_L13N_T2_A
##set_property -dict { PACKAGE_PIN C11 } IOSTANDARD LVCMOS33 } [get_ports { ETH_RXD[0] }]; #IO_L13P_T1_A
##set_property -dict { PACKAGE_PIN D10 } IOSTANDARD LVCMOS33 } [get_ports { ETH_RXD[1] }]; #IO_L19N_T1_A
##set_property -dict { PACKAGE_PIN B9 } IOSTANDARD LVCMOS33 } [get_ports { ETH_TXEN }]; #IO_L11N_T1_A
##set_property -dict { PACKAGE_PIN A10 } IOSTANDARD LVCMOS33 } [get_ports { ETH_TXD[0] }]; #IO_L14P_T1_A
##set_property -dict { PACKAGE_PIN A8 } IOSTANDARD LVCMOS33 } [get_ports { ETH_TXD[1] }]; #IO_L12N_T1_A
##set_property -dict { PACKAGE_PIN D5 } IOSTANDARD LVCMOS33 } [get_ports { ETH_REFCLK }]; #IO_L11P_T1_A
##set_property -dict { PACKAGE_PIN B8 } IOSTANDARD LVCMOS33 } [get_ports { ETH_INTN }]; #IO_L12P_T1_A

##Quad SPI Flash
##set_property -dict { PACKAGE_PIN K17 } IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[0] }]; #IO_L1P_T0_F
##set_property -dict { PACKAGE_PIN K18 } IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[1] }]; #IO_L1N_T0_F
##set_property -dict { PACKAGE_PIN L14 } IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[2] }]; #IO_L2P_T0_F
##set_property -dict { PACKAGE_PIN M14 } IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[3] }]; #IO_L2N_T0_F
##set_property -dict { PACKAGE_PIN L13 } IOSTANDARD LVCMOS33 } [get_ports { QSPI_CSN }]; #IO_L6P_T0_F

```

6 | Sistema di Testing

6.1 | Traccia

Punto 1 Progettare, implementare in VHDL e verificare mediante simulazione un sistema in grado di testare in maniera automatica una macchina combinatoria M avente 4 ingressi e 3 uscite binarie sottoponendole N ingressi diversi (si considerino una macchina M e un numero di input N a scelta dello studente).

Gli N valori di input per il test devono essere letti da una ROM, in cui essi sono precaricati, in corrispondenza di un segnale read.

Le N uscite fornite della macchina in corrispondenza di ciascuno degli input devono essere memorizzate in una memoria interna, che deve poter essere svuotata in qualsiasi momento in presenza di un segnale di reset.

Punto 2 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 | Descrizione Soluzione

Per questo esercizio abbiamo pensato ad una macchina combinatoria che ha le seguenti specifiche: 4 ingressi x_0, x_1, x_2 e x_3 e 3 uscite y_1, y_2 e y_3 , viene realizzata una funzione casuale.

Una volta definito il comportamento della macchina combinatoria, siamo passati a definire lo schema che descrive la nostra soluzione, visibile nella figura 6.2, caratterizzata dall'utilizzo di memorie che abbiamo chiamato RAM e ROM.

Come intuibile, le memorie RAM si intende che è possibile leggere e scrivere, mentre con le memorie Rom è permessa la sola lettura, inoltre è stato associato un contatore alle memorie per scandire l'indirizzo in maniera lineare.

La memoria ROM è stata costruita utilizzando lo stesso schema visibile nella figura 6.1, mentre la RAM in maniera comportamentale.

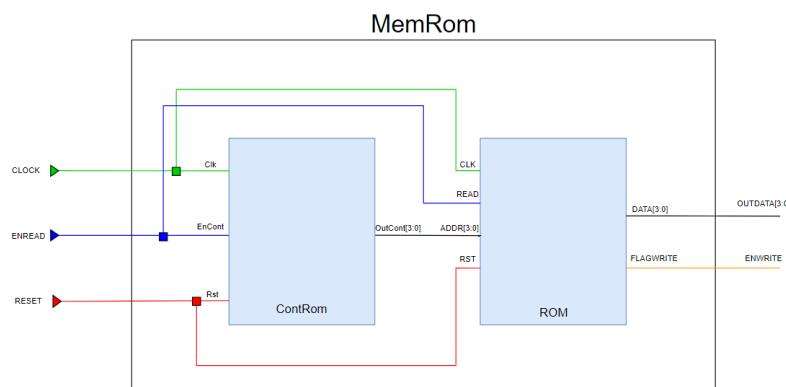


Figura 6.1: Struttura della Memoria ROM.

Il sistema nel complesso controlla la ROM tramite tre segnali di ingresso e produce in uscita un segnale di 4 bit, che viene passato in ingresso alla macchina combinatoria, che sulla base del dato ricevuto in ingresso produce tre bit di uscita che saranno memorizzate in una memoria interna, che può essere svuotata con un segnale di reset.

Come visibile dallo schematico in figura 6.2 è stato introdotto nella memoria Rom un segnale che è sostanzialmente un flag, sarà alto fin quando in uscita diamo lo stesso valore, in questo modo riusciamo a sincronizzare perfettamente la fase di lettura e scrittura della macchina di testing, però da notare che ciò è possibile solo perché stiamo utilizzando una macchina combinatoria abbastanza veloce.



Figura 6.2: Struttura macchina di testing al punto 1.

6.2.1 | Punto 2

Il sistema è costituito dalle entità che vediamo nella figura 6.3, quindi un debouncer, una memoria ROM, una macchina combinatoria ed una memoria interna.

Il debouncer è stato introdotto per permettere all'utente di eseguire uno alla volta i test salvati nella memoria ROM.

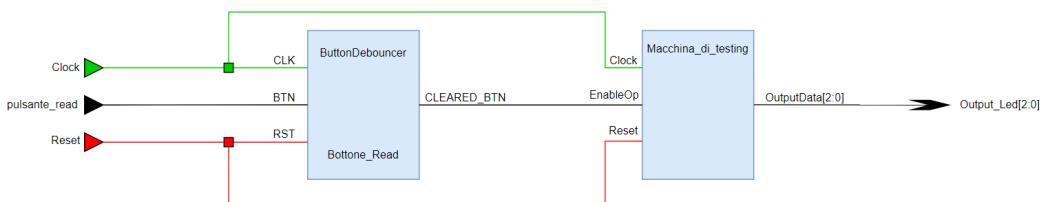


Figura 6.3: Struttura macchina di testing al punto 2.

6.3 | Testbench

Abbiamo eseguito una serie di operazioni per poi dare un segnale di reset e farle eseguire di nuovo.

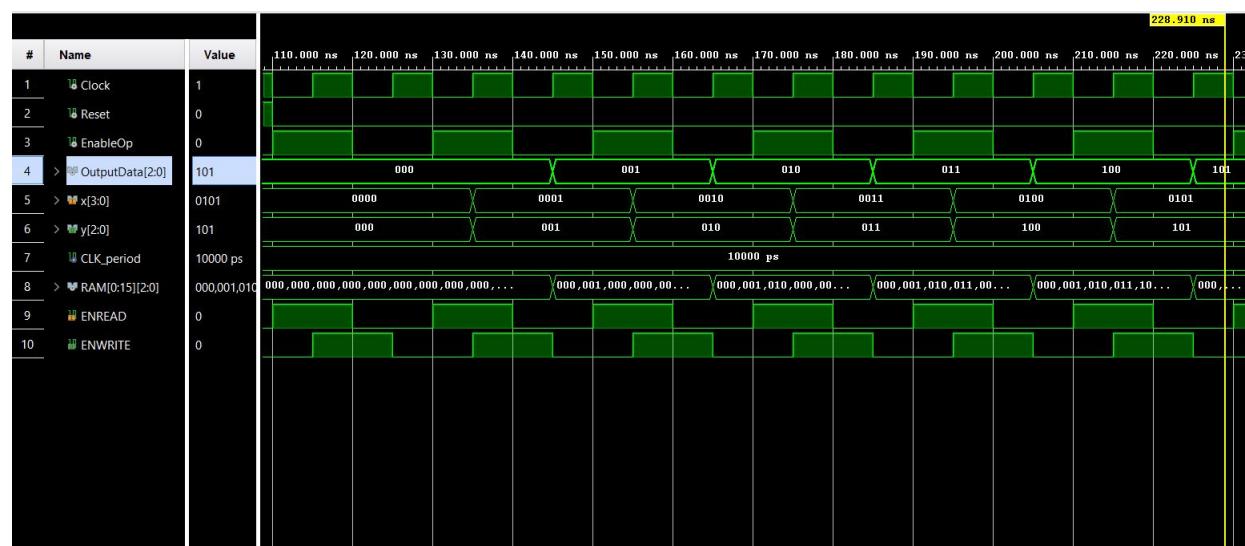


Figura 6.4: Testbench macchina di testing.

6.4 | Sintesi

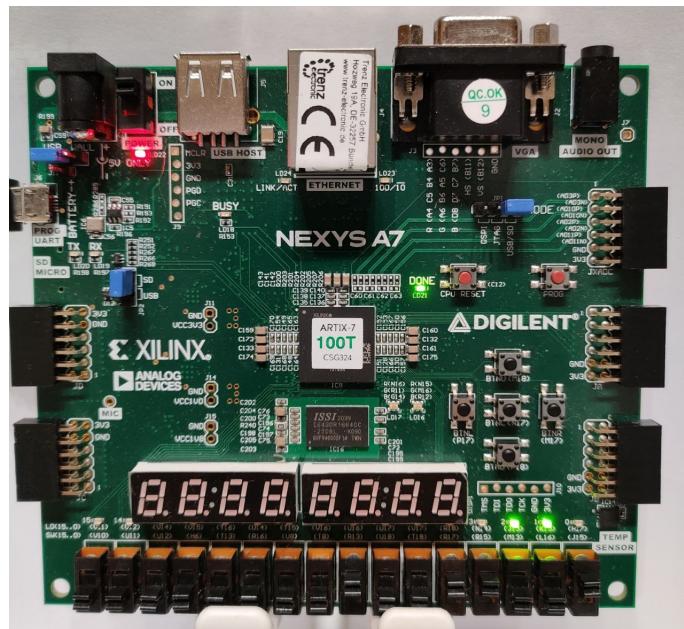


Figura 6.5: Sintesi della macchina di testing.

6.5 | Codice

6.5.1 | Punto 1 - Macchina Combinatoria

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity MacchinaComb4_3 is
  Port (
    x: in std_logic_vector(3 downto 0);
    y: out std_logic_vector(2 downto 0)
  );
end MacchinaComb4_3;

architecture dataflow of MacchinaComb4_3 is
begin
  y <= "000"      when x="0000"
  else "001"      when x="0001"
  else "010"      when x="0010"
  else "011"      when x="0011"
  else "100"      when x="0100"
  else "101"      when x="0101"
  else "110"      when x="0110"
  else "111"      when x="0111"
  else "000"      when x="1000"
  else "001"      when x="1001"
  else "010"      when x="1010"
  else "011"      when x="1011"
  else "100"      when x="1100"
  else "101"      when x="1101"
  else "110"      when x="1110"
  else "111"      when x="1111";
end dataflow;

```

6.5.2 | Punto 1 - Memoria Rom

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MemoriaRom is
generic(N:integer:=3);
Port (
    CLOCK:in std_logic;
    RESET:in std_logic;
    ENREAD:in std_logic;
    ENWRITE :out std_logic;
    OUTDATA : out std_logic_vector(N downto 0)

);

end MemoriaRom;

architecture structural of MemoriaRom is
component ContRom is
    generic(N:integer:=3);
    port(
        Clk, Rst: in std_logic:='0';
        EnCont : in std_logic:='0';
        OutCont : out std_logic_vector (N downto 0):= (others => '0')
    );
end component;

component ROM is
    generic(
        AddrLen: natural :=3;
        DataLen: natural :=3;
        NumberCells: natural :=15
    );
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ADDR : in std_logic_vector(AddrLen downto 0);
        FLAGWRITE : out std_logic;
        READ : in std_logic;
        DATA : out std_logic_vector(DataLen downto 0)
    );
end component;

signal address : std_logic_vector (3 downto 0):= (others => '0');

begin
    ContatoreRom: ContRom
    port map(
        Clk =>CLOCK,
        Rst=>RESET,
        EnCont =>ENREAD,
        OutCont =>address
    );
    MemRom: ROM
    port map(

```

```

    CLK =>CLOCK,
    RST =>RESET,
    ADDR =>address,
    FLAGWRITE =>ENWRITE,
    READ =>ENREAD,
    DATA =>OUTDATA
);

end structural;

```

6.5.3 | Punto 1 - Memoria Ram

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

entity MemoriaRam is
generic(
    AddrLen: natural :=3;
    DataLen: natural :=2;
    NumberCells: natural :=15
);
port
(
    CLK: in std_logic;
    WRITE: in std_logic;
    RST: in std_logic;
    DATAIN: in std_logic_vector (DataLen DOWNTO 0);
    DATAOUT: out std_logic_vector (DataLen DOWNTO 0)

);
END MemoriaRam;

ARCHITECTURE Behavioral OF MemoriaRam IS

    type ram_type is array (0 to NumberCells) of std_logic_vector(DataLen downto 0);

    signal RAM : ram_type := (
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000",
"000"
);
    signal address : std_logic_vector (AddrLen DOWNTO 0) := "0000";
begin
WRITE_MEM: Process (CLK)

```

```

begin
  if (CLK'event AND CLK = '1') then
    if (WRITE = '1') then
      RAM(conv_integer(address)) <= DATAIN;
      DATAOUT<=DATAIN;
      address <= std_logic_vector(unsigned(address) + 1);
    else

      end if;
    if (RST ='1') then
      RAM<= (others => (others => '0'));
      address<=(others => '0');
      DATAOUT<="000";
    end if;

    end if;
  end process;
end Behavioral;

```

6.5.4 | Punto 1 - Contatore Modulo N

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ContModN is
  generic(
    N:natural:=2

  );
  Port ( clock : in STD_LOGIC;
         reset : in STD_LOGIC;
         enable : in STD_LOGIC; --questo è l'enable del clock, insieme danno l'impulso di
         counter : out STD_LOGIC_VECTOR (N downto 0));
end ContModN;

architecture Behavioral of ContModN is

signal tmp : std_logic_vector (N downto 0) := (others => '0');

begin
  counter <= tmp;
  counter_process: process(clock)
  begin
    if (clock'event AND clock = '1') then
      if (reset = '1') then
        tmp <= (others => '0');
      else
        if (enable='1') then
          tmp <= std_logic_vector(unsigned(tmp) + 1);
        end if;
      end if;
    end if;
  end process;
end;

```

```
end process;
end Behavioral;
```

6.5.5 | Punto 2 - Button Debouncer

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

use IEEE.NUMERIC_STD.ALL;

entity ButtonDebouncer is
    generic (
        CLK_period: integer := 10; -- periodo del clock in nanosec
        btn_noise_time: integer := 100000000 --durata dell'oscillazione in nanosec
    );
    Port ( RST : in STD_LOGIC;
            CLK : in STD_LOGIC;
            BTN : in STD_LOGIC;
            CLEARED_BTN : out STD_LOGIC);
end ButtonDebouncer;

architecture Behavioral of ButtonDebouncer is

-- questo componente prende in input il segnale proveniente dal bottone e genera un
-- segnale "ripulito" che presenta un impulso della durata di un colpo di clock per
-- segnalare l'avvenuta pressione del bottone.
-- appena viene rilevato un segnale alto, la macchina si porta in uno stato RELEASED_VRFY
-- in cui attende che trascorra il tempo di "rimbalzo": se il segnale ncora alto vuole dire che
-- il bottone era stato davvero premuto e quindi si passa nello stato RELEASED_CNFMD
-- allo stesso modo, se il segnale si abbassa, si va in uno stato di RELEASED_VRFY dove
-- si attende che trascorra il tempo di "rimbalzo" per vedere se davvero il bottone tato
-- rilasciato, nel qual caso si va in RELEASED
-- in questo modo anche tenendo premuto il bottone viene generato un singolo impulso

type stato is (RELEASED, PRESSED_VRFY, PRESSED_CNFMD, RELEASED_VRFY);
signal BTN_state : stato := RELEASED;

constant max_count : integer := btn_noise_time/CLK_period; -- dipende dall'oscillazione del segnale
-- deve essere tale da superare tutto il tempo dell'oscillazione

begin

deb: process (CLK)
variable count: integer := 0;

begin
    if rising_edge(CLK) then

        if( RST = '1') then
            BTN_state <= RELEASED;
            CLEARED_BTN <= '0';
        end if;
        count := count + 1;
        if count > max_count then
            if BTN = '1' then
                BTN_state <= PRESSED_VRFY;
            else
                BTN_state <= RELEASED_VRFY;
            end if;
        end if;
    end if;
end process;
end Behavioral;
```

```

else
    case BTN_state is
        when RELEASED =>
            CLEARED_BTN<= '0';
            if( BTN = '1') then
                BTN_state <= PRESSED_VRFY;
            else
                BTN_state <= RELEASED;
            end if;

when PRESSED_VRFY => --appena vedo il bottone premuto inizio a contare
    if(count = max_count -1) then --quando arrivo al max count controllo se ncora alto
        if( BTN = '1') then
            count :=0;
            CLEARED_BTN <= '1';
            BTN_state <= PRESSED_CNFRMD; --se ncora alto vado a PRESSED_CNFRMD e also
        else
            BTN_state <= RELEASED;
        end if;
    else
        count:= count+1;
    end if;

when PRESSED_CNFRMD =>
    CLEARED_BTN <= '0';
    --
    if(count = 100*max_count -1) then
        count :=0;
        BTN_state <= NOT_PRESSED;
    else
        count:= count+1;
        BTN_state <= PRESSED_CNFRMD;
    end if;
    if( BTN = '0') then
        BTN_state <= RELEASED_VRFY;
    else
        BTN_state <= PRESSED_CNFRMD;
    end if;

when RELEASED_VRFY =>
    if(count = max_count -1) then --quando arrivo al max count controllo se ncora alto
        if( BTN = '0') then
            count :=0;
            BTN_state <= RELEASED;
        else
            BTN_state <= PRESSED_VRFY;
        end if;
    else
        count:= count+1;
    end if;

when others =>
    BTN_state <= RELEASED;
end case;
end if;
end if;
end process;

```

```
end Behavioral;
```

6.5.6 | Punto 2 - Sistema Complessivo

```
-- Company:
-- Engineer:
--
-- Create Date: 27.12.2022 10:43:29
-- Design Name:
-- Module Name: MacchinaTesting_Board - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MacchinaTesting_Board is
    generic(N:natural:=2);
    Port (
        Clock: in std_logic;
        Reset: in std_logic;
        pulsante_Read: in std_logic;
        Output_Led: out std_logic_vector(N downto 0)
    );
end MacchinaTesting_Board;

architecture Behavioral of MacchinaTesting_Board is

component MacchinaTesting is
    generic(N:natural:=2);
    Port (
        Clock: in std_logic;
        Reset: in std_logic;
        EnableOp: in std_logic;
        OutputData: out std_logic_vector(N downto 0)
    );

```

```
end component;

component ButtonDebouncer is
    generic (
        CLK_period: integer := 10; -- periodo del clock in nanosec
        btn_noise_time: integer := 100000000 --durata dell'oscillazione in nanosec
    );
    Port ( RST : in STD_LOGIC;
            CLK : in STD_LOGIC;
            BTN : in STD_LOGIC;
            CLEARED_BTN : out STD_LOGIC);
end component;

signal read_pulito : std_logic;

begin

Bottone_Read: ButtonDebouncer
    Generic map(
                    CLK_period => 10,
                    btn_noise_time => 100000000
                )
    port map(
        RST => Reset,
        CLK => Clock,
        BTN => pulsante_Read,
        CLEARED_BTN => read_pulito
    );

Macchina_di_Testing: MacchinaTesting
    Port map(
        Clock => Clock,
        Reset => Reset,
        EnableOp => read_pulito,
        OutputData => Output_Led
    );

end Behavioral;
```

7 | Comunicazione con Handshaking

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 dove $i = 0,..,N-1$.

Il nodo A trasmette a B ciascuna stringa X(i) utilizzando un protocollo di handshaking, mentre 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.

7.1 | Descrizione Soluzione

Per descrivere la comunicazione tra il nodo A e il nodo B abbiamo definito un handshake basato su 2 segnali:

- **Request_A**, questo segnale viene generato dal nodo A e viene utilizzato da quest'ultimo per richiedere al nodo B un'elaborazione, ma anche per indicare la fine della trasmissione da parte di A.
- **OK_B**, questo segnale è utilizzato da B come feedback alle richieste di A, se il segnale è basso allora B è libero e può eseguire un'elaborazione, viceversa B è ancora impegnato ed A deve aspettare.

Ad ogni impulso del segnale Start sul nodo A avverrà una comunicazione e il risultato verrà salvato nel nodo B, inoltre entrambi i nodi hanno un segnale di STOP per indicare che hanno letto e/o scritto tutta la memoria a loro disposizione e quindi non saranno più disponibili alla comunicazione.

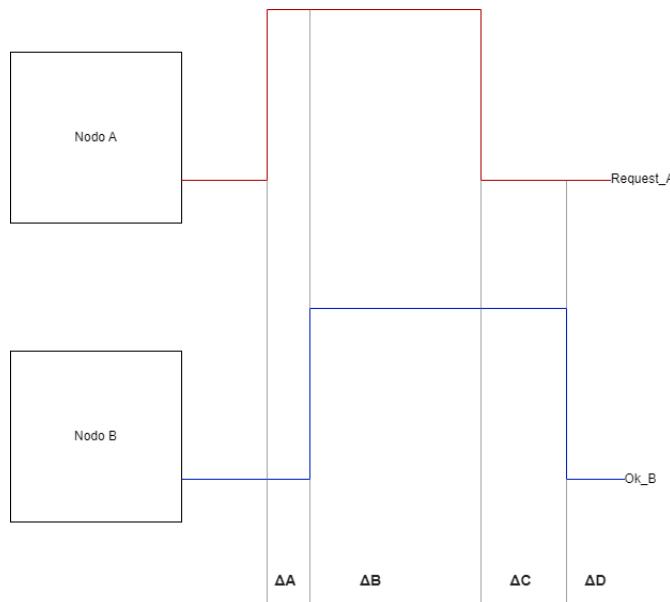


Figura 7.1: Forma d'onda del handshake.

Nella figura 7.1 è rappresentata la forma d'onda del handshake e sono individuati 4 intervalli di tempo, in particolare:

- Durante ΔA il nodo A richiede un elaborazione al nodo B che risulta essere inattivo.
- Durante ΔB il nodo A riceve conferma da B e inizia la trasmissione dei dati.
- Nel intervallo ΔC il nodo A ha terminato la trasmissione, mentre il nodo B sta effettuando l'elaborazione.
- Nel intervallo ΔD il nodo B ha terminato l'elaborazione ed è pronto a una nuova richiesta.

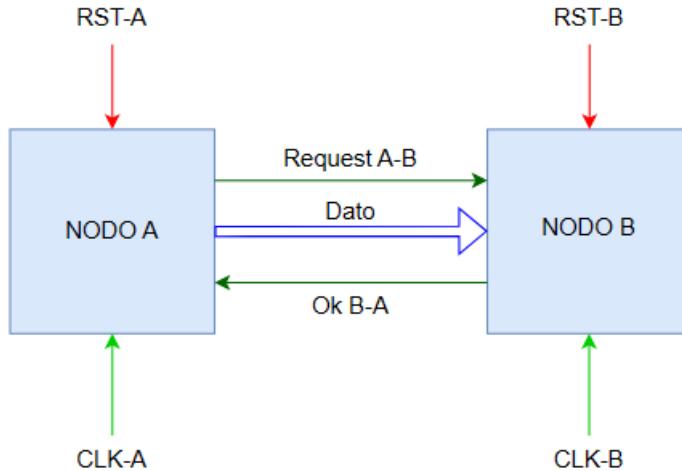


Figura 7.2: Schema generale del sistema complessivo.

7.1.1 | Struttura Nodo A

Abbiamo costruito il nodo A dividendo in parte operativa e parte di controllo.

Parte di Controllo L'unità di controllo è stata realizzata partendo da un automa, visibile nella figura 7.3, quindi è stato realizzato il protocollo di handshake dal punto di vista di A.

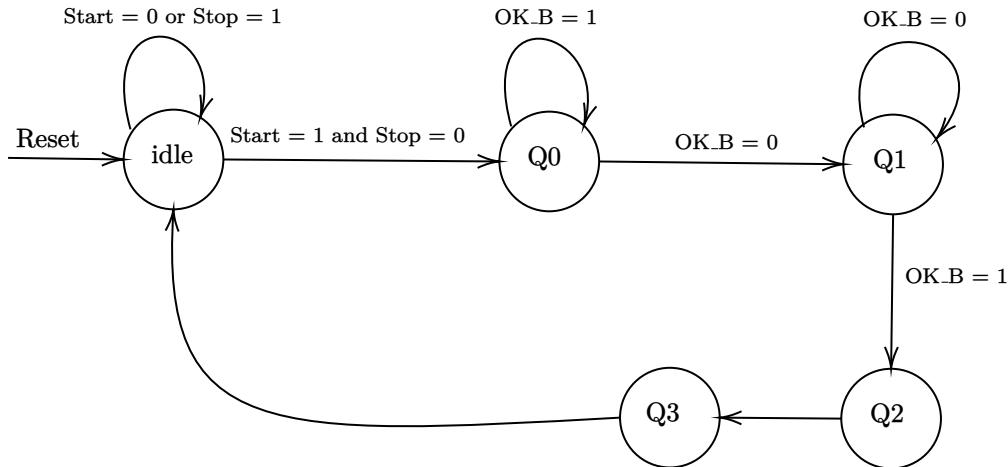


Figura 7.3: Automa del nodo A.

Ogni stato esegue un compito ben preciso, quindi gestisce alcuni dei segnali forniti dall'interfaccia del UC, la sua struttura è visibile nella figura 7.4, in particolare:

- **Idle** - è uno stato d'utilità che serve per mantenere il nodo in uno stato di inattività da cui esce con il segnale esterno di Start, oppure in cui permane se c'è quello interno di Stop.
- **Q0** - è lo stato che si occupa, quando viene dato il segnale di start, di controllare se il nodo B è disponibile all'elaborazione, in questo caso non vengono prodotti segnali d'uscita.
- **Q1** - è lo stato che aspetta da B l'avvenuta ricezione del segnale di request, in questo vengono prodotti due segnali:
 - **On Read** per abilitare la lettura dalla memoria interna e porre un dato nel registro per l'invio.
 - **Request_A** viene posto alto per realizzare il primo fronte di salita del segnale.

- **Q2** - questo stato si occupa di effettuare la trasmissione dei dati, quindi avremo:
 - **On Read** viene abbassato poiché è già avvenuta la scrittura nel registro.
 - **On Tx** viene posto alto per effettuare la trasmissione.
- **Q3** - questo è lo stato finale che termina la comunicazione da parte di A e setta la memoria per la prossima trasmissione, vengono impostati quindi i segnali:
 - **On Tx** a zero per terminare la comunicazione.
 - **On ADDR** per incrementare l'indirizzo della memoria.
 - **Request_A** viene abbassato per realizzare il fronte di discesa del segnale.



Figura 7.4: Unità di Controllo del nodo A.

Parte Operativa La struttura è visibile nella figura 7.5, in particolare: è presente una memoria con associato un contatore in modo da poterla scorrere linearmente e fornire il segnale di STOP per indicare che sono stati letti tutti i dati a disposizione, inoltre per l'invio dei dati è stato predisposto un registro chiamato RX con un segnale di enable che a tutti gli effetti permette la lettura e quindi la trasmissione.

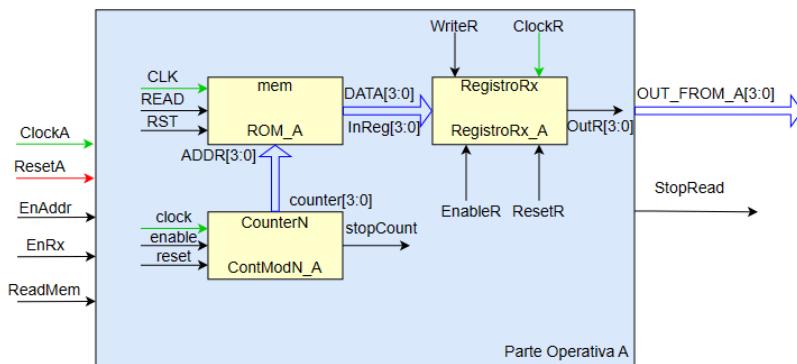


Figura 7.5: Unità operativa del nodo A.

7.1.2 | Struttura Nodo B

Anche l'unità B è stata costruita dividendo in parte operativa e parte di controllo.

Parte di Controllo È stata realizzata in modo analogo all'unità di controllo di A, quindi partendo da un automa e definendo la relativa interfaccia.

Quindi proseguiamo col analizzare l'automa, visibile nella figura 7.7, e le operazioni in ogni stato:

- **Idle** - Il nodo B deve essere sempre pronto all'esecuzione di una richiesta di A, quindi ha bisogno di questo stato che ad ogni colpo di clock controlla se ha ricevuto una richiesta, in questo stato non vengono generati segnali di controllo.
- **Q0** - Lo stato che si occupa di rispondere alle richieste di A e si pone in ricezione, quindi avremo:
 - **OkB** viene messo alto per rispondere ad A e quindi realizzare il primo fronte di salita del handshake.



Figura 7.6: Unità di controllo del nodo B.

- **On RX** con valore alto per porre il registro in ricezione.
- **Q1** - Quando A comunica la fine della trasmissione, si arriva in questo stato che si occupa di leggere i dati e preparare gli addendi per il sommatore:
 - **On Read** posto alto per leggere dalla memoria e dal registro RX.
 - **On RX** posto basso per disattivare la ricezione.
- **Q2** - Questo stato effettua la somma e disattiva la lettura in memoria quindi avremo:
 - **On Read** viene posto basso.
 - **On Add** viene posto alto.
 - **On Write** viene attivato per eseguire una scrittura in memoria del risultato della somma.
- **Q3** - l'ultimo stato con il compito di preparare la memoria per le prossime elaborazioni e di gestire il segnale di Stop, in particolare:
 - **On Add** viene posto basso.
 - **On Write** viene posto basso per terminare la scrittura.
 - **Ed Addr** viene utilizzato per scorrere la memoria di una posizione.
 - **OkA** viene posto basso per segnale la fine dell'elaborazione, ma nel caso in cui il segnale di stop è alto anche questo segnale rimarrà alto in modo da indicare l'impossibilità ad eseguire altre operazioni.

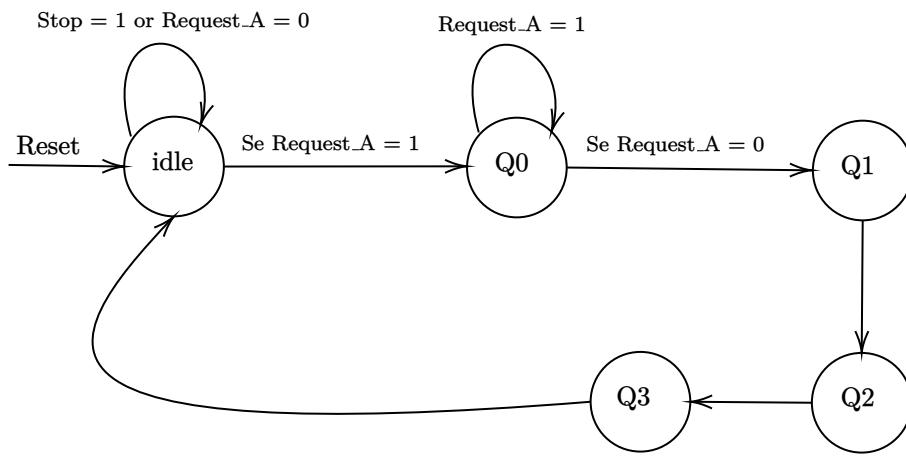


Figura 7.7: Automa del nodo B.

Parte Operativa La parte operativa di B si basa su un registro e due memorie, una chiamata ROM poiché a sola lettura ed una chiamata RAM poiché permette sia lettura che scrittura, inoltre l'indirizzo di entrambe le memorie è gestito da un solo contatore e anche in questo caso produce un segnale di Stop, infine, il registro RX serve per la ricezione dei dati ed è del tutto analogo a quello del nodo A.

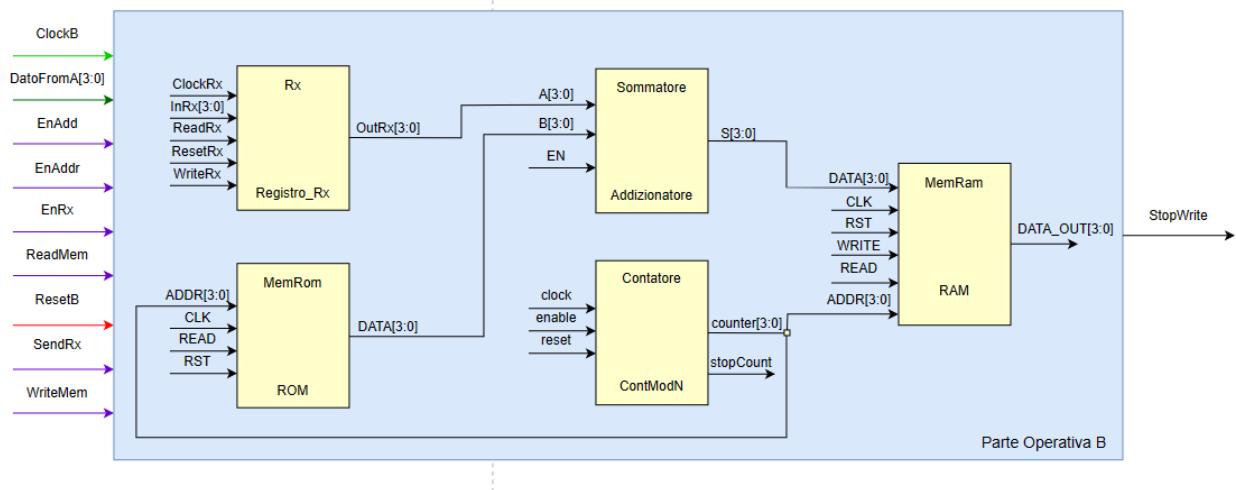


Figura 7.8: Unità operativa del nodo B.

7.2 | Testbench

Il testbench del handshake pone due clock diversi per il nodo A e il nodo B in modo da verificare al meglio l'asincronismo tra i due.

Viene posto alto il segnale di Start, quindi il nodo A invierà tutti i dati possibili nel tempo di simulazione, inoltre è stato introdotto un segnale di reset sempre su A per controllare come si comportava la macchina in questo caso, come visibile dalla figura 7.9 è avvenuto il comportamento atteso.

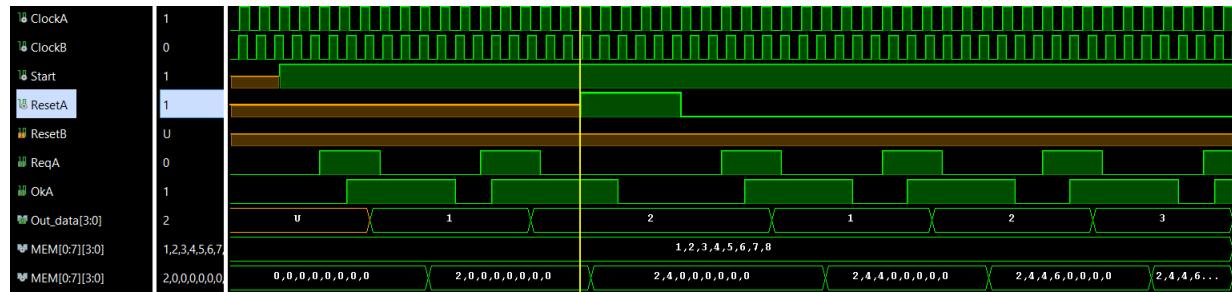


Figura 7.9: Testbench del handshake.

7.3 | Codice

7.3.1 | Nodo A - Sistema Complessivo

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

entity NodoA is
    generic(
        Datalen: natural := 3
    );
    port(
        ClockA: in std_logic;
        ResetA: in std_logic;
        StartA: in std_logic;

        --handshake
        Req_to_B: out std_logic;
        OK_by_B:  in std_logic;
```

```

        Out_data: out std_logic_vector(Datalen downto 0)
    );
end NodoA;

architecture structural of NodoA is

component NodoA_po is Port (
    Clock: in std_logic;
    Reset: in std_logic;

    EnTx:in std_logic;
    SetMEM :in std_logic;
    ReadMEM:in std_logic;
    StopMEM:out std_logic:= '0';
    OUT_FROM_A : out std_logic_vector(3 downto 0)
);
end component;

component NodoA_pc is port(
    StartA: in std_logic;
    ClockA: in std_logic;
    ResetA: in std_logic;

    ReqA:   out std_logic;
    OkB:    in  std_logic;

    rx:     out std_logic;
    read:   out std_logic;
    stop:   in  std_logic;
    set:    out std_logic
);
end component;

signal EnTx:      std_logic;
signal SetMEM : std_logic;
signal ReadMEM: std_logic;
signal StopMEM: std_logic;

begin

Parte_Operativa: NodoA_po port map(
    Clock      => ClockA,
    Reset      => ResetA,

    EnTx       => EnTx,
    SetMEM     => SetMEM,
    ReadMEM   => ReadMEM,
    StopMEM   => StopMEM,
    OUT_FROM_A => Out_data
);

Parte_Controllo: NodoA_pc port map(
    StartA    => StartA,
    ClockA   => ClockA,
    ResetA   => ResetA,

    ReqA      => Req_to_B,
    OkB       => OK_by_B,

```

```

        rx      => EnTX,
        read    => ReadMEM,
        stop    => StopMEM,
        set     => SetMEM
    );

```

```
end structural;
```

7.3.2 | Nodo A - Parte Operativa

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NodoA_po is
    generic (
        DataLen: natural := 3
    );
    Port (
        Clock: in std_logic;
        Reset: in std_logic;

        EnTx:in std_logic;
        SetMEM :in std_logic;
        ReadMEM:in std_logic;
        StopMEM:out std_logic:= '0';
        OUT_FROM_A : out std_logic_vector(3 downto 0) := "0000"
    );
end NodoA_po;

architecture Behavioral of NodoA_po is

component L_MEM_R is
    generic(
        DataLen: natural :=3
    );
    Port(
        ClkIn : in STD_LOGIC;
        RstIn : in STD_LOGIC;
        EnRead : in STD_LOGIC;
        EndRead :out STD_LOGIC;
        cnt: in STD_LOGIC;
        OutData : out std_logic_vector(DataLen downto 0)
    );
end component;

component Registro is
    generic(N: natural:=3);
    port(
        ClockRx, ResetRx: in std_logic;
        InRx: in std_logic_vector(N downto 0);
        WriteRx: in std_logic;
        ReadRx: in std_logic;
        OutRx: out std_logic_vector(N downto 0)
    );
end component;

signal OutMEM: std_logic_vector(DataLen downto 0);

```

```

begin

MEM: L_MEM_R port map(
    ClkIn    => Clock,
    RstIn    => reset,
    EnRead   => ReadMEM,
    EndRead  => StopMEM,
    cnt       => SetMEM,
    OutData  => OutMEM
);

RX: Registro port map(
    ClockRx      => Clock,
    ResetRx      => Reset,
    InRx         => OutMEM,
    ReadRX       => EnTx,
    WriteRx      => ReadMEM,
    OutRx        => OUT_FROM_A
);

```

```
end Behavioral;
```

7.3.3 | Nodo A - Parte di Controllo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NodoA_pc is
    port(
        StartA: in std_logic;
        ClockA: in std_logic;
        ResetA: in std_logic;

        ReqA:    out std_logic;
        OkB:     in  std_logic;

        rx:      out std_logic;
        read:    out std_logic;
        stop:    in  std_logic;
        set:     out std_logic
    );
end NodoA_pc;

architecture Behavioral of NodoA_pc is

type stato is (idle, q0, q1, q2, q3);
signal stato_corrente : stato := idle;
signal stato_next: stato;

begin

--Evoluzione stati
process(ClockA, ResetA, stato_corrente)
begin

    if ResetA = '1' then
        stato_corrente <= idle;
    elsif (ClockA'event and ClockA = '1') then

```

```

        stato_corrente <= stato_next;
    end if;

    case stato_corrente is
    when idle =>
        if StartA = '1' and stop = '0' then
            stato_next <= q0;
        elsif StartA = '0' or stop = '1' then
            stato_next <= idle;
        end if;
    when q0 =>
        if OkB = '1' then
            stato_next <= q0;
        elsif OkB = '0' then
            stato_next <= q1;
        end if;
    when q1 =>
        if OkB = '0' then
            stato_next <= q1;
        elsif OkB = '1' then
            stato_next <= q2;
        end if;
    when q2 =>
        stato_next <= q3;
    when q3 =>
        stato_next <= q0;

    end case;
end process;

--OUTPUT
process(stato_corrente)
begin
    case stato_corrente is
        when idle =>
            ReqA      <= '0';
            read      <= '0';
            set       <= '0';
            rx        <= '0';
        when q0      =>
            ReqA      <= '0';
            read      <= '0';
            set       <= '0';
            rx        <= '0';
        when q1      =>
            ReqA      <= '1';
            read      <= '1';
        when q2      =>
            rx        <= '1';
            read      <= '0';
        when q3      =>
            rx        <= '0';
            ReqA      <= '0';
            set       <= '1';
    end case;
end process;

end Behavioral;

```

7.3.4 | Nodo B - Sistema Complessivo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NodoB is
    Port (
        ClockB: in std_logic;
        ResetB: in std_logic := '0';

        ---segnali handshake
        ReqA_B: in std_logic;
        OkB_A: out std_logic := '0';
        DatoA: in std_logic_vector(3 downto 0)
    );
end NodoB;

architecture Behavioral of NodoB is

--PARTE DI CONTROLLO
component nodob_Cp is
    Port (
        ClockB: in std_logic;
        ResetB: in std_logic := '0';

        ReqA: in std_logic;
        OkA: out std_logic := '0';

        --segnali controllo parte operativa
        EnRic: out std_logic;
        ReadRic: out std_logic;
        RdMEM: out std_logic;
        WrMEM: out std_logic;
        IncMEM: out std_logic;
        StopMEM: in std_logic;
        DoADD: out std_logic
    );
end component;

--PARTE OPERATIVA
component Nodod_Op is
    Port (
        DatoFromA: in std_logic_vector(3 downto 0);
        ClockB: in std_logic;
        ResetB: in std_logic := '0';
        EnRx: in std_logic;
        EnAdd : in STD_LOGIC;
        SendRx: in std_logic;
        WriteMem:in std_logic;
        ReadMem:in std_logic;
        STOP_SUM:out std_logic:= '0';
        SetMem: in std_logic := '0'
    );
end component;

signal Rx_enable: std_logic;
signal Rx_read: std_logic;
signal Mem_read: std_logic;
signal Mem_write: std_logic;

```

```

signal Mem_set:      std_logic;
signal Mem_stop:    std_logic;
signal Add_enable:  std_logic;

begin

Parte_Controllo:nodob_Cp
  port map(
    ClockB  => ClockB,
    ResetB  => ResetB,
    ReqA     => ReqA_B,
    OkA      => OkB_A,
    EnRic    => Rx_enable,
    ReadRic  => Rx_read,
    RdMEM    => Mem_read,
    WrMEM    => Mem_write,
    IncMEM   => Mem_set,
    StopMEM  => Mem_stop,
    DoADD    => Add_enable
  );

Parte_Operativa: Nodod_0p
  port map(
    DatoFromA  => DatoA,
    ClockB     => ClockB,
    ResetB     => ResetB,
    EnRx       => Rx_enable,
    EnAdd      => Add_enable,
    SendRx     => Rx_read,
    WriteMem   => Mem_write,
    ReadMem    => Mem_read,
    STOP_SUM   => Mem_stop,
    SetMem     => Mem_set
  );
end Behavioral;

```

7.3.5 | Nodo B - Parte Operativa

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Nodod_0p is
  Port (
    DatoFromA:  in std_logic_vector(3 downto 0);
    ClockB:    in std_logic;
    ResetB:    in std_logic := '0';
    EnRx:      in std_logic;
    EnAdd : in STD_LOGIC;
    SendRx:    in std_logic;
    WriteMem:in std_logic;
    ReadMem:in std_logic;
    STOP_SUM:out std_logic:= '0';
    SetMem:  in std_logic := '0'
  );
end Nodod_0p;

```

```

architecture structural of Nodod_Op is

component L_MEM_R is
generic(
    DataLen: natural :=3
);
Port(
    ClkIn:      in STD_LOGIC;
    RstIn:      in STD_LOGIC;
    EnRead:     in STD_LOGIC;
    EndRead:    out STD_LOGIC := '0';
    cnt:        in STD_LOGIC;
    OutData:    out std_logic_vector(DataLen downto 0)
);
end component;

component L_MEM_RW is
generic(
    DataLen: natural :=3
);
Port(
    ClkIn:      in STD_LOGIC;
    RstIn:      in STD_LOGIC;
    EnRead:     in STD_LOGIC;
    EnWrite:    in STD_LOGIC;
    EndRead:    out STD_LOGIC := '0';
    cnt:        in STD_LOGIC;
    InData:     in std_logic_vector(DataLen downto 0);
    OutData:    out std_logic_vector(DataLen downto 0)
);
end component;

component Registro is
generic(N: natural:=8);
port(
    ClockRx, ResetRx: in std_logic;
    InRx: in std_logic_vector(N downto 0);
    WriteRx: in std_logic;
    ReadRx: in std_logic;
    OutRx: out std_logic_vector(N downto 0)
);
end component;

component Addizionatore is
generic(
    N:integer:=3
);
port(
    A : in std_logic_vector(N downto 0);
    B : in std_logic_vector(N downto 0);
    EN:in std_logic;
    S : out std_logic_vector(N downto 0)
);
end component;

signal OutFromRx: std_logic_vector(3 downto 0):= "0000";
signal OutFromMem: std_logic_vector(3 downto 0):= "0000";
signal OutFromAdd: std_logic_vector(3 downto 0):= "0000";

```

```

begin

Memoria_ADD: L_MEMORY
    Port Map(
        ClkIn      => ClockB,
        RstIn      => ResetB,
        EnRead     => ReadMem,
        cnt         => SetMem,
        OutData    => OutFromMem
    );

Memoria_SUM: L_MEMORY_RW
    Port Map(
        ClkIn      => ClockB,
        RstIn      => ResetB,
        EnRead     => '0',
        --TEORICAMENTE NA RAM DA CUI LEGGO I RISULTATI,
        ---MA IN QUESTO ES NON ANDRFARLO
        EnWrite   => WriteMem,
        cnt         => SetMem,
        EndRead   => STOP_SUM,
        InData    => OutFromAdd
    );

RX: Registro
    port map(
        ClockRx   => ClockB,
        ResetRx   => ResetB,
        InRx      => DatoFromA,
        WriteRx   => EnRx,
        ReadRx    => SendRx,
        OutRx     => OutFromRx
    );

Sum:Addizionatore
    port map(
        A          => OutFromRx,
        EN         => EnAdd,
        B          => OutFromMem,
        S          => OutFromAdd
    );

end structural;

```

7.3.6 | Nodo B - Parte di Controllo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity nodob_Cp is
    Port (
        ClockB: in std_logic;
        ResetB: in std_logic := '0';

        ReqA: in std_logic;
        OkA: out std_logic := '0';

        --segnali controllo parte operativa

```

```

EnRic: out std_logic;
ReadRic: out std_logic;
RdMEM: out std_logic;
WrMEM: out std_logic;
IncMEM: out std_logic;
StopMEM: in std_logic;
DoADD: out std_logic
);
end nodob_Cp;

architecture Behavioral of nodob_Cp is

type stato is (idle,q0,q1,q2,q3);
signal stato_corrente : stato := idle;
signal stato_next: stato;

begin

--Process evoluzione stati
process (ClockB, ResetB)
begin
    if ResetB = '1' then
        stato_corrente <= idle;
    elsif(ClockB'event and ClockB = '1') then
        stato_corrente <= stato_next;
    end if;

    case stato_corrente is
    when idle =>
        if (ReqA = '0' or StopMEM = '1') then
            stato_next <= idle;
        elsif (ReqA = '1' and StopMEM = '0') then
            stato_next <= q0;
        end if;
    when q0 =>
        if ReqA = '0' then
            stato_next <= q1;
        elsif ReqA = '1' then
            stato_next <= q0;
        end if;
    when q1 =>
        stato_next <= q2;
    when q2 =>
        stato_next <= q3;
    when q3 =>
        stato_next <= idle;
    end case;
end process;

--OUTPUT
process(stato_corrente)
begin
    case stato_corrente is
    when idle =>
        EnRic    <= '0';
        ReadRic <= '0';
        RdMEM   <= '0';
        WrMEM   <= '0';

```

```

        IncMEM  <= '0';
        DoADD   <= '0';

        if StopMEM = '0' then
            okA <= '0';
        elsif StopMEM = '1' then
            okA <= '1';
        end if;
    when q0 =>
        EnRic   <= '1';
        ReadRic <= '0';
        okA     <= '1';
    when q1 =>
        EnRic   <= '0';
        ReadRic <= '1';
        RdMEM   <= '1';
    when q2 =>
        ReadRic <= '0';
        RdMEM   <= '0';
        DoADD   <= '1';
        WrMEM   <= '1';
    when q3 =>
        DoAdd   <= '0';
        WrMEM   <= '0';
        IncMEM  <= '1';
    end case;
end process;

end Behavioral;

```

7.3.7 | Mem RW

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity MemRw is
    generic(
        AddrLen: natural :=3;
        DataLen: natural :=3;
        NumberCells: natural :=7
    );
    port(
        CLK : in std_logic;
        RST : in std_logic;

        READ : in std_logic;
        WRITE : in std_logic;

        ADDR : in std_logic_vector(AddrLen downto 0);
        DATAIN : in std_logic_vector(DataLen downto 0);
        DATAOUT : out std_logic_vector(DataLen downto 0)
    );
end MemRw;

architecture behavioural of MemRw is

type mem_type is array (0 to NumberCells) of std_logic_vector(DataLen downto 0);
signal MEM : mem_type := (others => (others => '0'));

```

```

begin

RW_mem:process(CLK, WRITE, READ, RST)
variable MEM_addr: natural range 0 to NumberCells;
begin

    MEM_addr := conv_integer(ADDR);
    if (CLK' event and CLK='0') then

        if WRITE = '1' and READ = '0' then
            MEM(MEM_addr) <= DATAIN;
        end if;

        if READ = '1' and WRITE = '0' then
            DATAOUT <= MEM(MEM_addr);
        end if;

        if RST = '1' and WRITE = '0' and READ = '0' then
            MEM <= (others => (others => '0'));
        end if;
    end if;

end process;
end behavioural;

```

7.3.8 | Mem R

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity MemR is
    generic(
        AddrLen: natural :=3;
        DataLen: natural :=3;
        NumberCells: natural :=7
    );
    port(
        CLK : in std_logic;

        READ : in std_logic;

        ADDR : in std_logic_vector(AddrLen downto 0);
        DATAOUT : out std_logic_vector(DataLen downto 0)
    );
end MemR;

architecture behavioural of MemR is

type mem_type is array (0 to NumberCells) of std_logic_vector(DataLen downto 0);
signal MEM : mem_type := (
"0001",
"0010",
"0011",
"0100",
"0101",
"0110",
"0111",

```

```
"1000"  
);  
  
begin  
  
R_mem:process(CLK, READ)  
variable MEM_addr: natural range 0 to NumberCells;  
begin  
    MEM_addr := conv_integer(ADDR);  
    if (CLK' event and CLK='0') then  
        if READ = '1' then  
            DATAOUT <= MEM(MEM_addr);  
        end if;  
    end if;  
end process;  
end behavioural;
```

8 | Processore

A partire dall'implementazione fornita di un 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.
- Opzionale, si descriva il funzionamento del processore in merito alle istruzioni di input/output
- Solo ove possibile, si sintetizzi il processore su FPGA.

8.1 | Analisi Architettura

8.1.1 | Introduzione

Il processore MIC-1 nasce come interprete hardware del bytecode generato da un programma Java: nello specifico viene considerato il sottoinsieme delle istruzioni che lavorano solo sugli interi della Java Virtual Machine, ovvero la Integer Java Virtual Machine (IJVM).

Inoltre presenta un'architettura a stack e poiché è una macchina che ha più bus, introduce il concetto di parallelismo interno per cercare di ottimizzare al meglio le prestazioni della singola istruzione.

Il nostro intento non è utilizzare tale processore nel suo ruolo nativo, bensì analizzarne sia l'architettura, la quale ha il compito di implementare le istruzioni ISA - Instruction Set Architecture, sia la tecnica di microprogrammazione che utilizza.

8.1.2 | Datapath

La parte operativa del processore di Tanenbaum, mostrata in figura 8.1, è composta da 10 registri, due bus tramite cui essi possono scambiare dati e un ALU, che consente di fare operazioni logico/aritmetiche di base, inoltre troviamo uno shift register, utile in alcune operazioni matematiche, quali moltiplicazioni e divisioni binarie.

I registri necessari alla comunicazione con la memoria sono 2 coppie: una è dedicata alla lettura delle istruzioni composta dal **PC** - Program Counter e l'**MBR** - Memory Byte Register, mentre l'altra coppia si occupa della lettura e scrittura dei dati con i registri **MAR** - Memory Address Register e **MDR** - Memory Data Register; Risulta evidente che area dati e area istruzioni sono separate.

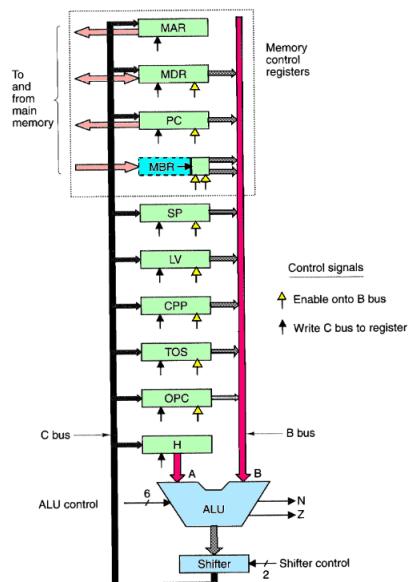


Figura 8.1: Unità operativa del processore MIC-1.

I registri rimanenti sono il registro H (registro di Holding), che permette di contenere uno degli operandi dell'ALU, i registri CPP, LV, OPC e i registri necessari alla gestione dello stack, ovvero il registro TOS (Top of Stack) e SP (Stack Pointer) che ci permettono di accedere alla testa dello stack, rispettivamente all'indirizzo e al valore.

Il registro LV serve per puntare alle variabili di appoggio che utilizza la macchina Java, mentre CPP è un puntatore a valori costanti, anch'esso utilizzato per semplificare la gestione della macchina virtuale, infine OPC permette di appoggiare qualcosa di utile nella micro-istruzione.

Per quanto riguarda la comunicazione tra i registri: il bus B permette di caricare su di esso il valore di un registro, mentre il bus C permette di scrivere su un registro il contenuto presente su di esso.

Dunque i due bus rappresentano rispettivamente un multiplexer e un demultiplexer, quindi se vogliamo scrivere il contenuto del registro X_i sul registro X_j dobbiamo selezionare con il multiplexer X_i e sul demultiplexer X_j .

I due bus però, come notiamo dalla figura 8.1, sono collegati tramite l'ALU e una delle sue possibili funzioni è far passare l'operando B inalterato, tale scelta è motivata dal fatto che quando dimensioniamo il clock dobbiamo considerare il percorso più lungo e dato che non si può ipotizzare di avere un clock a lunghezza variabile a seconda dell'istruzione considerata, si preferisce semplificare il circuito imponendo il passaggio per l'ALU.

8.1.3 | Unità di Controllo e Microistruzioni

L'unità di controllo del processore MIC-1 è realizzata in logica microprogrammata, quindi sono presenti:

- Un generatore degli indirizzi di partenza, ovvero l'indirizzo della micro-ROM a cui si accede per eseguire una data operazione.
- La memoria di controllo, ovvero la micro-ROM stessa.
- Il Program Counter della micro-ROM (MPC).

Le istruzioni ISA sono le istruzioni di interfaccia con il programmatore, ad ogni una di esse corrisponde una microprocedura che è costituita da un blocco di microistruzioni.

Per eseguire un codice operativo eseguiamo una microprocedura, ma la difficoltà consiste nel riuscire a determinare da un codice operativo qual è l'indirizzo di memoria da cui partire per operare con le microistruzioni.

Un altro problema è come ottimizzare la memoria: spesso istruzioni differenti condividono parte delle sequenze da eseguire, occorre organizzare la memoria in modo da individuare e scrivere un'unica volta le microistruzioni comuni.

La tecnica adottata nel processore MIC-1 è di includere in ciascuna microistruzione un campo **Next-Address** che indichi la localizzazione della successiva microistruzione da eseguire, dunque nel caso di un salto occorre anche capire in che modo modificare tale indirizzo così da ottenere il nuovo valore corretto. Nel processore MIC-1 sono necessari 29 segnali di controllo, questi bit servono a determinare il percorso dei dati per un singolo ciclo di clock, viene dunque aggiunta una parte necessaria a determinare cosa effettuare al ciclo successivo: i campi Addr e JAM:

- Il campo Addr (a 9 bit) è necessario a individuare la microistruzione da eseguire al successivo ciclo di clock.
- il campo JAM (su 3 bit) serve a gestire le operazioni di salto.

Quando è necessario effettuare un salto, come detto, il valore del Next-Address viene opportunamente modificato prima di essere inserito nel PC, per fare ciò si utilizzano i bit del campo JAM e i bit N e Z provenienti dall'ALU; A seconda di tali valori, nel registro MPC copieremo il Next Address con diverse modifiche:

- Se il campo JAM vale 000, allora il Next Address viene copiato senza modifiche sull'MPC.
- Se JAMN è alto, se ne calcola l'OR con il flag N e se ne pone il risultato nel bit più significativo di MPC;
- Se JAMZ è alto, se ne calcola l'OR con il flag Z e se ne pone il risultato nel bit più significativo di MPC;
- Se JAMN e JAMZ sono entrambi alti, si calcola l'OR rispetto a entrambi i flag;

- Se è alto il bit JMPC si effettua l'OR tra gli 8 bit di MBR e gli 8 bit meno significativi di next address e il risultato viene posto in MPC.

8.2 | Studio Istruzione 1 - BIPUSH

La prima istruzione che abbiamo studiato è la BIPUSH, che effettua il push del byte specificato sullo stack, è stata scelta per lo più per prendere familiarità con i vari tools.

8.2.1 | Simulazione

Per l'analisi del funzionamento del nostro microprogramma abbiamo usato gli strumenti visti a lezione, in primis abbiamo rigenerato i file .vhd della RAM e del control store con il comando cmake, successivamente abbiamo utilizzato la utility gtkwave per leggere tutti i segnali associati al nostro file testbench.

La microistruzione è così definita:

```
bipush = 0x10
SP = MAR = SP + 1
PC = PC +1; fetch
MDR = TOS = MBR; wr; goto main
```

Questa istruzione è abbastanza semplice, descriviamo ora il flusso delle istruzioni in particolare:

1. Predispone il registro MAR e SP in modo che lo stack possa crescere inserendo il nuovo byte.
2. Viene incrementato il pc, poiché il byte operando per questa istruzione è già stato pre-caricato dal main, ma devo comunque eseguire il fetch per caricare in MBR l'opcode successivo, cioè il byte da inserire nello stack.
3. A seguito del fetch in MBR c'è il byte che vogliamo inserire nello stack, quindi lo carichiamo in MDR e diamo il comando write, così da averlo disponibile dopo due cicli dove serve.

È stato realizzato un semplice programma per testare adeguatamente l'istruzione, andando a modificare il file program.jvm, viene riportato di seguito:

```
.main
    BIPUSH 0xA
    BIPUSH 0xE
    HALT
.endmethod
```

Infine è riportata la simulazione ottenuta, visibile in 8.2, è possibile notare come l'evoluzione dei registri per ogni bipush impiega 4 colpi di clock e le prime due operazioni della microistruzione sono eseguite in parte in contemporanea.

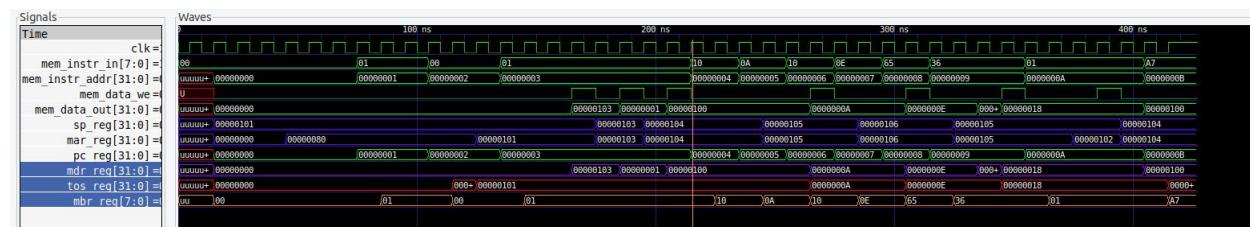


Figura 8.2: Simulazione del comportamento della BIPUSH.

8.3 | Studio Istruzione 2 - IAND

La seconda istruzione che abbiamo scelto è la IAND, essa effettua una AND bit a bit dei due valori sullo stack.

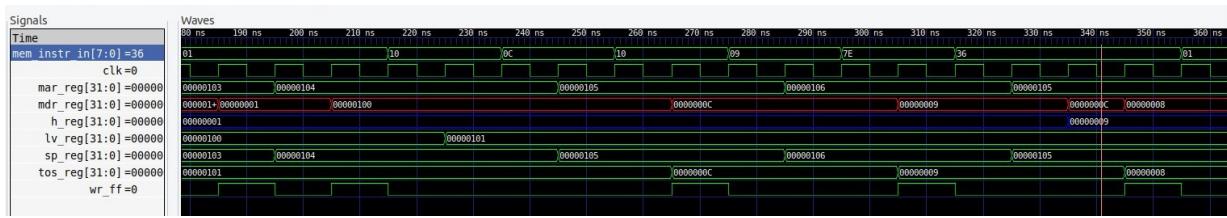


Figura 8.3: Simulazione del comportamento di IAND.

8.3.1 | Simulazione

Abbiamo provveduto alla sua analisi mediante simulazione in maniera analoga all'istruzione precedente, per prima cosa abbiamo analizzato il codice della singola microistruzione:

```
iand = 0xB6
      MAR = SP = SP - 1; rd
      H = TOS
      MDR = TOS = MDR AND H; wr; goto main
```

Anche in questo caso descriviamo il flusso nel particolare:

1. La prima delle 2 parole è già in TOS, quindi avvia la lettura della seconda che si trova a SP - 1.
2. Copia in H la prima parola, la seconda sarà in MDR al termine di questo ciclo.
3. MDR viene aggiornato con la and e si avvia la scrittura su SP - 1; il multiassegnamento consente di mantenere aggiornato anche TOS.

È stato realizzato un programma per testare adeguatamente l'istruzione, è molto semplice: utilizza due istruzioni BIPUSH per inserire i valori 0xC e 0x9 nello stack, dopodiché avviene la IAND e infine il risultato è posto in una variabile chiamata a; Il codice è riportato di seguito:

```
.main
  .var
    a
  .endvar
  BIPUSH 0xC
  BIPUSH 0x9
  IAND
  ISTORE a
  HALT
.endmethod
```

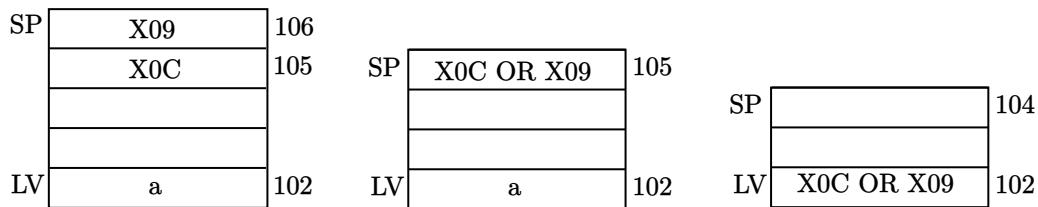


Figura 8.4: Evoluzione dello stack durante le operazioni di IAND.

Studiando la simulazione, visibile in 8.3, possiamo notare l'evoluzione del registro SP che inserisce i due operandi agli indirizzi 105 e 106, mentre il risultato per come è definita la IAND andrà in 105, infine la ISTORE esegue il pop dello stack e inserisce il valore nella variabile a.

8.3.2 | Modifica

La modifica dell'istruzione è stata effettuata sostituendo l'istruzione di AND bit a bit con l'istruzione di OR bit a bit, sempre riutilizzando il programma precedente.

Ciò non ha creato alcun tipo di problema al micro-compilatore, perché non abbiamo aggiunto codice, quindi lasciato inalterata la dimensione e le locazioni di memoria in cui è memorizzata la micro-istruzione. Di seguito viene riportato il codice della IAND modificata come descritto:

```
iand = 0xB6
    MAR = SP = SP - 1; rd
    H = TOS
    MDR = TOS = MDR OR H; wr; goto main
```

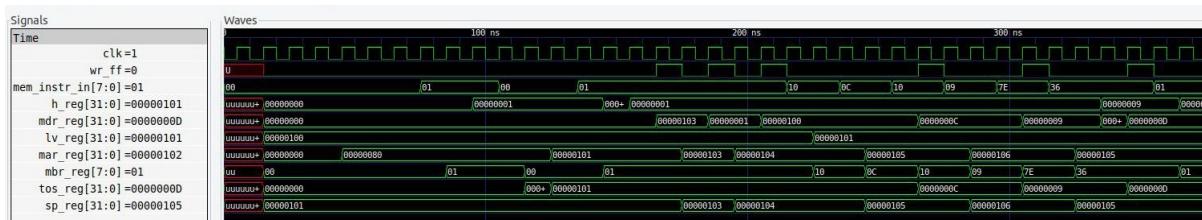


Figura 8.5: Simulazione del comportamento di IAND modificato.

8.4 | Studio Istruzione 3 - IFLT

Infine abbiamo analizzato e modificato l'istruzione IFLT, essa esegue il pop di una parola ed effettua il salto alla locazione corrente + offset se la parola è negativa.

8.4.1 | Simulazione

Come per le altre istruzioni, descriviamo prima il codice che ne definisce il comportamento:

```
iflt = 0x9D:
    MAR = SP = SP - 1; rd
    OPC = TOS
    TOS = MDR
    N = OPC; if (N) goto T; else goto F
```

1. Il salto avviene in base alla parola in cima allo stack, quindi devo in ogni caso farne il pop.
2. Nel mentre salva temporaneamente il TOS in OPC.
3. Aggiorno la cima dello stack in TOS.
4. Usiamo OPC per valutare il bit N (negative); se N è settato salta a T, altrimenti a F.

T ed F sono etichette ed indicano due precise zone della microrom dove sono definite alcune operazioni. T è simile alla goto, deve fare il fetch del secondo byte di offset dell'istruzione e poi effettuare il salto.

```
T: OPC = PC - 1; fetch; goto goto_condizionato
```

Invece F evita il secondo byte di offset che non occorre e pre-carica il prossimo opcode.

```
F: PC = PC +1
    PC = PC +1; fetch;
    goto Main
```

8.4.2 | Modifica

La modifica in questo caso è stata effettuando provando a far eseguire più operazioni alla microistruzione, confronta le due parole in cima allo stack ed effettua un salto alla locazione corrente + offset se sono uguali.

Il codice dell'istruzione è il seguente:

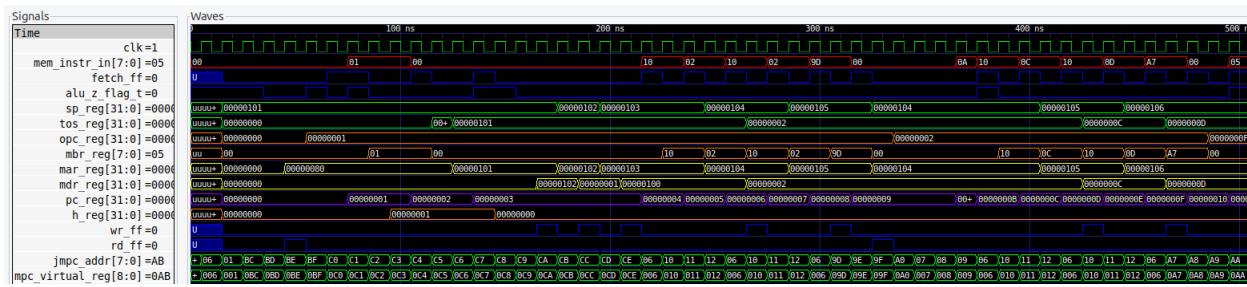


Figura 8.6: Simulazione della IFLT.

```

iflt = 0x9D:
    MAR = SP - 1; rd
    OPC = TOS
    H = MDR
    Z= OPC - H; if (Z) goto T; else goto F

```

1. Prima di tutto andiamo a leggere il penultimo valore dello stack.
2. Nel mentre salviamo il valore del TOS in OPC.
3. Dopo due cicli di clock avremo il valore in MDR e lo andiamo a mettere in H per poterlo valutare.
4. Andiamo a valutare Z come differenza tra OPC e H, tale flag sarà 1 se vale 0, quindi se sono uguali.

Abbiamo lasciato invariato il comportamento delle etichette T ed F poiché uguali a tutte le istruzioni di salto condizionato, quindi abbiamo testato l'istruzione nei due casi utilizzando un semplice programma.

```

.main
BIPUSH 0x2
BIPUSH 0x1
IFLT salto
BIPUSH 0xC
BIPUSH 0xD
GOTO fine
salto: BIPUSH 0xF
fine: HALT
.endmethod

```

In questo caso la IFLT troverà un valore diverso da zero in cima allo stack, ovvero il risultato 2-1, pertanto si procede con l'esecuzione delle due bipush e poi terminare in halt.

```

.main
BIPUSH 0x2
BIPUSH 0x2
IFLT salto
BIPUSH 0xC
BIPUSH 0xD
GOTO fine
salto: BIPUSH 0xF
fine: HALT
.endmethod

```

In quest'altro caso la IFLT trova in cima allo stack due valori uguali, pertanto OPC - H sarà pari a zero e la IFLT indirizzerà in questo caso al ramo true, quindi effettua il salto condizionato in cui viene ad effettuata la singola bipush 0XF per poi terminare il flusso d'esecuzione.

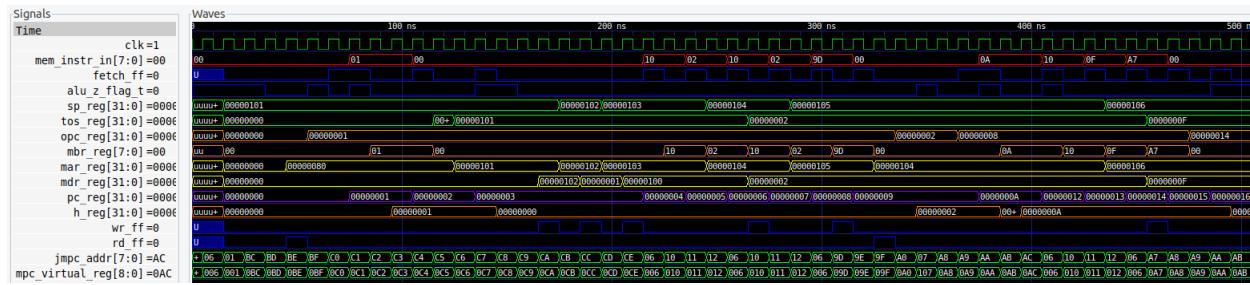


Figura 8.7: Simulazione della IFLT modificata nel caso 1.

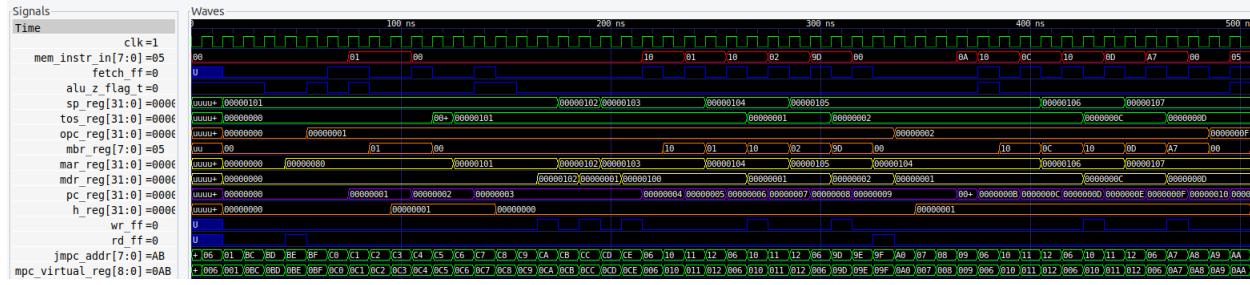


Figura 8.8: Simulazione della IFLT modificata nel caso 2.

9 | Interfaccia Seriale

Punto 1 Sfruttando l'implementazione fornita dalla Digilent di un dispositivo UART (componente RS232RefComp.vhd), progettare e implementare in VHDL un sistema costituito da 2 nodi A e B collegati tra loro mediante una interfaccia seriale.

Il sistema A acquisisce una stringa di 8 bit dall'utente (mediante gli switch della board di sviluppo) e la invia mediante la seriale al sistema B, che la manda in output sui led della board di sviluppo.

Punto 2 Implementare uno dei seguenti sistemi a scelta dello studente:

- **2_UART_MEM:** come variante dell'esercizio 8.1, il sistema A invia al sistema B tramite l'interfaccia seriale N stringhe di 8 bit contenute all'interno di una memoria ROM, le stringhe ricevute vengono memorizzate in una memoria locale a B.

Il progetto deve prevedere che A utilizzi un componente contatore per scandire le N stringhe da inviare.

- **UART_PC:** il sistema realizza la comunicazione fra un nodo A rappresentato da un componente sintetizzato su un FPGA e un nodo B rappresentato da un terminale seriale in esecuzione su PC (es. Termite), previa connessione di PC e board tramite dispositivo fisico RS232 (uno degli endpoint di comunicazione è rappresentato dal PC).

Il componente A acquisisce una stringa di 8 bit che rappresenta un carattere in codifica ASCII fornita dall'utente mediante gli switch della board di sviluppo, e la invia mediante il dispositivo UART al terminale B in esecuzione sul PC, in cui il carattere viene visualizzato.

Allo stesso modo, il componente deve essere in grado di ricevere attraverso lo stesso dispositivo UART (oppure una seconda UART) un carattere trasmesso dal terminale e mostrarlo sui led.

9.1 | Introduzione

L'UART (Universal Asynchronous Receiver-Transmitter) è un dispositivo hardware che converte flussi di bit di dati da un formato parallelo a un formato seriale asincrono o viceversa con la comunicazione può essere di tre tipi: simplex, half duplex, full duplex.

Per garantire il corretto funzionamento i dispositivi di trasmissione e ricezione devono accordarsi su:

- velocità di bit (baud rate)

- lunghezza dei caratteri
- numero di bit di parità
- numero di bit di stop

La configurazione tipica è costituita da 8 bit dati, nessun bit di parità e un bit di stop.

Questo componente, come visibile nella figura 9.1, è costituito da 6 ingressi e 7 uscite, inoltre è possibile considerarlo come formato da due componenti: uno per trasmettere informazioni seriali e uno per ricevere informazioni seriali.

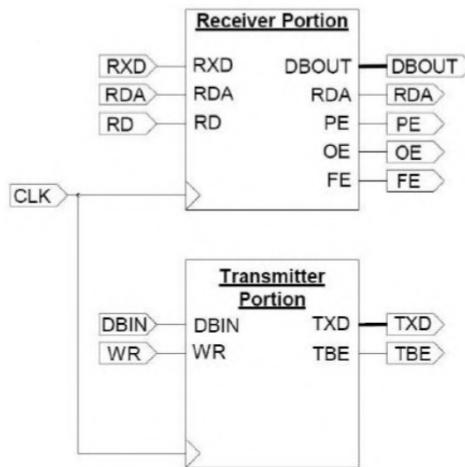


Figura 9.1: Architettura UART divisa in porzione RX e TX.

Analizziamo innanzitutto gli ingressi:

- RXD: il dato seriale ricevuto;
- DBIN : il dato parallelo di input che viene ricevuto;
- RDA - read data available: che indica che il dato è disponibile per essere letto;
- RD: uno strobe per la lettura;
- WR: uno strobe per la scrittura;
- RST: il segnale di reset.

Analizziamo adesso invece le uscite:

- TXD: il dato seriale di output che dev'essere trasmesso;
- DBOUT: il dato parallelo di output che viene trasmesso;
- RDA: lo stesso segnale visto in ingresso, che è un segnale sia di ingresso che di uscita;
- TBE - Transfer Buffer Empty: quando il trasmettitore UART ha completato l'invio di un carattere e il buffer di trasmissione è vuoto. Questo viene trattato come un'indicazione del fatto che non rimane alcun dato da trasmettere;
- PE - Parity Error: è fallito il controllo di parità e quindi bisogna procedere alla ritrasmissione;
- OE - Overrun Error: il dato viene copiato dallo shift register al registro prima che il precedente sia stato svuotato dalla CPU.

Ad esempio, può capitare quando riceviamo un carattere, ma prima che il processore potesse leggerlo, arriva un secondo carattere e lo sovrascriviamo nel registro. In questo caso il dato sovrascritto è stato perso e si genera un errore;

- FE - Framing Error: ci aspettiamo un carattere di sincronizzazione in un determinato istante che però non arriva, questo significa che si è persa la sincronizzazione sull'intero frame e quindi anche vecchi caratteri letti possono essere falsi.

9.2 | Descrizione Soluzione

9.2.1 | Punto 1 - UART a Tappo

Innanzitutto mostriamo lo schematico che descrive la soluzione in figura 9.2, come possiamo notare abbiamo una rete complessiva che dall'esterno manda nei vari componenti che la compongono una serie di segnali.

Il primo componente che abbiamo partendo dalla sinistra è il nodo A, questo prende in ingresso un segnale di 8 bit posto come input al sistema e che viene dato tramite gli switch su board.

Il nodo A produce un'uscita di 8 bit che viene manda in ingresso all'UART nel segnale DBIN, la quale prende anche in ingresso il clock.

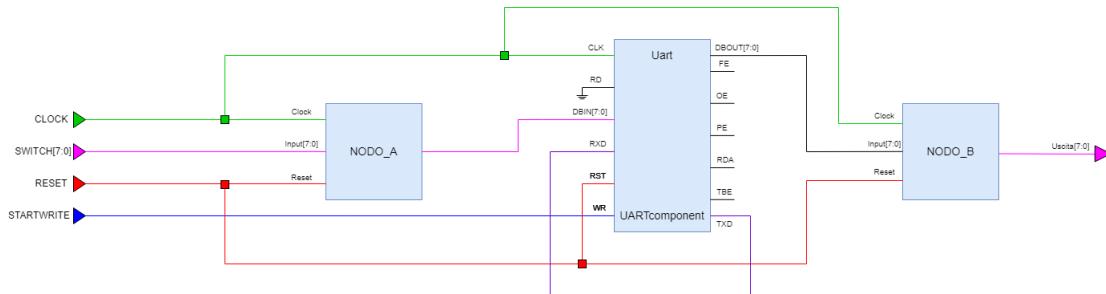


Figura 9.2: Schematico della soluzione al punto 1.

L'UART trasmetterà il segnale del nodo A sull'uscita DBOUT che viene mandata in ingresso al nodo B, il quale prende quest'ingresso e lo manda in uscita alla nostra rete complessiva, ricordiamo che quest'uscita complessiva dev'essere mostrata sui led della scheda.

9.2.2 | Punto 2 - UART MEM

Per questa soluzione sono stati sviluppati due nodi, ognuno con la propria memoria e che comunicano attraverso un protocollo di handshake.

Protocollo è stato costruito in maniera simile all'esercizio 7 sfruttando 2 segnali, **ReqA** e **OkB**, i loro fronti d'onda permettono di scandire la comunicazione e l'eventuale necessità di ritrasmettere i dati.

Il protocollo, rappresentato nella figura 9.3, è suddiviso in intervalli ben precisi:

- L'intervallo 1 rappresenta il momento in cui il nodo A richiede una elaborazione al nodo B e aspetta una sua risposta.
- L'intervallo 2 rappresenta il momento in cui il nodo B risponde e entrambi si predispongono alla comunicazione.
- L'intervallo 3 è il momento in cui termina la comunicazione e il nodo B rileva eventuali errori.

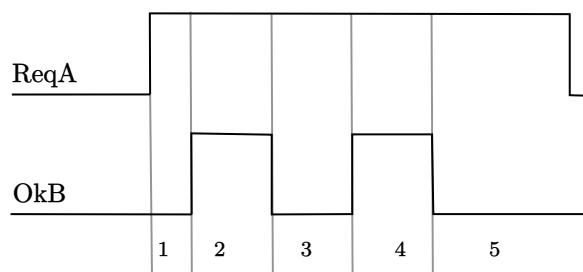


Figura 9.3: Descrizione protocollo.

- L'intervallo 4 è opzionale, infatti serve ad indicare che serve una ritrasmissione dei dati poiché sono stati rilevati degli errori.
- L'intervallo 5 è quello finale, il nodo B non ha rilevato errori ed aspetta che il nodo A abbassi il segnale ReqA per tornare in idle.

Nodo A in questa entità avviene la trasmissione prelevando i dati da una ROM, quindi la struttura è molto semplice in quanto vi è una UC con all'interno un contatore che comanda la memoria e l'interfaccia UART.

Possiamo notare che alcuni stati sono simili a quelli del trasmettitore del esercizio 7, però possiamo notare che lo stato Q3 è inedito, poiché esso si occupa di controllare che la comunicazione tramite UART sia terminata e inoltre aspetta che il nodo B fornisca un feedback sul messaggio appena ricevuto, in questo modo può tornare allo stato Q2 ed effettuare di nuovo una trasmissione.

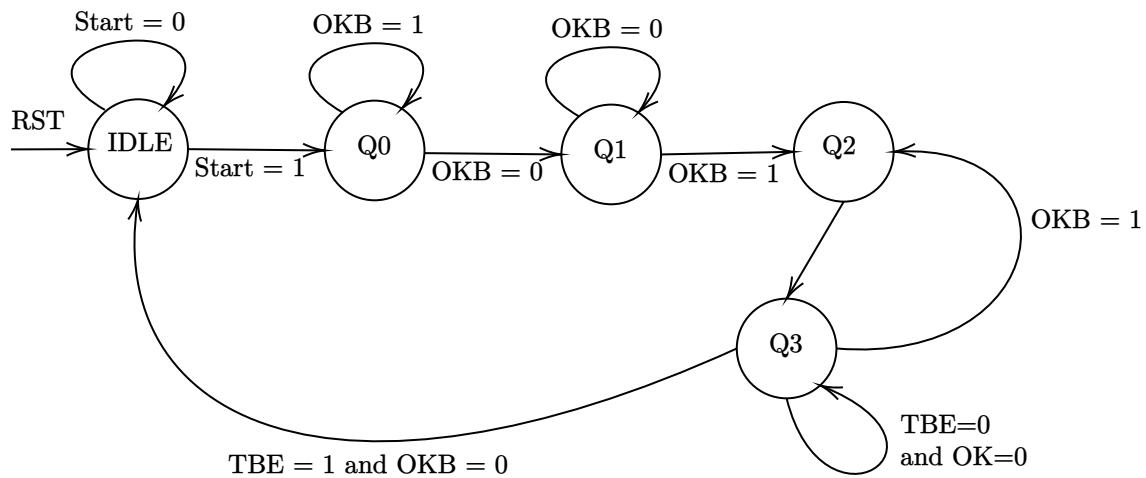
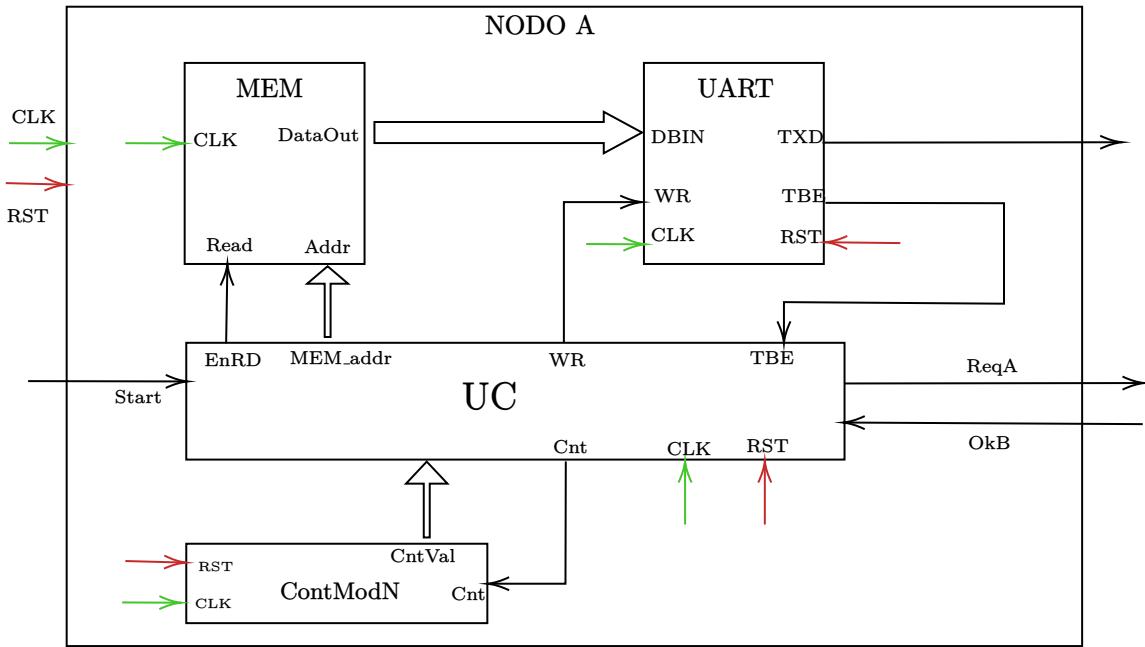
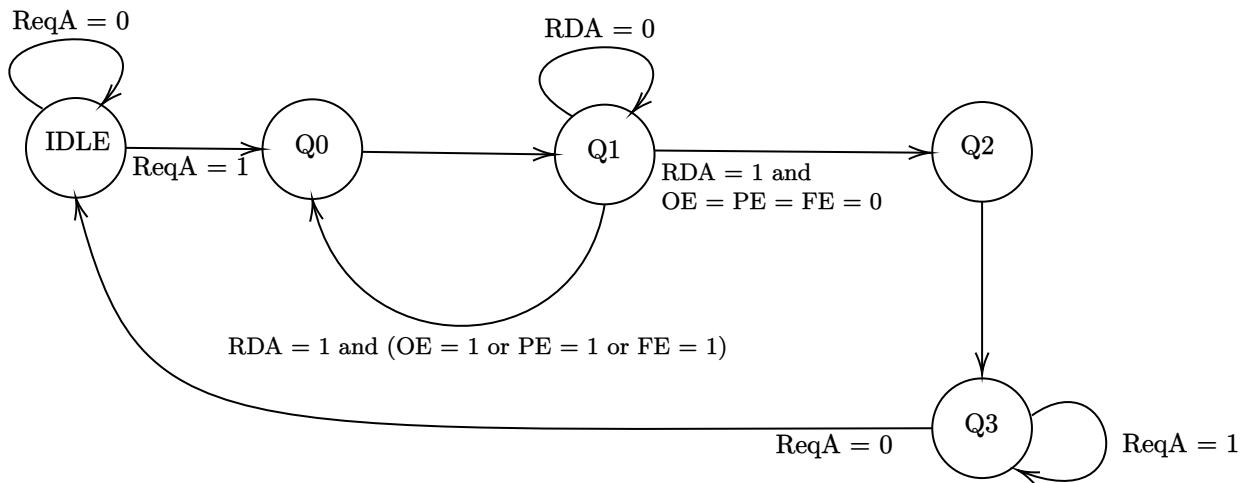


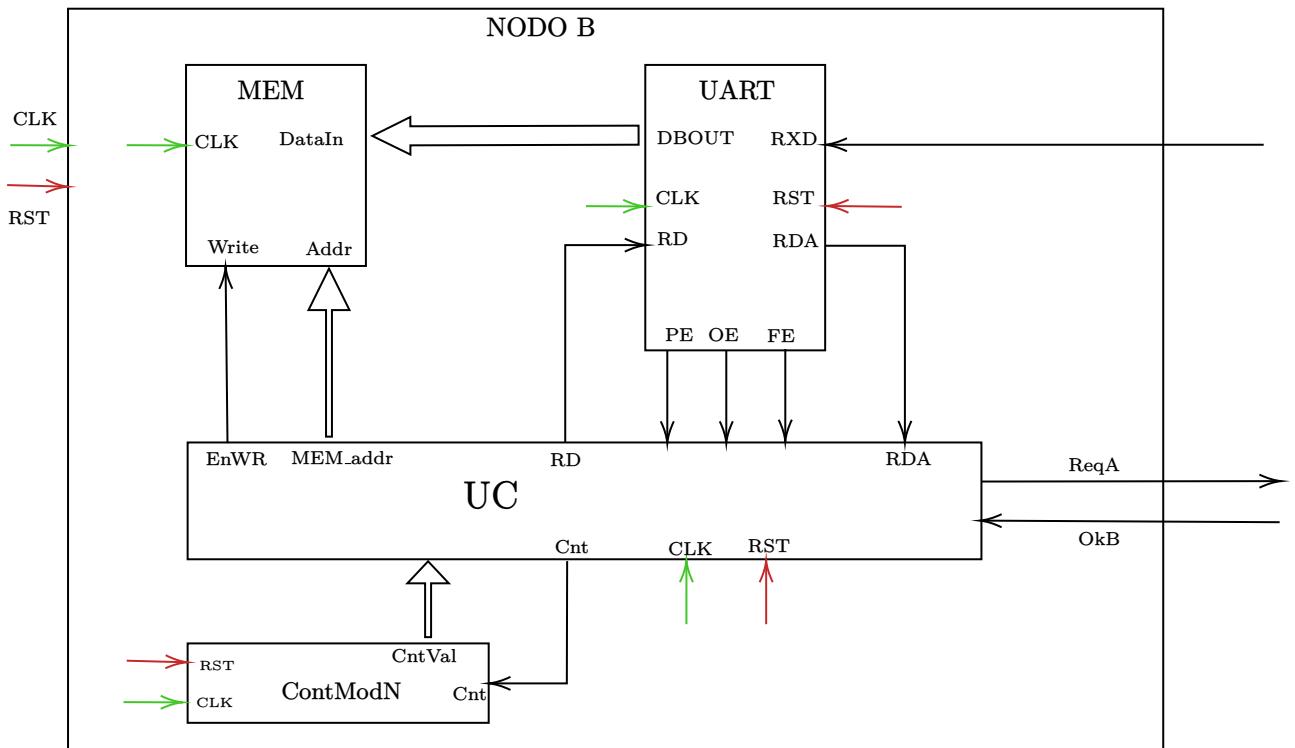
Figura 9.4: Automa del nodo A.

**Figura 9.5:** Struttura del nodo A.

Nodo B questo nodo è simile al precedente, deve settare la uart in ricezione e salvare il messaggio in memoria, la struttura è nella figura 9.7.

Anche qui abbiamo lo stato Q1 che aspetta la fine della ricezione e eventualmente aspetta una ritrasmissione dopo che ha controllato i bit OE, PE, FE che indicano la presenza di alcuni tipi di errori.

**Figura 9.6:** Automa del nodo B.

**Figura 9.7:** Struttura del nodo B.

9.3 | Testbench

**Figura 9.8:** Simulazione della UART al punto 1.

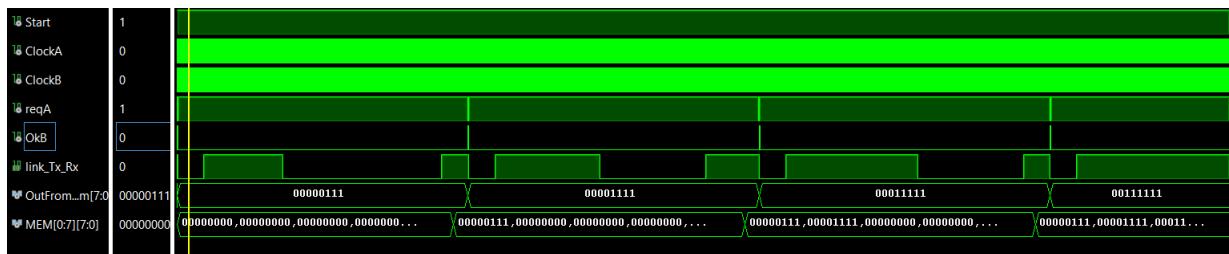


Figura 9.9: Simulazione della UART al punto 2.

9.4 | Sintesi

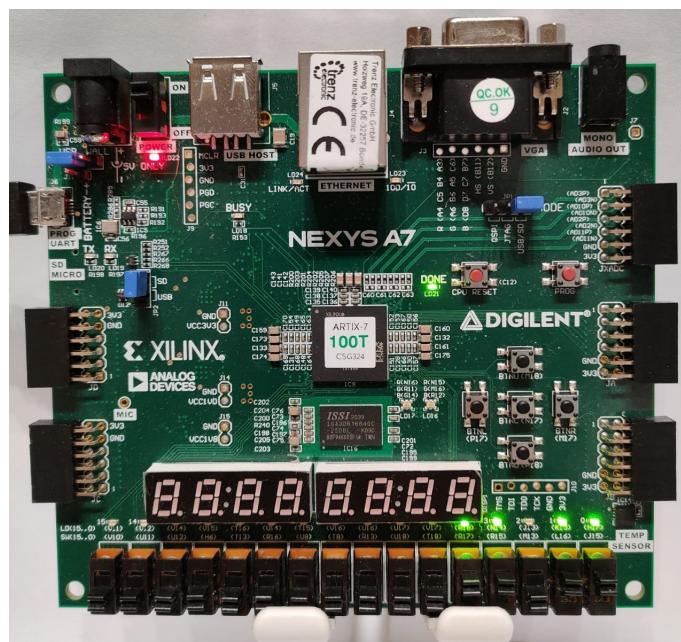


Figura 9.10: Sintesi della UART a tappo.

9.5 | Codice

9.5.1 punto 1 - Sistema Complessivo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SistemaComplessivo is
    generic (N:integer:=7);
    Port (
        CLOCK,RESET:in std_logic;
        SWITCH:in std_logic_vector(N downto 0);
        LEDS:out std_logic_vector(N downto 0);
        STARTWRITE:in std_logic:='0'
    );
end SistemaComplessivo;
architecture strucrtural of SistemaComplessivo is

    signal OutFromA: std_logic_vector(N downto 0);
    signal OutFromB: std_logic_vector(N downto 0);
    signal UartIn: std_logic_vector(N downto 0);
    signal UartOut: std_logic_vector(N downto 0);

```

```

signal linkTx_Rx: std_logic :='0';

component Nodo is
generic( N:integer:=7);
Port (
    Input :in std_logic_vector( N downto 0):=( others =>'0');
    Clock,Reset :in std_logic:='0';
    Uscita :out std_logic_vector( N downto 0):=( others =>'0')
 );
end component;

component ButtonDebouncer is
generic (
    CLK_period: integer := 10; -- periodo del clock in nanosec
    btn_noise_time: integer := 10000000 --durata dell'oscillazione in nanosec
 );
Port ( RST : in STD_LOGIC;
    CLK : in STD_LOGIC;
    BTN : in STD_LOGIC;
    CLEARED_BTN : out STD_LOGIC
 );
end component;

component UARTcomponent is
Generic (
    --@48MHz
    BAUD_DIVIDE_G : integer := 26;           --115200 baud
    BAUD_RATE_G   : integer := 417

    --@26.6MHz
    BAUD_DIVIDE_G : integer := 14;           --115200 baud
    BAUD_RATE_G   : integer := 231
);
Port (
    TXD      : out      std_logic      := '1';
    RXD      : in       std_logic;
    CLK      : in       std_logic;
    DBIN     : in       std_logic_vector (7 downto 0);
    -- Input parallel data to be transmitted
    DBOUT    : out      std_logic_vector (7 downto 0);
    -- Received parallel data output
    RDA      : inout    std_logic;
    TBE      : out      std_logic      := '1';
    RD       : in       std_logic;
    WR       : in       std_logic;
    PE       : out      std_logic;
    FE       : out      std_logic;
    OE       : out      std_logic;
    RST      : in       std_logic      := '0');
end component;

begin
Pulsante_RESET : ButtonDebouncer
    Port map (

```

```

RST =>RESET,
CLK=>CLOCK,
BTN =>RESET,
CLEARED_BTN =>RESET_pulito
);

Pulsante_STARTWRITE : ButtonDebouncer
  Port map (
    RST =>RESET_pulito,
    CLK=>CLOCK,
    BTN =>STARTWRITE,
    CLEARED_BTN =>STARTWRITE_pulito
  );

NODO_A: Nodo
  Port map (
    Input =>SWITCH,
    Clock=>CLOCK,
    Reset =>RESET_pulito,
    Uscita =>OutFromA
  );
Uart : UARTcomponent generic map(
  BAUD_DIVIDE_G => 14,
  BAUD_RATE_G  => 231
)
  Port map (
    TXD =>linkTx_Rx,
    RXD =>linkTx_Rx,
    CLK=>CLOCK,
    DBIN =>OutFromA,
    DBOUT =>UartOut,
    RDA      =>OPEN,
    TBE=>OPEN,
    RD=>'0',
    WR=>STARTWRITE_pulito,
    RST=>RESET_pulito
  );

NODO_B: Nodo
  Port map (
    Input =>UartOut,
    Clock=>CLOCK,
    Reset =>RESET_pulito,
    Uscita =>OutFromB
  );
LEDs<=OutFromB;
end structural;

```

9.5.2 | Punto 1 - Mapping

```

## Clock signal
set_property -dict { PACKAGE_PIN E3   IO_STANDARD LVCMOS33 } [get_ports { CLOCK }]; #IO_L12P_T1_MRCC
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { CLOCK }];

##Switches
set_property -dict { PACKAGE_PIN J15   IO_STANDARD LVCMOS33 } [get_ports { SWITCH[0] }]; #IO_L24N_T3_1
set_property -dict { PACKAGE_PIN L16   IO_STANDARD LVCMOS33 } [get_ports { SWITCH[1] }]; #IO_L3N_TO_D

```

```

set_property -dict { PACKAGE_PIN M13 } IOSTANDARD LVCMOS33 } [get_ports [ SWITCH[2] ]]; #IO_L6N_TO_DQ[2]
set_property -dict { PACKAGE_PIN R15 } IOSTANDARD LVCMOS33 } [get_ports [ SWITCH[3] ]]; #IO_L13N_T2_DQ[3]
set_property -dict { PACKAGE_PIN R17 } IOSTANDARD LVCMOS33 } [get_ports [ SWITCH[4] ]]; #IO_L12N_T1_DQ[4]
set_property -dict { PACKAGE_PIN T18 } IOSTANDARD LVCMOS33 } [get_ports [ SWITCH[5] ]]; #IO_L7N_T1_DQ[5]
set_property -dict { PACKAGE_PIN U18 } IOSTANDARD LVCMOS33 } [get_ports [ SWITCH[6] ]]; #IO_L17N_T2_DQ[6]
set_property -dict { PACKAGE_PIN R13 } IOSTANDARD LVCMOS33 } [get_ports [ SWITCH[7] ]]; #IO_L5N_TO_DQ[7]
#set_property -dict { PACKAGE_PIN T8 } IOSTANDARD LVCMOS18 } [get_ports [ SW[8] ]]; #IO_L24N_T3_34_Sch=sw
#set_property -dict { PACKAGE_PIN U8 } IOSTANDARD LVCMOS18 } [get_ports [ SW[9] ]]; #IO_25_34_Sch=sw
#set_property -dict { PACKAGE_PIN R16 } IOSTANDARD LVCMOS33 } [get_ports [ SW[10] ]]; #IO_L15P_T2_DQ[10]
#set_property -dict { PACKAGE_PIN T13 } IOSTANDARD LVCMOS33 } [get_ports [ SW[11] ]]; #IO_L23P_T3_A0[11]
#set_property -dict { PACKAGE_PIN H6 } IOSTANDARD LVCMOS33 } [get_ports [ SW[12] ]]; #IO_L24P_T3_35_Sch=sw
#set_property -dict { PACKAGE_PIN U12 } IOSTANDARD LVCMOS33 } [get_ports [ SW[13] ]]; #IO_L20P_T3_A0[13]
#set_property -dict { PACKAGE_PIN U11 } IOSTANDARD LVCMOS33 } [get_ports [ SW[14] ]]; #IO_L19N_T3_A0[14]
#set_property -dict { PACKAGE_PIN V10 } IOSTANDARD LVCMOS33 } [get_ports [ SW[15] ]]; #IO_L21P_T3_DQ[15]

## LEDs
set_property -dict { PACKAGE_PIN H17 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[0] ]]; #IO_L18P_T2_A2[16]
set_property -dict { PACKAGE_PIN K15 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[1] ]]; #IO_L24P_T3_RS[17]
set_property -dict { PACKAGE_PIN J13 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[2] ]]; #IO_L17N_T2_A2[18]
set_property -dict { PACKAGE_PIN N14 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[3] ]]; #IO_L8P_T1_D11[19]
set_property -dict { PACKAGE_PIN R18 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[4] ]]; #IO_L7P_T1_D09[20]
set_property -dict { PACKAGE_PIN V17 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[5] ]]; #IO_L18N_T2_A1[21]
set_property -dict { PACKAGE_PIN U17 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[6] ]]; #IO_L17P_T2_A1[22]
set_property -dict { PACKAGE_PIN U16 } IOSTANDARD LVCMOS33 } [get_ports [ LEDS[7] ]]; #IO_L18P_T2_A1[23]
#set_property -dict { PACKAGE_PIN V16 } IOSTANDARD LVCMOS33 } [get_ports [ LED[8] ]]; #IO_L16N_T2_A1[24]
#set_property -dict { PACKAGE_PIN T15 } IOSTANDARD LVCMOS33 } [get_ports [ LED[9] ]]; #IO_L14N_T2_SR[25]
#set_property -dict { PACKAGE_PIN U14 } IOSTANDARD LVCMOS33 } [get_ports [ LED[10] ]]; #IO_L22P_T3_A0[26]
#set_property -dict { PACKAGE_PIN T16 } IOSTANDARD LVCMOS33 } [get_ports [ LED[11] ]]; #IO_L15N_T2_DQ[27]
#set_property -dict { PACKAGE_PIN V15 } IOSTANDARD LVCMOS33 } [get_ports [ LED[12] ]]; #IO_L16P_T2_C[28]
#set_property -dict { PACKAGE_PIN V14 } IOSTANDARD LVCMOS33 } [get_ports [ LED[13] ]]; #IO_L22N_T3_A0[29]
#set_property -dict { PACKAGE_PIN V12 } IOSTANDARD LVCMOS33 } [get_ports [ LED[14] ]]; #IO_L20N_T3_A0[30]
#set_property -dict { PACKAGE_PIN V11 } IOSTANDARD LVCMOS33 } [get_ports [ LED[15] ]]; #IO_L21N_T3_DQ[31]

## RGB LEDs
#set_property -dict { PACKAGE_PIN R12 } IOSTANDARD LVCMOS33 } [get_ports [ LED16_B ]]; #IO_L5P_TO_D00[32]
#set_property -dict { PACKAGE_PIN M16 } IOSTANDARD LVCMOS33 } [get_ports [ LED16_G ]]; #IO_L10P_T1_D01[33]
#set_property -dict { PACKAGE_PIN N15 } IOSTANDARD LVCMOS33 } [get_ports [ LED16_R ]]; #IO_L11P_T1_SR[34]
#set_property -dict { PACKAGE_PIN G14 } IOSTANDARD LVCMOS33 } [get_ports [ LED17_B ]]; #IO_L15N_T2_DQ[35]
#set_property -dict { PACKAGE_PIN R11 } IOSTANDARD LVCMOS33 } [get_ports [ LED17_G ]]; #IO_O_14_Sch=sw[36]
#set_property -dict { PACKAGE_PIN N16 } IOSTANDARD LVCMOS33 } [get_ports [ LED17_R ]]; #IO_L11N_T1_SR[37]

## 7 segment display
#set_property -dict { PACKAGE_PIN T10 } IOSTANDARD LVCMOS33 } [get_ports [ CA ]]; #IO_L24N_T3_A00_D1[38]
#set_property -dict { PACKAGE_PIN R10 } IOSTANDARD LVCMOS33 } [get_ports [ CB ]]; #IO_25_14_Sch=cb[39]
#set_property -dict { PACKAGE_PIN K16 } IOSTANDARD LVCMOS33 } [get_ports [ CC ]]; #IO_25_15_Sch=cc[40]
#set_property -dict { PACKAGE_PIN K13 } IOSTANDARD LVCMOS33 } [get_ports [ CD ]]; #IO_L17P_T2_A26_15[41]
#set_property -dict { PACKAGE_PIN P15 } IOSTANDARD LVCMOS33 } [get_ports [ CE ]]; #IO_L13P_T2_MRCC_14[42]
#set_property -dict { PACKAGE_PIN T11 } IOSTANDARD LVCMOS33 } [get_ports [ CF ]]; #IO_L19P_T3_A10_D2[43]
#set_property -dict { PACKAGE_PIN L18 } IOSTANDARD LVCMOS33 } [get_ports [ CG ]]; #IO_L4P_TO_D04_14[44]
#set_property -dict { PACKAGE_PIN H15 } IOSTANDARD LVCMOS33 } [get_ports [ DP ]]; #IO_L19N_T3_A21_VREF[45]
#set_property -dict { PACKAGE_PIN J17 } IOSTANDARD LVCMOS33 } [get_ports [ AN[0] ]]; #IO_L23P_T3_FOE[46]
#set_property -dict { PACKAGE_PIN J18 } IOSTANDARD LVCMOS33 } [get_ports [ AN[1] ]]; #IO_L23N_T3_FWE[47]
#set_property -dict { PACKAGE_PIN T9 } IOSTANDARD LVCMOS33 } [get_ports [ AN[2] ]]; #IO_L24P_T3_A01[48]
#set_property -dict { PACKAGE_PIN J14 } IOSTANDARD LVCMOS33 } [get_ports [ AN[3] ]]; #IO_L19P_T3_A22[49]
#set_property -dict { PACKAGE_PIN P14 } IOSTANDARD LVCMOS33 } [get_ports [ AN[4] ]]; #IO_L8N_T1_D12[50]
#set_property -dict { PACKAGE_PIN T14 } IOSTANDARD LVCMOS33 } [get_ports [ AN[5] ]]; #IO_L14P_T2_SRC[51]
#set_property -dict { PACKAGE_PIN K2 } IOSTANDARD LVCMOS33 } [get_ports [ AN[6] ]]; #IO_L23P_T3_35[52]
#set_property -dict { PACKAGE_PIN U13 } IOSTANDARD LVCMOS33 } [get_ports [ AN[7] ]]; #IO_L23N_T3_A02[53]

```

```
##Buttons
#set_property -dict { PACKAGE_PIN C12    IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }];
set_property -dict { PACKAGE_PIN N17    IOSTANDARD LVCMOS33 } [get_ports { RESET }];
#set_property -dict { PACKAGE_PIN M18    IOSTANDARD LVCMOS33 } [get_ports { BTNU }];
set_property -dict { PACKAGE_PIN P17    IOSTANDARD LVCMOS33 } [get_ports { STARTWRITE }];
#set_property -dict { PACKAGE_PIN M17    IOSTANDARD LVCMOS33 } [get_ports { BTNR }];
#set_property -dict { PACKAGE_PIN P18    IOSTANDARD LVCMOS33 } [get_ports { BTND }];
#set_property -dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports { BTND }]; #IO_L9N_T1_DQS_D
```

9.5.3 | Punto 2 - Nodo A

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

entity NodoA is
  Port (
    ClockA : in STD_LOGIC:='0';
    ResetA : in STD_LOGIC:='0';
    StartA: in STD_LOGIC:='0';
    OkB: in std_logic;
    ReqA: out std_logic;
    link_Tx_Rx:out STD_LOGIC
  );
end NodoA;

architecture structural of NodoA is

component UC_A is
  Port (
    ClockA, ResetA: in std_logic;
    Start: in std_logic:='0';

    ----handshake
    OkB: in std_logic;
    ReqA: out std_logic;

    ----UART
    Tbe: in std_logic:='0';
    EnWr: out std_logic:='0';

    ---MEM
    RD_mem: out std_logic;
    OutCont: out std_logic_vector(2 downto 0) := (others => '0')
  );
end component;

component MemR is
generic(
  AddrLen: natural :=2;
  DataLen: natural :=7;
  NumberCells: natural :=7
);
port(
  CLK : in std_logic;

  READ : in std_logic;

  ADDR : in std_logic_vector(AddrLen downto 0);

```

```

        DATAOUT : out std_logic_vector(DataLen downto 0)
    );
end component;

component UARTcomponent is
    Generic (
        --@48MHz
        BAUD_DIVIDE_G : integer := 26;           --115200 baud
        BAUD_RATE_G   : integer := 417

        --@26.6MHz
        BAUD_DIVIDE_G : integer := 14;           --115200 baud
        BAUD_RATE_G   : integer := 231
    );
    Port (
        TXD      : out      std_logic      := '1';
        RXD      : in       std_logic;
        CLK      : in       std_logic;
        DBIN     : in       std_logic_vector(7 downto 0);          -- Input port
        DBOUT    : out      std_logic_vector(7 downto 0);          -- Receivin
        RDA      : inout    std_logic;
        TBE      : out      std_logic      := '1';
        RD       : in       std_logic;
        WR       : in       std_logic;
        PE       : out      std_logic;
        FE       : out      std_logic;
        OE       : out      std_logic;
        RST      : in       std_logic      := '0');
    );
end component;

signal OutFromRom : std_logic_vector(7 downto 0):= (others => '0');
signal address : std_logic_vector(2 downto 0):= (others => '0');
signal tbeTx,writeTx,rdaTx:std_logic:='0';
signal EN_RD: std_logic;

begin
    UC: UC_A
    port map(
        ClockA  => ClockA,
        ResetA  => ResetA,
        Start    => StartA,
        ----handshake
        OkB     => OkB,
        ReqA    => ReqA,
        ----UART
        Tbe     => tbeTx,
        EnWr   => writeTx,
        ----MEM
        RD_mem  => EN_RD,
        OutCont => address
    );

```

```

UartTx:UARTcomponent
    Generic map (
        --@26.6MHz
        BAUD_DIVIDE_G =>26,
        BAUD_RATE_G=>417
    )
    Port map (
        TXD =>link_Tx_Rx,                                -- Transmitted serial data
        RXD =>'0',                                     -- Received serial
        CLK =>ClockA,                                    -- Clock signal
        DBIN =>OutFromRom,                             -- Input parallel data to be transmitted
        DBOUT =>open,                                    -- Received parallel data output
        RDA      =>rdaTx,                               -- Read Data Avai
        TBE      =>tbeTx,                               -- Transfer Buffer Emty
        RD=>'1',                                      -- Read Strobe
        WR=>writeTx,                                 -- Write Strobe
        RST      =>ResetA);                           -- Reset signal

RomMem: MemR
port map(
    CLK      => ClockA,
    READ     => EN_RD,
    ADDR     => address,
    DATAOUT  => OutFromRom
);

end structural;

```

9.5.4 | Punto 2 - Nodo B

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NodoB is
    Port (
        ClockB : in STD_LOGIC:='0';
        ResetB : in STD_LOGIC:='0';
        Link_Rx_Tx : in STD_LOGIC;
        OkB: out std_logic;
        reqA: in std_logic
    );
end NodoB;

architecture Structural of NodoB is

component UARTcomponent is
    Generic (
        --@48MHz
        BAUD_DIVIDE_G : integer := 26;           --115200 baud
        --BAUD_RATE_G   : integer := 417

        --@26.6MHz
        BAUD_DIVIDE_G : integer := 14;           --115200 baud
        BAUD_RATE_G   : integer := 231
    );

```

```

Port (
    TXD      : out      std_logic      := '1';
    RXD      : in       std_logic;
    CLK      : in       std_logic;
    DBIN     : in       std_logic_vector(7 downto 0);
    DBOUT    : out      std_logic_vector(7 downto 0);
    RDA      : inout    std_logic;
    TBE      : out      std_logic      := '1';
    RD       : in       std_logic;
    WR       : in       std_logic;
    PE       : out      std_logic;
    FE       : out      std_logic;
    OE       : out      std_logic;
    RST      : in       std_logic      := '0');

end component;

component MemRw is
    generic(
        AddrLen: natural :=2;
        DataLen: natural :=7;
        NumberCells: natural :=7
    );
    port(
        CLK : in std_logic;
        RST : in std_logic;

        READ : in std_logic;
        WRITE : in std_logic;

        ADDR : in std_logic_vector(AddrLen downto 0);
        DATAIN : in std_logic_vector(DataLen downto 0);
        DATAOUT : out std_logic_vector(DataLen downto 0)
    );
end component;

component UC_B is
    Port (
        ClockB, ResetB: in std_logic;

        ----handshake
        OkB: out std_logic;
        ReqA: in std_logic;

        ----UART
        RDA: in std_logic:='0';
        EnRD: out std_logic:='0';
        OE, PE, FE: in std_logic;

        ----MEM
        WR_mem: out std_logic;
        OutCont: out std_logic_vector(2 downto 0) := (others => '0')
    );
end component;

```

```

signal OutFromRx:           std_logic_vector (7 DOWNTO 0);
signal rdaRx, tbeRx, rdRx:  std_logic := '0';
signal OE, PE, FE:          std_logic := '0';
signal WR_mem:              std_logic;
signal address:             std_logic_vector( 2 downto 0);

begin

UC: UC_B
port map(
    ClockB => ClockB,
    ResetB => ResetB,
    ----handshake
    OkB => OkB,
    ReqA => ReqA,
    ----UART
    RDA => rdaRx,
    EnRD => rdRx,
    OE => OE,
    PE => PE,
    FE => FE,
    ---MEM
    WR_mem => WR_mem,
    OutCont => address
);

Rx: UARTcomponent
Generic map (
    --@4MHz
    --                               BAUD_DIVIDE_G : integer := 26;           --115200 baud
    --                               BAUD_RATE_G   : integer := 417
    --@26.6MHz
    BAUD_DIVIDE_G =>26,
    BAUD_RATE_G=>417
)
Port map(
    TXD      => open,                                -- Transmitted serial data output
    RXD      => Link_Rx_Tx,                          -- 
    CLK       => ClockB,                            -- Clock
    DBIN     => "00000000",                         -- Input parallel data to be transmitted
    DBOUT    => OutFromRx,                           -- Received parallel data output
    RDA      => rdaRx,                             -- Read Data
    TBE      => open,                               -- Transfer Buffer Emty
    RD       => rdRx,                               -- Read Strobo
    WR       => '0',                                -- Write
    RST      => ResetB,                            -- Reset signal
    PE       => PE,                                 -- Parity enable
    FE       => FE,                                 -- 
    OE       => OE                                  -- 
);

```

```

MEM: MemRw
port map(
    CLK      => ClockB,
    RST      => ResetB,
    READ     => '0',
    WRITE    => WR_mem,
    ADDR     => address,
    DATAIN   => OutFromRx,
    DATAOUT  => open
);

```

9.6 | Punto 2 - UC A

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity UC_A is
    Port (
        ClockA, ResetA: in std_logic;
        Start: in std_logic:='0';

        ----handshake
        OkB: in std_logic;
        ReqA: out std_logic;

        ----UART
        Tbe: in std_logic:='0';
        EnWr: out std_logic:='0';

        ---MEM
        RD_mem: out std_logic;
        OutCont: out std_logic_vector(2 downto 0) := (others => '0')
    );
end UC_A;

architecture Behavioral of UC_A is

type stato is (idle, q0, q1, q2, q3);
signal stato_corrente : stato := idle;
signal stato_next: stato;

---signale contatori
signal cnt_mem: std_logic;
signal ctr_mem: std_logic_vector(2 downto 0) := "000";

begin

---EVOLUZIONE
process(ClockA, ResetA)
begin

```

```

if (ClockA'event and ClockA='1') then
  if ResetA = '1' then
    stato_corrente <= idle;
  else
    stato_corrente <= stato_next;
  end if;
end if;

case stato_corrente is
when idle =>
  if Start = '1' then
    stato_next <= q0;
  elsif Start = '0' then
    stato_next <= idle;
  end if;

  RD_mem      <= '0';
  ReqA        <= '0';
  cnt_mem     <= '0';

when q0 =>
  if OkB = '1' then
    stato_next <= q0;
  elsif OkB = '0' then
    stato_next <= q1;
  end if;
when q1 =>
  if OkB = '0' then
    stato_next <= q1;
  elsif OkB = '1' then
    stato_next <= q2;
  end if;
  RD_mem      <= '1';
  ReqA        <= '1';
when q2 =>
  stato_next <= q3;
  ---abilito trasmissione
  EnWr        <= '1';
when q3 =>
  if Tbe = '0' and OkB = '0' then
    --aspetto fine trasmissione
    stato_next <= q3;
  end if;

  if OkB = '1' then
    --effettuo ritrasmissione
    stato_next <= q2;
  end if;

  if Tbe = '1' and OkB = '0' then
    -- trasmissione completata
    stato_next <= idle;
    cnt_mem <= '1';
    ReqA    <= '0';
  end if;

  ---disabilito strobe
  EnWr        <= '0';

```

```

        RD_mem  <=  '0';
    end case;
end process;

---CONTATORE MEM
process(ClockA)
begin
    if (ClockA'event and ClockA='1') then
        if ResetA = '1' then
            ctr_mem <= (others => '0');
        elsif cnt_mem = '1' then
            ctr_mem <= ctr_mem + 1;
        else
            ctr_mem <= ctr_mem;
        end if;
    end if;

    OutCont  <= ctr_mem;
end process;

end Behavioral;

```

9.7 | Punto 2 - UC B

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity UC_B is
    Port (
        ClockB, ResetB: in std_logic;

        ----handshake
        OkB: out std_logic;
        ReqA: in std_logic;

        ----UART
        RDA: in std_logic:='0';
        EnRD: out std_logic:='0';
        OE, PE, FE: in std_logic;

        ---MEM
        WR_mem: out std_logic;
        OutCont: out std_logic_vector(2 downto 0) := (others => '0')
    );
end UC_B;

architecture Behavioral of UC_B is

type stato is (idle, q0, q1, q2, q3);
signal stato_corrente : stato := idle;
signal stato_next: stato;

---signale contatori
signal cnt_mem: std_logic;
signal ctr_mem: std_logic_vector(2 downto 0) := "000";

```

```

begin

---EVOLUZIONE
process(ClockB, ResetB)
begin
    if (ClockB'event and ClockB='1') then
        if ResetB = '1' then
            stato_corrente <= idle;
        else
            stato_corrente <= stato_next;
        end if;
    end if;

    case stato_corrente is
    when idle =>
        if ReqA = '1' then
            stato_next <= q0;
        elsif ReqA = '0' then
            stato_next <= idle;
        end if;
        EnRD      <= '0';
        WR_mem    <= '0';
        cnt_mem   <= '0';
        OkB       <= '0';
    when q0 =>
        stato_next <= q1;
        OkB       <= '1';
    when q1 =>
        if RDA = '1' then
            if (OE = '1' or PE = '1' or FE = '1') then
                stato_next <= q0;
            else
                stato_next <= q2;
            end if;
        else
            OkB <= '0';
            stato_next <= q1;
        end if;
    when q2 =>
        OkB <= '0';
        EnRD      <= '1';
        WR_mem    <= '1';
        stato_next <= q3;
    when q3 =>
        if ReqA = '0' then
            cnt_mem   <= '1';
            stato_next <= idle;
        elsif ReqA = '1' then
            stato_next <= q3;
        end if;
        EnRD      <= '0';
        WR_mem    <= '0';
        OkB       <= '0';
    end case;
end process;

---CONTATORE MEM
process(ClockB)

```

```
begin
  if (ClockB'event and ClockB='1') then
    if ResetB = '1' then
      ctr_mem <= (others => '0');
    elsif cnt_mem = '1' then
      ctr_mem <= ctr_mem + 1;
    else
      ctr_mem <= ctr_mem;
    end if;
  end if;

  OutCont <= ctr_mem;
end process;

end Behavioral;
```

10 | Switch Multistadio

Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network, lo switch progettato deve operare come segue:

- Lo switch deve consentire lo scambio di messaggi di 2 bit ciascuno da un nodo sorgente a un nodo destinazione in un rete con 4 nodi, implementando uno schema a priorità fissa fra i nodi (ed. nodo 1 più prioritario, con priorità decrescenti fino al nodo 4).
- Opzionale, rimuovendo l'ipotesi di lavorare secondo uno schema a priorità fra i nodi e considerando una rete di 8 nodi, lo switch deve gestire eventuali conflitti generati da collisioni secondo un meccanismo a scelta (ad es. perdendo uno dei messaggi in conflitto).
- Opzionale, si implementi un protocollo di handshaking semplice regolato da una coppia di segnali (pronto a inviare/pronto a ricevere) per l'invio di ciascun messaggio fra due nodi.

10.1 | Descrizione Soluzione

10.1.1 | Introduzione

Immaginiamo uno switch a soli due ingressi e due uscite, così semplice perché sarà facile da programmare e perché in realtà in questo modo non dovrà implementare un algoritmo di instradamento, ma si limiterà solo a prendere una decisione sulla uscita su o giù che richiede solamente un bit.

Supponendo N nodi distinti e utilizzando dispositivi con solo due ingressi e due uscite, saranno sufficienti $\log_2 n$ stadi per risolvere il problema della connettività, ad esempio con 8 nodi bastano 3 stadi per garantire la connettività totale del sistema.

In questa architettura per gestire l'indirizzamento l'indirizzo viene disposto in k bit e deve essere diviso in tante parti quanti sono gli stadi, in modo da comandare opportunamente i blocchi.

Così facendo la logica di controllo è distribuita negli stadi, quindi sarà il nodo a cui arriva il messaggio ad analizzare l'indirizzo e comportarsi di conseguenza.

Ad esempio quando abbiamo 8 nodi e quindi 3 bit per l'indirizzo, se vogliamo raggiungere il nodo 6 con l'indirizzo 110, allora si può utilizzare il primo bit dell'indirizzo per pilotare il primo stadio, il secondo per il secondo stadio e il terzo per il terzo stadio.

Ovviamente questa è una condizione abbastanza forte dal punto di vista dell'architettura, che è possibile realizzare soltanto utilizzando particolari proprietà topologiche della rete, un esempio è rappresentato dalla **"omega network"**, la quale si basa sul concetto del perfect shuffling.

10.1.2 | Perfect Shuffling

Tale tecnica fa riferimento al modo con cui le carte di un mazzo vengono meschiate con l'intento di tornare all'ordine originale.

Meschiare perfettamente significa dividere un mazzo in due metà e la prima carta della prima metà del mazzo viene accoppiata con la prima carta della seconda metà, la seconda carta della prima metà viene accoppiata con la seconda carta della seconda metà e così via.

Si può dimostrare che un mazzo di carte ordinato dopo $\log_2 n$ volte questa operazione ritorna all'ordine iniziale, dove n è il numero di carte.

Volendo trasportare quest'idea alla creazione di collegamenti tra 8 nodi (numerati da 0 a 7), avremo al primo stadio 4 switch con due ingressi che ricevono i nodi corrispondenti alle combinazioni ottenute dopo il primo passaggio dello shuffling, infatti avremo le coppie 0-4, 1-5, 2-6 e 3-7.

Al secondo stadio ripetiamo la stessa operazione e avremo i collegamenti 0-2, 4-6, 1-3 e 5-7, infine al terzo stadio otteniamo in uscita l'ordinamento iniziale.

Va notato la flessibilità dall'architettura che grazie alla forte modularità dei componenti rende lo switch molto scalabile, infatti è facile realizzarne uno di grandi dimensioni, mentre non è possibile per gli switch diretti.

10.1.3 | Punto 1 - Sistema Complessivo

Anche in questo caso abbiamo decomposto la macchina in unità operativa e unità di controllo.

Unità Operativa realizza la rete di switch secondo il modello Omega Network.

Lo switch è realizzato tramite la composizione di un multiplexer 2:1 e un demultiplexer 1:2, prende in ingresso un bit di indirizzo sorgente e un bit di indirizzo destinazione, che rappresentano rispettivamente l'ingresso di selezione al multiplexer e al demultiplexer.

Nel complesso abbiamo due switch nel primo stadio e due switch nel secondo stadio, ciascuno dei quali produce chiaramente due uscite che sono state collegate tra di esse in modo da rispettare il perfect shuffling come è possibile vedere in figura 10.1.

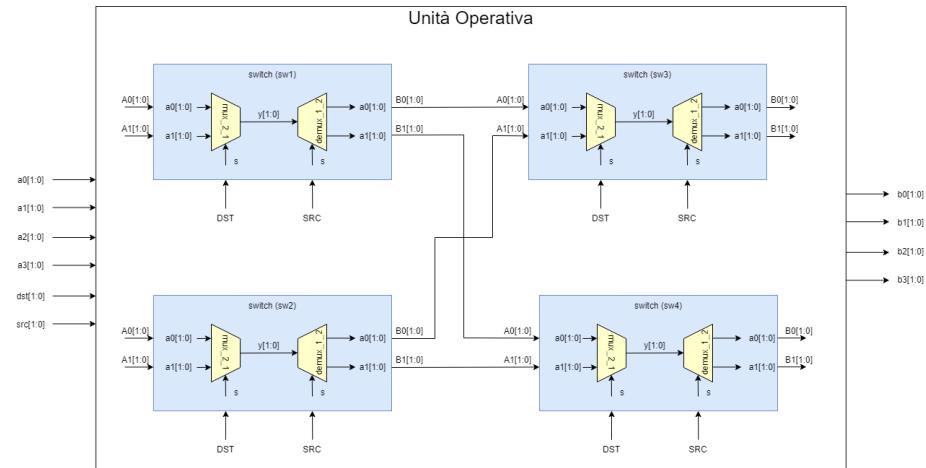


Figura 10.1: Schema del UO.

Unità Controllo Quest'unità, rappresentata nella figura 10.2, permette a ciascun nodo di comunicare la sua eventuale volontà di trasmettere, tali input sono i segnali in base ai quali l'arbitro di volta in volta decide, in base alla maggiore priorità, quale nodo può trasmettere.

Inoltre mentre nell'unità operativa viaggiano solo i bit di dato, nell'unità di controllo negli ingressi x_i riceve da ciascun nodo che vuole trasmettere una stringa di 4 bit:

- 2 bit di indirizzo destinazione;
- 2 bit di informazione effettiva.

Produce due segnali che rappresentano il nodo che sta comunicando e il destinatario del messaggio, sono indicati in output come S e D.

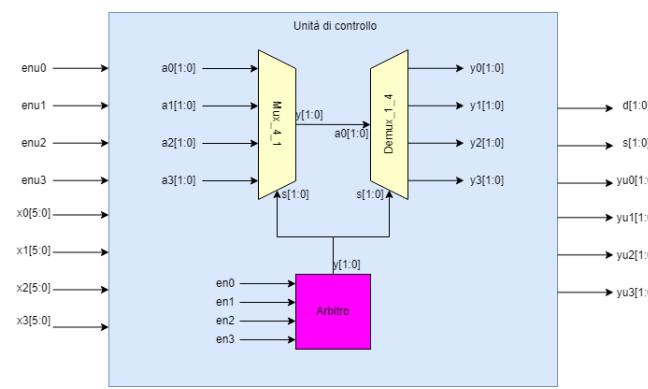


Figura 10.2: Schema del UC.

10.1.4 | Punto 2 - Variante Sistema Complessivo

In questo caso abbiamo una rete complessiva composta esclusivamente da 8 switch, senza più fare distinzione tra parte di controllo e parte operativa, la strategia adottata è quella di far perdere un messaggio nel caso di una collisione.

La rete complessiva prende in ingresso 8 segnali, che vengono mappati opportunamente con gli ingressi dei vari switch e ciò è chiaramente visibile in figura 10.3, inoltre due segnali src e dst sono utilizzati come selezioni rispettivamente dei multiplexer 2:1 e dei demultiplexer 1:2.

In particolare: i 4 switch del primo stadio prendono come selezioni il bit più significativo di src e dst, mentre i 4 switch del secondo stadio prendono come selezioni il bit meno significativo.

Il primo switch, quello in alto a sinistra nell'unità operativa, manda un'uscita nello switch 5 e l'altra nello switch 6, in modo analogo fa il secondo switch.

Il terzo switch invece manda un'uscita nello switch 7 e l'altra nello switch 8, e lo stesso fa il quarto switch. Infine, gli switch del secondo stadio mandano le proprie uscite all'esterno della rete complessiva.

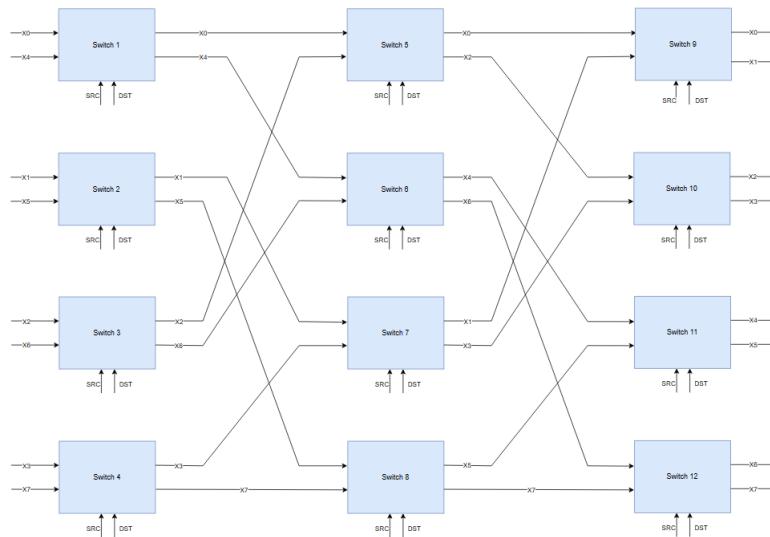


Figura 10.3: Struttura omega network al punto 2.

10.2 | Testbench

10.2.1 | Test sistema 1

In questo caso è stato definito un semplice testbench che parte con tutti i segnali di abilitazione alti e tutti i nodi che forniscono un messaggio, durante lo scandire del tempo le varie abilitazioni si abbassano per controllare che l'arbitro funzioni bene, si può notare un comportamento come sperato nella figura 10.4.

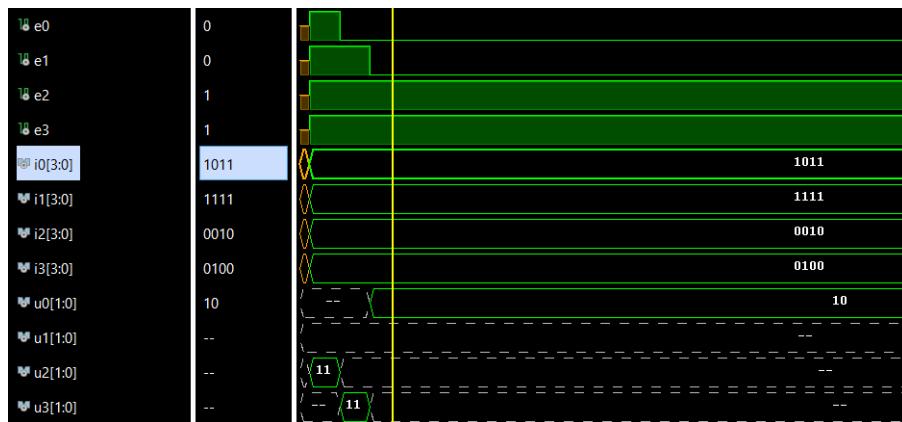


Figura 10.4: Testbench switch al punto 1.

10.2.2 | Test sistema 2

In questo caso facciamo un test per vedere se effettivamente viene perso un ingresso: dando in ingresso sia il segnale xi_1 che xi_3 avranno collisione nel settimo switch, la collisione è risolta dal fatto che il secondo bit del segnale di src seleziona l'uscita di xi_1 in quello switch.

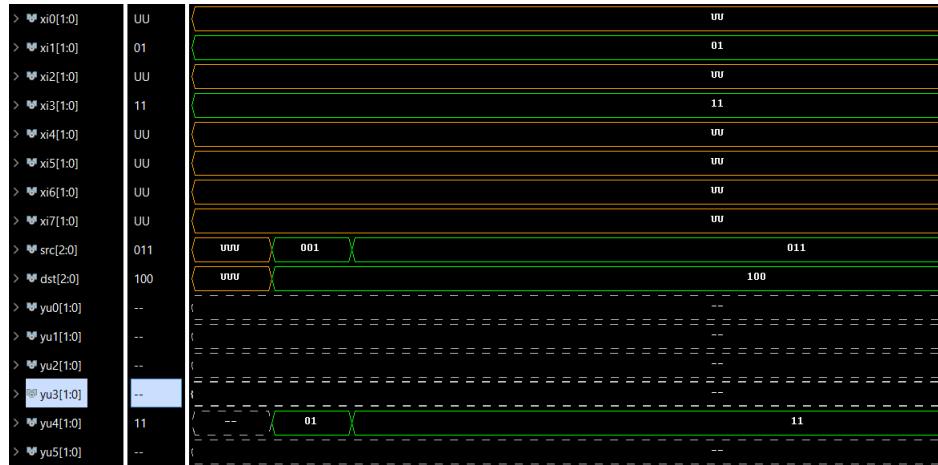


Figura 10.5: Testbench switch al punto 2.

10.3 | Codice

10.3.1 | Parte 1 - Switch Elementare

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

entity switch is
    generic (
        N: natural := 2 --dimensione pacchetto
    );
    Port (
        A0, A1: in std_logic_vector(N-1 downto 0);
        B0, B1: out std_logic_vector(N-1 downto 0);
        SRC: in std_logic;
        DST: in std_logic
    );
end switch;
architecture structural of switch is
    component mux_21_gen is
        generic(
            N: natural := 2
        );
        port(
            a0, a1: in std_logic_vector(N-1 downto 0);
            s: in std_logic;
            Y: out std_logic_vector(N-1 downto 0)
        );
    end component;
    component demux_21_gen is
        generic(
            N: natural := 2
        );
        port(
            y: in std_logic_vector( N-1 downto 0);

```

```

        s: in std_logic;
        a0, a1: out std_logic_vector(N-1 downto 0)
    );
end component;

signal k0: std_logic_vector (N-1 downto 0);

begin
    mux: mux_21_gen generic map(N)
        port map(
            a0 => A0,
            a1 => A1,
            s => SRC,
            y => k0
        );
    demux: demux_21_gen generic map(N)
        port map(
            y => k0,
            s => DST,
            a0 => B0,
            a1 => B1
        );
end structural;

```

10.3.2 | Parte 1 - Arbitro

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Arbitro is
    Port (
        en0, en1, en2, en3: in std_logic;
        y: out std_logic_vector(1 downto 0);
        src: out std_logic_vector(1 downto 0)
    );
end Arbitro;

architecture Behavioral of Arbitro is
begin
    y <= "00" when en0 = '1' else
        "01" when en1 = '1' else
        "10" when en2 = '1' else
        "11" when en3 = '1' else
        "--";

    src <= "00" when en0 = '1' else
        "01" when en1 = '1' else
        "10" when en2 = '1' else
        "11" when en3 = '1' else
        "--";

end Behavioral;

```

10.3.3 | Parte 1 - Unità Controllo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```

entity UnitContr is
    generic(
        N: natural:= 4; --dimensione dati
        payload: natural:= 2;
        M: natural:= 2 -- dimensione indirizzi
    );
    port(
        x0, x1, x2, x3: in std_logic_vector(N-1 downto 0);
        enu0, enu1, enu2, enu3: in std_logic;
        s, d: out std_logic_vector(M-1 downto 0);
        yu0, yu1, yu2, yu3: out std_logic_vector(payload-1 downto 0)
    );
end UnitContr;

architecture structural of UnitContr is

component Arbitro is
    port(
        en0, en1, en2, en3: in std_logic;
        y: out std_logic_vector(1 downto 0);
        src: out std_logic_vector(1 downto 0)
    );
end component;
component mux_41_gen is
    generic(
        N: natural:= 2;
        M: natural:= 2
    );
    port(
        a0, a1, a2, a3: in std_logic_vector(N-1 downto 0);
        s: in std_logic_vector(M-1 downto 0);
        y: out std_logic_vector(N-1 downto 0)
    );
end component;
component demux_41_gen is
    generic(
        N: natural:= 6;
        M: natural:= 2
    );
    port(
        a0: in std_logic_vector(N-1 downto 0);
        s: in std_logic_vector(M-1 downto 0);
        y0, y1, y2, y3: out std_logic_vector(N-1 downto 0)
    );
end component;

signal k: std_logic_vector(M-1 downto 0);
signal p: std_logic_vector(payload-1 downto 0);

begin

    mux: mux_41_gen
        generic map(payload, M)
        port map(
            a0 => x0(1 downto 0),

```

```

    a1 => x1(1 downto 0),
    a2 => x2(1 downto 0),
    a3 => x3(1 downto 0),
    s  => k,
    y  => p
);

demux: demux_41_gen
generic map(payload, M)
port map(
    a0 => p,
    s  => k,
    y0 => yu0,
    y1 => yu1,
    y2 => yu2,
    y3 => yu3
);

arb: Arbitro
port map(
    en0 => enu0,
    en1 => enu1,
    en2 => enu2,
    en3 => enu3,
    src => s,
    y => k
);

    d <= x0 (3 downto 2) when k ="00" else
        x1 (3 downto 2) when k ="01" else
        x2 (3 downto 2) when k ="10" else
        x3 (3 downto 2) when k ="11" else
        "--";
end structural;

```

10.3.4 | Parte 1 - Unità Operativa

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity UnitOp is
    generic(
        N: natural:= 2; --dimensione pacchetto
        M: natural:= 2 --dimensione indirizzamento
    );
    Port (
        a0, a1, a2, a3 : in STD_LOGIC_VECTOR (N-1 downto 0);
        src: in std_logic_vector(M-1 downto 0);
        dst: in std_logic_vector(M-1 downto 0);
        b0, b1, b2, b3 : out std_logic_vector(N-1 downto 0)
    );
end UnitOp;

architecture structural of UnitOp is
    component switch is
        generic(
            N: natural:= 2
        );

```

```

    port(
      A0, A1: in std_logic_vector(N-1 downto 0);
      B0, B1: out std_logic_vector(N-1 downto 0);
      SRC: in std_logic;
      DST: in std_logic
    );
  end component;
  signal k0, k1, k2, k3: std_logic_vector(N-1 downto 0);

begin
  sw1: switch generic map(N)
    port map(
      A0 => a0,
      A1 => a1,
      B0 => k0,
      B1 => k1,
      SRC => src(0),
      DST => dst(1)
    );
  sw2: switch generic map(N)
    port map(
      A0 => a2,
      A1 => a3,
      B0 => k2,
      B1 => k3,
      SRC => src(0),
      DST => dst(1)
    );
  sw3: switch generic map(N)
    port map(
      A0 => k0,
      A1 => k2,
      B0 => b0,
      B1 => b1,
      SRC => src(1),
      DST => dst(0)
    );
  sw4: switch generic map(N)
    port map(
      A0 => k1,
      A1 => k3,
      B0 => b2,
      B1 => b3,
      SRC => src(1),
      DST => dst(0)
    );
end structural;

```

10.3.5 | Parte 1 - Sistema Complessivo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity OmegaNetwork is
  generic(
    N: natural:= 6;
    payload: natural := 2;
    M: natural:= 2
  );

```

```

Port (
    e0, e1, e2, e3: in std_logic;
    i0, i1, i2, i3: in std_logic_vector(N-1 downto 0);
    u0, u1, u2, u3: out std_logic_vector(payload-1 downto 0)
);
end OmegaNetwork;

architecture Structural of OmegaNetwork is

component UnitOP is
    generic(
        N: natural:= 2;
        M: natural:= 2
    );
    Port (
        a0, a1, a2, a3 : in STD_LOGIC_VECTOR (N-1 downto 0);
        src: in std_logic_vector(M-1 downto 0);
        dst: in std_logic_vector(M-1 downto 0);
        b0, b1, b2, b3 : out std_logic_vector(N-1 downto 0)
    );
end component;

component UnitContr is
    generic(
        N: natural:= 6;
        payload: natural:= 2;
        M: natural:= 2
    );
    port(
        x0, x1, x2, x3: in std_logic_vector(N-1 downto 0);
        enu0, enu1, enu2, enu3: in std_logic;
        s, d: out std_logic_vector(M-1 downto 0);
        yu0, yu1, yu2, yu3: out std_logic_vector(payload-1 downto 0)
    );
end component;
signal k0, k1, k2, k3: std_logic_vector(payload-1 downto 0);
signal SRC, DST: std_logic_vector(M-1 downto 0);

begin

unit_controllo: UnitContr
generic map(4, 2, 2)
port map(
    x0 => i0,
    x1 => i1,
    x2 => i2,
    x3 => i3,
    enu0 => e0,
    enu1 => e1,
    enu2 => e2,
    enu3 => e3,
    yu0 => k0,
    yu1 => k1,
    yu2 => k2,
    yu3 => k3,
    s => SRC,
    d => DST
);

```

```

unit_operativa: UnitOP
generic map(2, 2)
port map(
    a0 => k0,
    a1 => k1,
    a2 => k2,
    a3 => k3,
    src => SRC,
    dst => DST,
    b0 => u0,
    b1 => u1,
    b2 => u2,
    b3 => u3
);
end Structural;

```

10.3.6 | Parte 2 - Variante Sistema Complessivo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity OmegaNetwork_ver2 is
    generic (
        N: natural:= 2
    );
    Port (
        xi0, xi1, xi2, xi3, xi4, xi5, xi6, xi7: in std_logic_vector(N-1 downto 0);
        src, dst: in std_logic_vector(N downto 0);
        yu0, yu1, yu2, yu3, yu4, yu5, yu6, yu7: out std_logic_vector(N-1 downto 0)
    );
end OmegaNetwork_ver2;

architecture structural of OmegaNetwork_ver2 is

    component switch is
        generic(
            N: natural:= 2
        );
        port(
            A0, A1: in std_logic_vector(N-1 downto 0);
            B0, B1: out std_logic_vector(N-1 downto 0);
            SRC: in std_logic;
            DST: in std_logic
        );
    end component;

    signal k0, k1, k2, k3, k4, k5, k6, k7: std_logic_vector(N-1 downto 0);
    signal z0, z1, z2, z3, z4, z5, z6, z7: std_logic_vector(N-1 downto 0);

begin

    ---primo stadio
    sw1: switch generic map(N)
        port map(
            A0 => xi0,
            A1 => xi4,
            B0 => k0,
            B1 => k1,

```

```

        SRC => src(2),
        DST => dst(2)
    );
sw2: switch generic map(N)
    port map(
        A0 => xi1,
        A1 => xi5,
        B0 => k2,
        B1 => k3,
        SRC => src(2),
        DST => dst(2)
    );
sw3: switch generic map(N)
    port map(
        A0 => xi2,
        A1 => xi6,
        B0 => k4,
        B1 => k5,
        SRC => src(2),
        DST => dst(2)
    );
sw4: switch generic map(N)
    port map(
        A0 => xi3,
        A1 => xi7,
        B0 => k6,
        B1 => k7,
        SRC => src(2),
        DST => dst(2)
    );

```

---secondo stadio

```

sw5: switch generic map(N)
    port map(
        A0 => k0,
        A1 => k4,
        B0 => z0,
        B1 => z1,
        SRC => src(1),
        DST => dst(1)
    );
sw6: switch generic map(N)
    port map(
        A0 => k1,
        A1 => k5,
        B0 => z2,
        B1 => z3,
        SRC => src(1),
        DST => dst(1)
    );
sw7: switch generic map(N)
    port map(
        A0 => k2,
        A1 => k6,
        B0 => z4,
        B1 => z5,
        SRC => src(1),

```

```
DST => dst(1)
);
sw8: switch generic map(N
    port map(
        A0 => k3,
        A1 => k7,
        B0 => z6,
        B1 => z7,
        SRC => src(1),
        DST => dst(1)
);
--terzo stadio

sw9: switch generic map(N
    port map(
        A0 => z0,
        A1 => z4,
        B0 => yu0,
        B1 => yu1,
        SRC => src(0),
        DST => dst(0)
);
sw10: switch generic map(N
    port map(
        A0 => z1,
        A1 => z5,
        B0 => yu2,
        B1 => yu3,
        SRC => src(0),
        DST => dst(0)
);
sw11: switch generic map(N
    port map(
        A0 => z2,
        A1 => z6,
        B0 => yu4,
        B1 => yu5,
        SRC => src(0),
        DST => dst(0)
);
sw12: switch generic map(N
    port map(
        A0 => z3,
        A1 => z7,
        B0 => yu6,
        B1 => yu7,
        SRC => src(0),
        DST => dst(0)
);
end structural;
```

11 | Macchina Aritmetica

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

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

Opzionalmente, la macchina implementata può essere sintetizzata su FPGA e testata mediante l'utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di sviluppo in dotazione al gruppo, la modalità di utilizzo degli stessi è a completa discrezione degli studenti.

11.1 | Descrizione Soluzione

Per la soluzione del problema abbiamo scelto di implementare il moltiplicatore di Robertson.

Questo moltiplicatore è un moltiplicatore sequenziale ed è capace di fare calcoli con valori col segno, iniziamo spiegando brevemente come funziona per valori unsigned.

Prima di tutto quando moltiplichiamo due numeri X e Y unsigned con N bit otteniamo una stringa di $2N$ bit, quindi prima di tutto partiamo con una stringa di tutti zeri di tale lunghezza, che chiamiamo somma parziale.

L'algoritmo andrà ad aggiungere alla somma parziale Y o una stringa di zero in base al valore della i-esima cifra di X, dopodiché effettuerà uno shift a destra e la cifra che resterà sarà la nuova somma parziale, in figura 11.1 le somme parziali sono indicate con P e in rosso ci sono le cifre sommate ad ogni passo.

Alla fine del algoritmo avremo due stringhe di N bit, una delle due, detta A, è il risultato dell'ultima somma parziale e l'altra, detta Q, è composta dai vari elementi shiftati a sinistra.

Y 1010	
X 1101	
0000 0000	P0
1010	
0000 1010	ADD;shift
000 0101	0 P1
0000	
000 0101	0 ADD;shift
00 0010	10 P2
1010	
00 1100	10 ADD;shift
0 0110	010 P3
1010	
1 0000	010 ADD;shift
1000	0010
	A Q

Figura 11.1: Esempio di moltiplicazione con Robertson

Questa macchina ha un datapath, con qualche accorgimento, è formato da un registro per contenere il moltiplicando, poi avremo un sommatore, un multiplexer che seleziona lo 0 o il moltiplicando, uno shift-register che inizialmente contiene il moltiplicatore nelle N cifre meno significative, alla fine delle somme parziali avrà al suo interno il risultato.

11.1.1 | Cifre negative

Il problema dell'algoritmo, proposto fin'ora, è che funziona solo per i numeri positivi.

Quando abbiamo dei numeri negativi in complementi a due, ogni cifra del numero ha un peso dato dalla notazione posizionale, ma la cifra più significativa ha un peso particolare.

Infatti se abbiamo n bit la cifra più significativa è quella moltiplicata per 2^{n-1} , ma se il numero è negativo il peso della cifra significativa è negativo, cioè $-1 \cdot 2^{n-1}$, questo influisce perché l'algoritmo scandisce una cifra per volta ma non possiamo trattare una cifra che ha peso negativo come una cifra di peso positivo. Quindi distinguiamo 4 casi diversi con delle apposite correzioni per adattare l'algoritmo:

1. $X > 0, Y > 0$ - Facciamo un'operazione tra unsigned usando i passi ADD e Shift.
2. $X > 0, Y < 0$ - Ogni volta che moltiplichiamo Y per una cifra $X_i = 1$ la somma parziale è negativa, quindi dopo lo shift poniamo la cifra più significativa di A ad 1.
3. $X < 0, Y > 0$ - La somma parziale per i primi $n-1$ termini è positiva, l'ultima somma andremo a fare un passo correttivo dove effettuiamo una sottrazione invece di una somma.
4. $X < 0, Y < 0$ - In questo caso seguiamo un procedimento analogo al caso 2, ma l' n -esimo prodotto sarà ottenuto da un passo correttivo con la sottrazione con nel caso 3.

Nel caso 2, ci accorgiamo subito che il valore del moltiplicando è negativo e quindi settiamo un latch F a 1, da quel momento in poi nel fare lo shift mettiamo in testa al registro A un 1.

11.1.2 | Schema della macchina

La macchina è formata da una parte di controllo e una parte operativa.

La macchina di controllo scandisce le operazioni da fare bit a bit in base alla definizione del algoritmo, nella figura 11.2 è stato definito l'automa dell'unità.

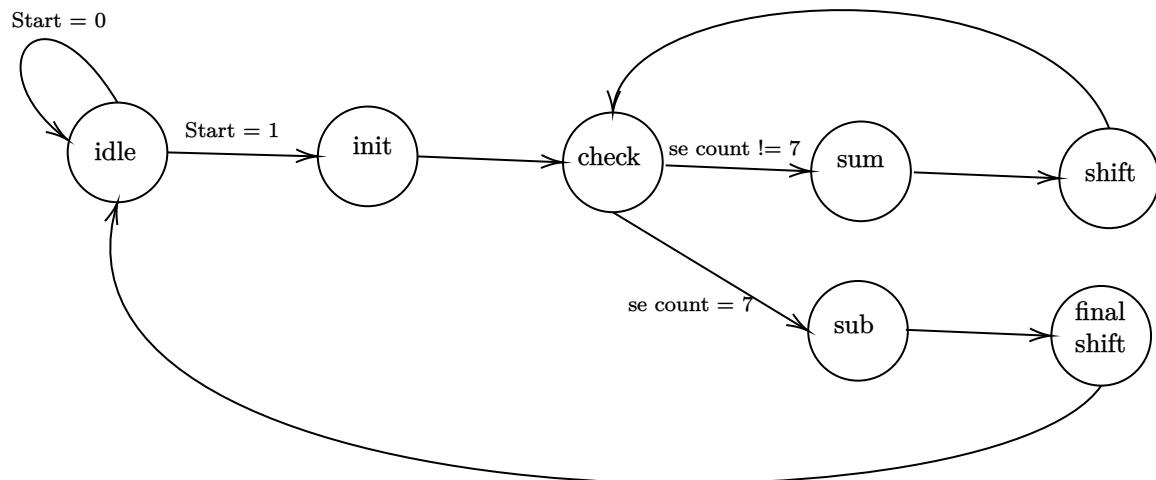


Figura 11.2: Automa unità di controllo.

Lo schema del moltiplicatore come possiamo notare dalla figura 11.3 è abbastanza articolato. Innanzitutto abbiamo dall'esterno dei segnali di start, reset e clock, il primo segnale viene mandato in ingresso all'unità di controllo e serve per avviare l'automa di tale unità, mentre gli altri due segnali servono rispettivamente per resettare la macchina e per la temporizzazione.

Abbiamo poi i due ingressi di 8 bit A e B, dove il primo entra direttamente in ingresso nello shift register, mentre l'altro entra in un registro, il quale memorizza al proprio interno il valore e lo manda poi in uscita ad un multiplexer 2:1.

Questo multiplexer ha due possibili uscite: il valore B e una stringa di 8 zeri, selezionate in base al segnale fornito dall'unità di controllo; l'uscita viene poi mandata in ingresso ad una porta XOR insieme ad un segnale di subtract, anch'esso fornito dall'unità di controllo, e l'uscita di questa porta entra poi in un ADDER.

Infine notiamo che dallo shift register esce un segnale RIS che sono i 16 bit che rappresentano il risultato della moltiplicazione.

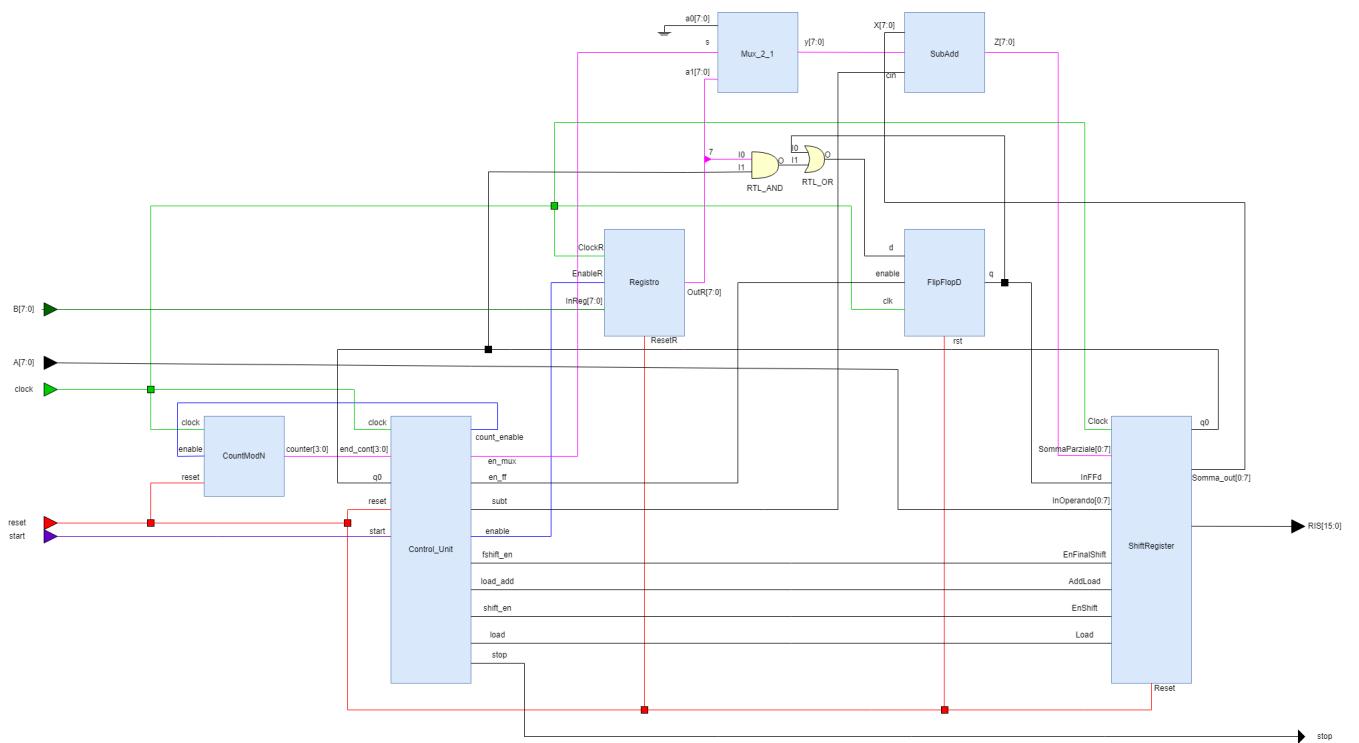


Figura 11.3: Struttura complessiva del moltiplicatore di Robertson.

11.2 | Testbench

Sono stati realizzati 4 test, uno per ogni possibile combinazione di X e Y.

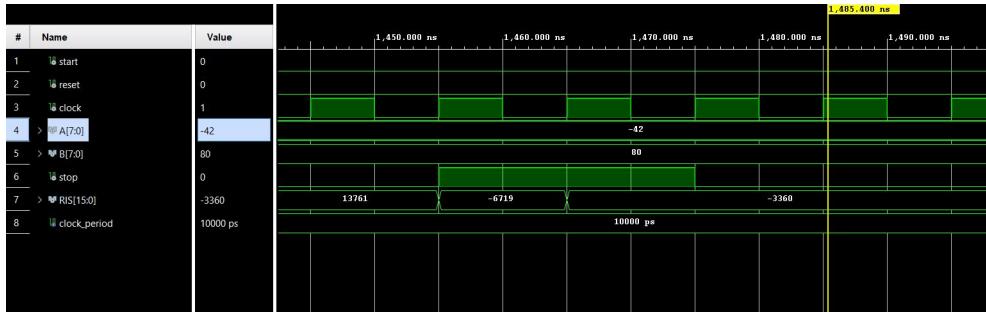


Figura 11.4: Test per valori discordi caso 1.

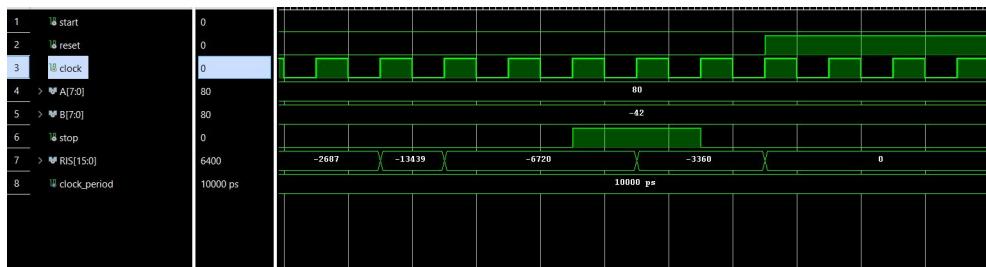
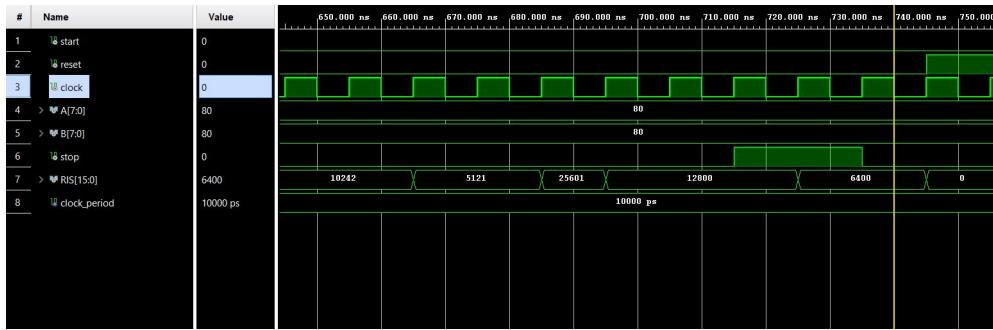
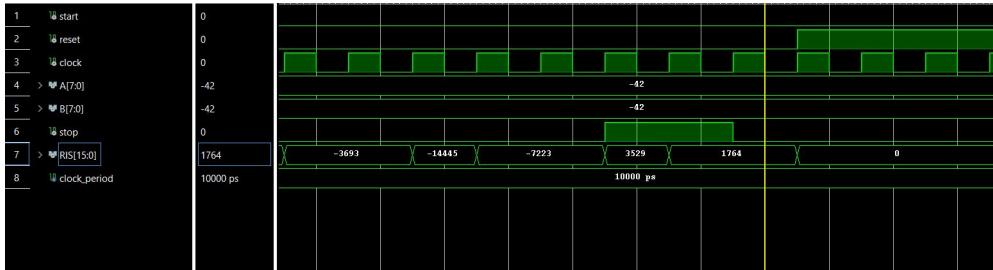
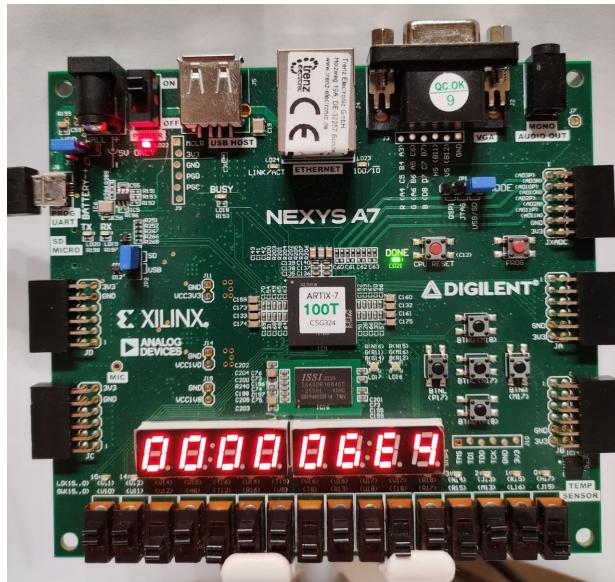


Figura 11.5: Test per valori discordi caso 2.

**Figura 11.6:** Test per valori entrambi positivi.**Figura 11.7:** Test per valori entrambi negativi.

11.3 | Sintesi

**Figura 11.8:** Prodotto di due numeri negativi su board.

11.4 | Codice VHDL

Svariati componenti sono stati realizzati in precedenza per altri esercizi, quindi vengono riportati solo quelli realizzati per questo esercizio.

11.4.1 | Sistema Complessivo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Rete_Complessiva is
    generic (N: natural := 8);
    port(
        start, reset, clock: in std_logic;
        A,B: in std_logic_vector(7 downto 0);
        stop: out std_logic;
        RIS: out std_logic_vector(15 downto 0)
    );
end Rete_Complessiva;

architecture Behavioral of Rete_Complessiva is

component Registro is
    generic (N : natural := 8);
    port(
        InReg: in std_logic_vector(N-1 downto 0);
        ClockR, ResetR, EnableR: in std_logic;
        OutR: out std_logic_vector(N-1 downto 0)
    );
end component;

component FlipFlopD is
    port (
        clk, rst, enable, d: in std_logic;
        q: out std_logic
    );
end component;

component Mux_2_1 is
    generic (N : natural := 8);
    port(
        a0, a1 : in std_logic_vector(N-1 downto 0);
        s : in STD_LOGIC;
        y : out std_logic_vector(N-1 downto 0)
    );
end component;

component SubAdd is
    generic (
        N: natural :=8
    );
    port(
        X,Y:in std_logic_vector(N-1 downto 0);
        cin:in std_logic;
        Z:out std_logic_vector(N-1 downto 0);
        cout:out std_logic
    );

```

```

end component;

component ShiftRegister is
    generic (N : natural := 8);
    port(
        InOperando : in std_logic_vector (0 to N-1);
        InFFd: in std_logic;
        Clock, Reset, Load, AddLoad, EnShift, EnFinalShift: in std_logic;
        SommaParziale : in std_logic_vector (0 to N-1);

        Somma_Out: out std_logic_vector (0 to N-1);
        q0 : out std_logic;
        Q : out std_logic_vector (0 to 2*N-1)
    );
end component;

component ContModN is
    generic (N: natural := 4);
    port(
        clock : in STD_LOGIC;
        reset : in STD_LOGIC;
        enable : in STD_LOGIC; --questo Ã  l'enable del clock, insieme danno l'impulso di
        counter : out STD_LOGIC_VECTOR (N-1 downto 0)
    );
end component;

component Control_Unit is
    port(
        clock: in std_logic;
        reset: in std_logic;
        start: in std_logic;
        q0: in std_logic;
        end_cont: in std_logic_vector(3 downto 0);
        count_enable: out std_logic;
        subt: out std_logic;
        en_mux: out std_logic;
        enable: out std_logic;
        en_ff : out std_logic;
        shift_en: out std_logic;
        fshift_en : out std_logic;
        load: out std_logic;
        load_add: out std_logic;
        stop: out std_logic
    );
end component;

signal e1,k0,k1,k2,k3 : std_logic_vector(7 downto 0);
signal t3: std_logic_vector(3 downto 0);
signal t1,t2,t4,t5,t6,t7,t8,t9,t10 : std_logic;
signal s1 : std_logic_vector(15 downto 0);
signal L, f_s, e_f: std_logic;
signal input_ff,f1: std_logic;

begin

R: Registro
    port map(

```

```

        InReg => B,
        ClockR => clock,
        ResetR => reset,
        EnableR => t8,
        OutR => k1
    );

MUX: mux_2_1
    generic map (N)
    port map(
        a0 => "00000000",
        a1 => k1,
        s => t7,
        y => k2
    );

input_ff <= (k1(7) and t2) or f1;

FF: FlipFlopD
    port map (
        clk => clock,
        rst => reset,
        d => input_ff,
        enable => e_f,
        q => f1
    );

RP: SubAdd
    generic map (N)
    port map(
        X => k3,
        Y => k2,
        cin => t10,
        Z => e1,
        cout => t1
    );

SR: ShiftRegister
    generic map (N)
    port map(
        InOperando => A,
        InFFd => f1,
        Clock => clock,
        Reset => reset,
        Load => t5,
        AddLoad => L,
        EnShift => t9,
        EnFinalShift => f_s,
        SommaParziale => e1,
        Somma_Out => k3,
        q0 => t2,
        Q => s1
    );

C: ContModN
    port map(
        clock => clock,
        reset => reset,

```

```

        enable => t6,
        counter => t3
    );

UC: Control_Unit
    port map(
        clock => clock,
        reset => reset,
        start => start,
        q0 => t2,
        end_cont => t3,
        subt => t10,
        count_enable => t6,
        en_mux => t7,
        enable => t8,
        en_ff => e_f,
        shift_en => t9,
        fshift_en => f_s,
        load => t5,
        load_add => L,
        stop => t4
    );
    stop <= t4;
    RIS <= s1;

end Behavioral;

```

11.4.2 | Unità di Controllo

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Control_Unit is
    port(
        clock: in std_logic;
        reset: in std_logic;
        start: in std_logic;
        q0: in std_logic;
        end_cont: in std_logic_vector(3 downto 0);
        count_enable: out std_logic;
        subt: out std_logic;
        en_mux: out std_logic;
        en_ff : out std_logic;
        enable: out std_logic;
        shift_en: out std_logic;
        fshift_en : out std_logic;
        load: out std_logic;
        load_add: out std_logic;
        stop: out std_logic
    );
end Control_Unit;

architecture Behavioral of Control_Unit is

type state is (idle,init,check,sum,shift,subtract,final_shift);

```

```

signal current, nxt: state := idle;

begin

process_state: process(clock, reset)
begin
    if(clock' event and clock='1') then
        if (reset='1') then
            current <= idle;
        else
            current <= nxt;
        end if;
    end if;
end process;

fsm: process(current, start, reset,q0,end_cont)

begin

case current is

    when idle =>
        count_enable <= '0';
        subt <= '0';
        en_mux <= '0';
        enable <= '0';
        shift_en <= '0';
        load <= '0';
        stop <= '0';
        en_ff <= '1';
        if(start='1') then
            nxt <= init;
        else
            nxt <= idle;
        end if;

    when init =>
        enable <= '1';
        load <= '1';
        nxt <= check;
        en_ff <= '1';

    when check =>
        enable <= '0';
        load <= '0';
        load_add <= '1';
        nxt <= sum;
        en_mux <= q0;
        count_enable <= '1';
        shift_en <= '0';
        fshift_en <= '0';
        en_ff <= '1';
        if(end_cont = "0111") then
            nxt <= subtract;
            subt <= '1';
        end if;
end if;

```

```

        when subtract =>
            nxt <= final_shift;
            shift_en <= '0';
            fshift_en <= '1';
            subt <= '0';
            load_add <= '0';
            count_enable <= '0';
            stop <= '1';
            en_ff <= '0';

        when sum =>
            nxt <= shift;
            load_add <= '0';
            count_enable <= '0';
            shift_en <= '1';
            fshift_en <= '0';
            en_ff <= '0';

        when shift =>
            nxt <= check;
            shift_en <= '0';
            fshift_en <= '0';
            en_ff <= '0';

        when final_shift =>
            shift_en <= '0';
            fshift_en <= '0';
            nxt <= idle;
            en_ff <= '0';

    end case;

end process;

end Behavioral;

```

11.4.3 | Ripple Carry Adder

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity RippleCarryAdd is
generic(
    N : integer := 4
);
Port (
    A : in std_logic_vector(N-1 downto 0);
    B : in std_logic_vector(N-1 downto 0);
    S: out std_logic_vector(N-1 downto 0);
    C_in: in std_logic;
    C_out: out std_logic
);
end RippleCarryAdd;

architecture Structural of RippleCarryAdd is
component FullAdder is
Port (
    a,b,c_in :in std_logic;
    s,c_out :out std_logic

```

```
        );
end component;
signal sx : std_logic_vector(N-1 downto 0);
signal cx : std_logic_vector(N-1 downto 0);

begin
Gen: for i in 0 to N-1 generate
    gen_FA0: if i=0 generate
        FA_0: FullAdder
        port map(
            a => A(i),
            b => B(i),
            c_out => cx(i),
            c_in =>C_in,
            s=>S(i)

        );
    end generate;

gen_FA_N_1: if i<N-1 and i>0 generate
    FA_N_1: FullAdder
    port map(
        a => A(i),
        b => B(i),
        c_in => cx(i-1),
        c_out => cx(i),
        s=>S(i)

    );
end generate;

gen_FA_N: if i=N-1 and i>0 generate
    FA_N: FullAdder
    port map(
        a => A(i),
        b => B(i),
        c_in => cx(i-1),
        c_out => C_out,
        s=>S(i)

    );
end generate;

end generate;
end Structural;
```

12 | Esercizio Libero

Progettare, implementare in VHDL e testare mediante simulazione (e, optionalmente, mediante sintesi su board), un sistema le cui specifiche siano definite dallo studente e rientrino in una delle seguenti tipologie:

- Modifica di esercizi già proposti (processore, rete di interconnessione o interfaccia seriale) mediante aggiunta/aggiornamento di funzionalità.
Esempio: si potrebbe pensare di modificare l'interfaccia seriale aggiungendo segnali specifici per l'handshaking fra due entità.
- Progetto di sistemi che assolvono a specifici compiti noti
Esempio: si potrebbe pensare di implementare una specifica macchina aritmetica non trattata a lezione, una funzione crittografica, una rete neurale, ecc.
- Progetto di sistemi che integrano opportunamente componenti visti a lezione (contatori, registri, macchine aritmetiche, ecc.).

A titolo di esempio, è possibile fare riferimento ai seguenti due esercizi:

- Progettare un sommatore di byte seriale (le cifre degli addendi devono essere fornite serialmente a coppie alla macchina) a partire da un sommatore di bit.
Il sommatore deve terminare le sue operazioni appena il valore temporaneo della somma diventa maggiore di un valore M fornito in input.
- Si consideri un nodo A che contiene una memoria ROM di N ($N \geq 4$) locazioni da 8 bit ciascuna.
Progettare un sistema in grado di trasmettere mediante handshaking completo tutti i valori strettamente positivi contenuti nella memoria di A ad un nodo B.
Il nodo B, ricevuti i valori da A, li trasmetterà ad un nodo C mediante una comunicazione parallela con handshaking.

Traccia scelta Abbiamo deciso di svolgere la traccia d'esame del dicembre 2022, il testo è il seguente: progettare, implementare in VHDL e simulare un sistema composto da 2 nodi, A e B.

A trasmette serialmente a B una stringa memorizzata in un registro a scorrimento di 24 bit.

B riceve la stringa ed estrae da essa 3 operandi da 8 bit ciascuno, OP1, OP2 e OP3 (sottostringhe da 8 bit). B confronta OP1 e OP2, se OP1=OP2 effettua un secondo confronto con OP3, memorizzando un carattere 00 internamente se risultano uguali, in caso contrario (se OP1!=OP2 o se OP1=OP2 ma OP1!=OP3) B memorizza un carattere 11.

Si integri un componente comparatore in B, e un contatore in A e B.

Per rendere più completo l'esercizio abbiamo deciso di prevedere una memoria sia nel nodo A che nel nodo B per l'invio di N stringhe e il relativo salvataggio dei risultati.

12.1 | Descrizione Soluzione

Abbiamo deciso di imparci di non utilizzare un componente UART e quindi implementare un coppia TX - RX che effettua un protocollo di handshake per l'invio di ogni bit.

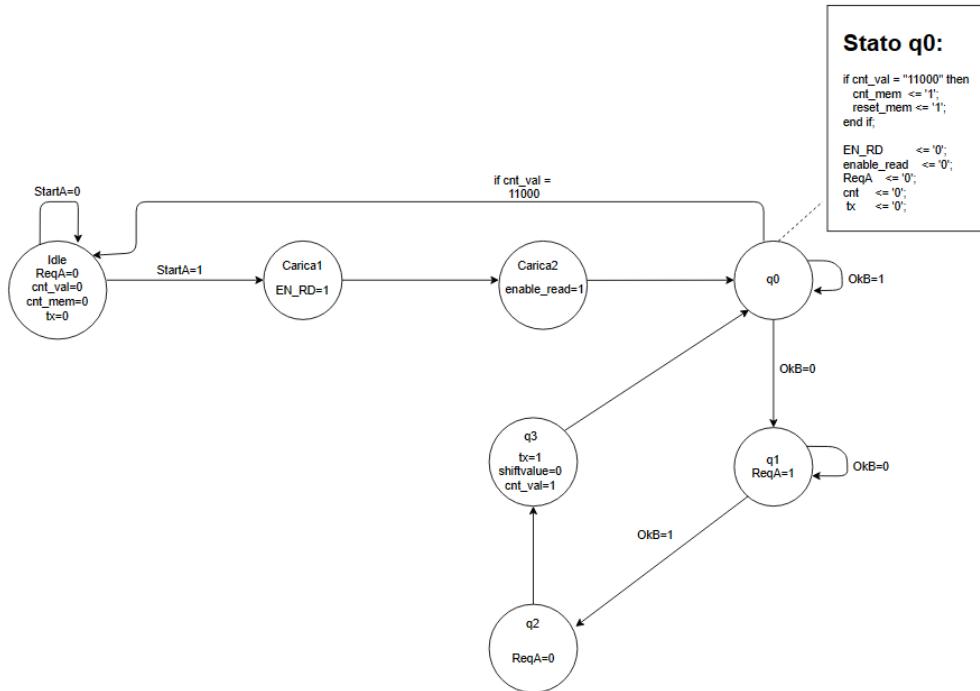
Per descrivere la comunicazione tra il nodo A e il nodo B abbiamo definito un handshake simile a quello del esercizio 7, l'attenzione è stata posta nella durata effettiva della trasmissione del dato e della sua ricezione poiché sono stati utilizzati degli shift register e quindi è stato fatto in modo che il ricevitore non scriva più volte del dovuto un singolo bit.

12.1.1 | Struttura Nodo A

Abbiamo costruito il nodo A dividendo in parte operativa e parte di controllo.

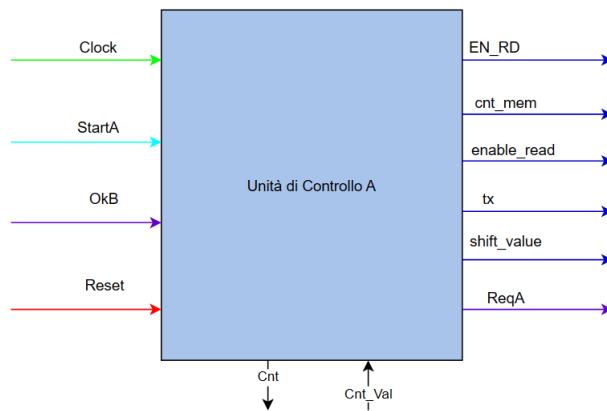
Parte di controllo l'unità è stata realizzata, come al solito, partendo da un automa visibile in figura 12.1. Ogni stato ha un compito ben preciso e quindi gestisce alcuni segnali ben precisi:

- Idle - Abbiamo deciso che la macchina inviava una stringa ogni volta che riceveva dall'esterno un segnale di start, questo stato rappresenta l'inattività del sistema, nonché lo stato di reset.

**Figura 12.1:** Automa del nodo A.

- Carica 1 - questo primo stato, come da nome, si occupa di eseguire il caricamento della stringa dalla ram verso lo shift register gestendo anche l'apposito contatore.
- Carica 2 - questo ulteriore stato esegue il caricamento in parallelo dello shift register.
- Q0 - si occupa di verificare di aver inviato 24 bit e che il nodo B sia disponibile a ricevere un altro bit.
- Q1 - è lo stato che invia la request al nodo B ed aspetta la sua risposta.
- Q2 - si occupa di effettuare la trasmissione del dato.
- Q3 - esegue lo shift dei dati ed aggiorna i contatori.

Complessivamente l'interfaccia del nodo A risulta essere composta da pochi segnali e dalla presenza di un contatore che scandisce il numero di bit inviati, nella figura 12.2.

**Figura 12.2:** Interfaccia unità di controllo del nodo A.

Parte operativa il datapath in questo caso è molto semplice, vedi figura 12.3, infatti è composto dall'unità di controllo, uno shift register e dalla memoria, la quale è stata ripresa anch'essa dal esercizio 7 quindi contiene internamente il proprio contatore.

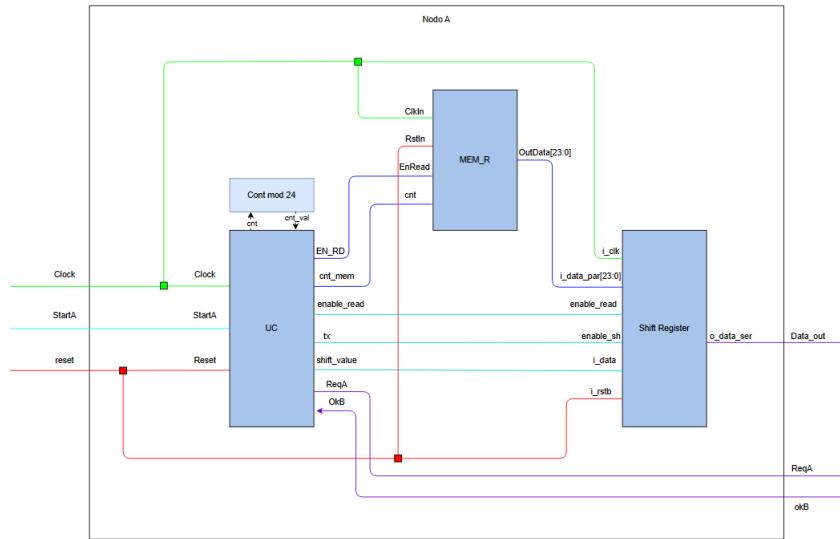


Figura 12.3: Struttura del nodo A.

12.1.2 | Struttura Nodo B

Parte di Controllo questa è stata realizzando partendo dal automa in figura 12.4, in breve caratterizziamo tutti gli stati:

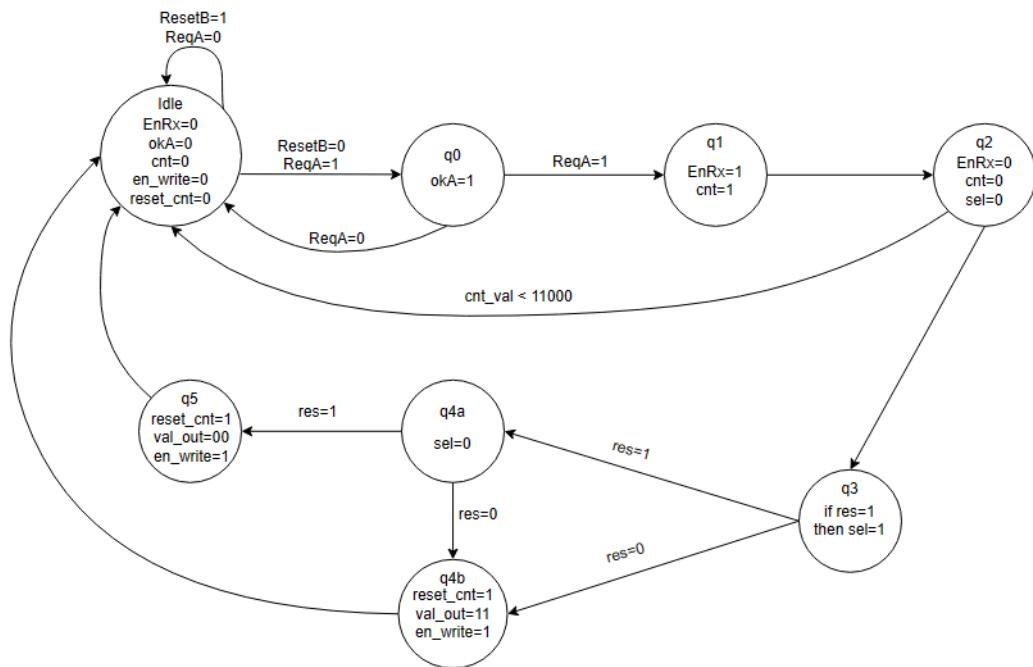


Figura 12.4: Automa del nodo B

- **Idle** - il nodo B tramite questo stato è sempre in attesa di una richiesta di A, inoltre funge da stato di reset.

- Q0 - questo nodo accoglie una richiesta da A e fa in modo che il segnale di request sia alto almeno per 2 colpi di clock, poiché abbiamo deciso che il protocollo deve utilizzare un primo intervallo di tempo per preparare i nodi e in un successivo intervallo avviene la trasmissione.
 - Q1 - questo stato si occupa della ricezione tramite lo shift register.
 - Q2 - in questo stato si controlla il numero di bit ricevuti e si decide se mettersi in attesa di un altro oppure se passare alla elaborazione del dato.
 - Q3 - con questo stato viene valutata la condizione $OP1 = OP2$ e si decide se salvare un 00 oppure passare a valutare anche $OP1 = OP3$.
 - Q4a - questo stato valuta la condizione $OP1 = OP3$.
 - Q4b - questo stato è incaricato di salvare in memoria 00.
 - Q5 - questo stato è incaricato di salvare in memoria 11.

Parte operativa questa unità è stata realizzata nel modo più semplice possibile, come da figura 12.5 è possibile notare un MUX per introdurre in un comparatore l'OP2 o L'OP3.

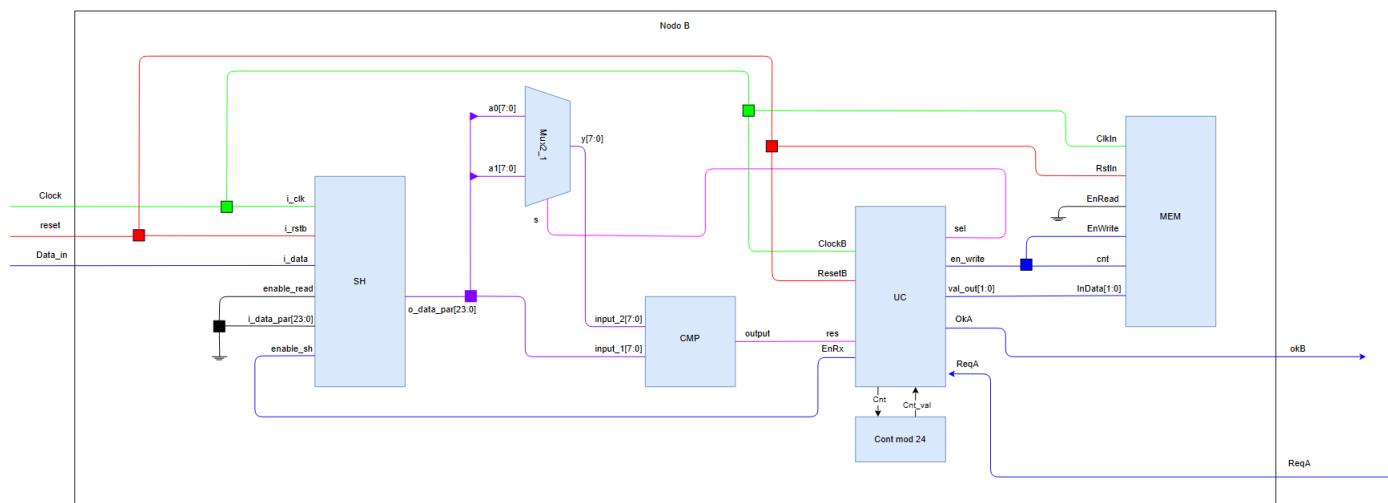


Figura 12.5: Architettura del nodo B

12.2 | Testbench

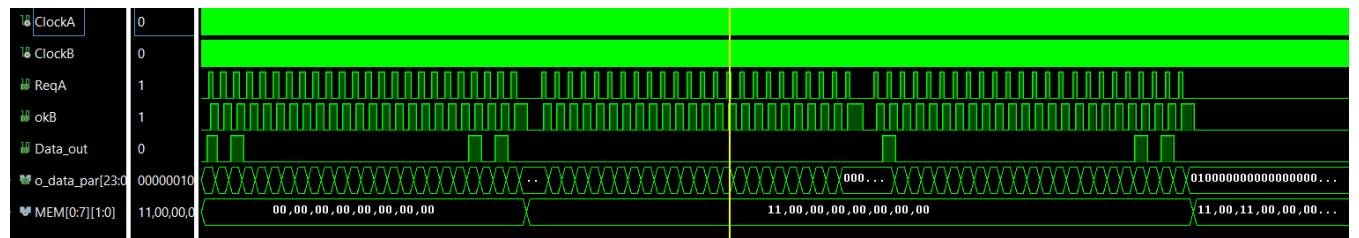


Figura 12.6: Testbench comunicazione seriale.

12.3 | Codice

12.3.1 | Nodo A

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

```

entity Tx_seriale is
  port (
    Data_out: out std_logic;

    ReqA: out std_logic;
    okB: in std_logic;

    Clock: in std_logic;
    reset: in std_logic;

    StartA: in std_logic
  );
end Tx_seriale;

architecture Behavioral of Tx_seriale is

component UC_1 is
  Port (
    Clock: in std_logic;
    Reset: in std_logic := '0';

    StartA: in std_logic;
    ReqA: out std_logic;
    OkB: in std_logic := '0';

    --segnali controllo parte operativa
    rx: out std_logic;
    shift_value: out std_logic;
    enable_read: out std_logic;
    EN_RD: out std_logic;
    stopMEM: in std_logic;
    cnt_mem: out std_logic
  );
end component;

component shift_register is
generic(
  data_len: natural := 23
);
port (
  i_clk          : in  std_logic;
  i_rstb         : in  std_logic;
  i_data          : in  std_logic;
  i_data_par     : in  std_logic_vector(data_len downto 0);
  o_data_par     : out std_logic_vector(data_len downto 0);
  o_data_ser     : out std_logic;
  enable_read    : in  std_logic;
  enable_sh      : in  std_logic
);
end component;

component L_MEM_R is
  generic(
    DataLen: natural :=24
  );
  Port(
    ClkIn:      in  STD_LOGIC;

```

```

        RstIn:      in STD_LOGIC;
        EnRead:     in STD_LOGIC;
        cnt:        in STD_LOGIC;
        OutData:    out std_logic_vector(DataLen downto 0)
    );
end component;

signal enable_sh: std_logic;
signal enable_read: std_logic;
signal i_data: std_logic;
signal Data_out_par: std_logic_vector(23 downto 0);
signal i_data_par: std_logic_vector(23 downto 0);

---Lettura rom
signal EN_RD: std_logic;
signal stopMEM: std_logic;
signal cnt_mem: std_logic;
begin

UC: UC_1
port map(
    Clock => Clock,
    Reset => reset,

    StartA => StartA,
    ReqA    => ReqA,
    OkB     => okB,

    --segnali controllo parte operativa
    rx        => enable_sh,
    enable_read => enable_read,
    shift_value => i_data,
    EN_RD      => EN_RD,
    stopMEM    => stopMEM,
    cnt_mem    => cnt_mem
);

SR: shift_register
generic map(
    data_len => 23
)
port map (
    i_clk      => Clock,
    i_rstb    => reset,
    i_data     => i_data,
    i_data_par => i_data_par,
    o_data_par => Data_out_par,
    o_data_ser => Data_out,
    enable_read => enable_read,
    enable_sh   => enable_sh
);

MEM_R: L_MEMORY
generic map(
    DataLen => 23
)
Port map (

```

```

        ClkIn      => Clock,
        RstIn      => reset,
        EnRead    => EN_RD,
        cnt        => cnt_mem,
        OutData   => i_data_par
    );

```

```
end Behavioral;
```

12.3.2 | Nodo A - UC

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

```

```
entity UC_1 is
```

```

Port (
    Clock: in std_logic;
    Reset: in std_logic := '0';

    StartA: in std_logic;
    ReqA: out std_logic;
    OkB: in std_logic := '0';

    --segnali controllo parte operativa
    rx: out std_logic;
    shift_value: out std_logic;
    enable_read: out std_logic;

    --segnali memoria
    EN_RD: out std_logic;
    stopMEM: in std_logic;
    cnt_mem: out std_logic
);

```

```
end UC_1;
```

```
architecture Behavioral of UC_1 is
```

```

---contatore tx
    signal cnt: std_logic;
    signal cnt_val: std_logic_vector(4 downto 0);
    signal reset_mem: std_logic;

```

```

component ContModN is
generic(
    N:natural:=4
);
Port ( clock : in STD_LOGIC;
       reset : in STD_LOGIC;
       enable : in STD_LOGIC; --questo è l'enable del clock, insieme danno l'impulso di
       counter : out STD_LOGIC_VECTOR (N downto 0)
);
end component;

```

```

type stato is (idle, q0, q1, q2, q3, carica1, carica2);
signal stato_corrente : stato := idle;
signal stato_next: stato;

```

```

begin

--Evoluzione stati
process(Clock, Reset, stato_corrente)
begin

    if Reset = '1' then
        stato_corrente <= idle;
    elsif (Clock'event and Clock = '1') then
        stato_corrente <= stato_next;
    end if;

    case stato_corrente is
    when idle =>
        if StartA = '1' then
            stato_next <= carica1;
        elsif StartA = '0' then
            stato_next <= idle;
        end if;
    when carica1 =>
        stato_next <= carica2;
    when carica2 =>
        stato_next <= q0;
    when q0 =>
        if cnt_val < "11000" then
            if OkB = '1' then
                stato_next <= q0;
            elsif OkB = '0' then
                stato_next <= q1;
            end if;
        elsif cnt_val = "11000" then
            stato_next <= idle;
        end if;
    when q1 =>
        if OkB = '0' then
            stato_next <= q1;
        elsif OkB = '1' then
            stato_next <= q2;
        end if;
    when q2 =>
        stato_next <= q3;
    when q3 =>
        stato_next <= q0;

    end case;
end process;

--OUTPUT
process(stato_corrente)
begin
    case stato_corrente is
        when idle =>
            ReqA      <= '0';
            cnt       <= '0';
            cnt_mem   <= '0';
            rx        <= '0';
            reset_mem <= '0';
        when carica1 =>

```

```

        if cnt_val = "00000" then
            EN_RD    <= '1';
        end if;
    when carica2 =>
        if cnt_val = "00000" then
            enable_read    <= '1';
        end if;
    when q0      =>
        if cnt_val = "11000" then
            cnt_mem    <= '1';
            reset_mem <= '1';
        end if;
        EN_RD          <= '0';
        enable_read   <= '0';
        ReqA           <= '0';
        cnt            <= '0';
        rx             <= '0';
    when q1      =>
        ReqA           <= '1';
    when q2      =>
        ReqA           <= '0';
    when q3      =>
        rx             <= '1';
        shift_value   <= '0';
        cnt            <= '1';
    end case;
end process;

cont: ContModN
port map(
    clock => Clock,
    reset => reset_mem,
    enable => cnt,
    counter => cnt_val
);

```

end Behavioral;

12.3.3 | Nodo B

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Rx_seriale is
    Port (
        Data_in: in std_logic;

        ReqA: in std_logic;
        okB: out std_logic;

        Clock: in std_logic;
        reset: in std_logic
    );
end Rx_seriale;

architecture Behavioral of Rx_seriale is

component UC_B is
    Port (

```

```

ClockB: in std_logic;
ResetB: in std_logic := '0';

ReqA: in std_logic;
OkA: out std_logic := '0';

--segnali controllo parte operativa
EnRx: out std_logic;
sel: out std_logic;
res: in std_logic;
en_write: out std_logic;
val_out: out std_logic_vector(1 downto 0)
);
end component;
```

```

component shift_register is
generic(
    data_len: natural := 23
);
port (
    i_clk: in std_logic;
    i_rstb: in std_logic;
    i_data: in std_logic;
    i_data_par: in std_logic_vector(data_len downto 0);
    o_data_par: out std_logic_vector(data_len downto 0);
    o_data_ser: out std_logic;
    enable_read: in std_logic;
    enable_sh: in std_logic
);
end component;
```

```

component comparatore is
generic(N:natural:=4);
Port (
    input_1: in std_logic_vector(N-1 downto 0);
    input_2: in std_logic_vector(N-1 downto 0);
    output: out std_logic --uscita 1 equivale a ingressi uguali viceversa per uscita 0
);
end component;
```

```

component mux_21_gen is
    generic (
        N: natural:= 8
    );
    Port (
        a0, a1: in std_logic_vector(N-1 downto 0);
        s: in std_logic;
        y: out std_logic_vector(N-1 downto 0)
    );
end component;
```

```

component L_MEM_RW is
generic(

    DataLen: natural :=3
);
Port(

    ClkIn: in STD_LOGIC;
    RstIn: in STD_LOGIC;

```

```

        EnRead:    in STD_LOGIC;
        EnWrite:   in STD_LOGIC;
        cnt:       in STD_LOGIC;
        InData:    in std_logic_vector(DataLen downto 0);
        OutData:   out std_logic_vector(DataLen downto 0)
    );
end component;

signal o_data_par: std_logic_vector(23 downto 0);
signal enableh_sh: std_logic;
signal o_data_ser: std_logic;
signal res: std_logic;
signal en_write: std_logic;
signal val_out: std_logic_vector(1 downto 0);
signal o_data_sel: std_logic_vector(7 downto 0);
signal sel: std_logic;
signal OutData: std_logic_vector(1 downto 0);

begin

UC: UC_B
port map(
    ClockB => Clock,
    ResetB => reset,

    ReqA => ReqA,
    OkA  => okB,

    --segnali controllo parte operativa
    EnRx => enableh_sh,
    res  => res,
    sel   => sel,
    en_write => en_write,
    val_out  => val_out
);

SH: shift_register
generic map(
    data_len => 23
)
port map (
    i_clk      => Clock,
    i_rstb     => reset,
    i_data     => Data_in,
    i_data_par => (others => '0'),
    o_data_par => o_data_par,
    o_data_ser => o_data_ser,
    enable_read => '0',
    enable_sh   => enableh_sh
);

CMP: comparatore
generic map (
    N => 8
)
port map (
    input_1 => o_data_par(7 downto 0),
    input_2 => o_data_sel,

```

```

        output  => res
);

MUX: mux_21_gen
generic map (
    N => 8
)
Port map(
    a0 => o_data_par(15 downto 8),
    a1 => o_data_par(23 downto 16),
    s  => sel,
    y  => o_data_sel
);

MEM: L_MEMORY
generic map(
    DataLen => 1
)
Port map (
    ClkIn      => Clock,
    RstIn      => reset,
    EnRead     => '0',
    EnWrite    => en_write,
    cnt        => en_write,
    InData     => val_out,
    OutData    => OutData
);
end Behavioral;

```

12.3.4 | Nodo B - UC

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity UC_B is
    Port (
        ClockB: in std_logic;
        ResetB: in std_logic := '0';

        ReqA: in std_logic;
        OkA: out std_logic := '0';

        --segnali controllo parte operativa
        EnRx: out std_logic;
        sel:  out std_logic;
        res:  in std_logic;
        en_write: out std_logic;
        val_out:  out std_logic_vector(1 downto 0)
    );
end UC_B;

architecture Behavioral of UC_B is

    ---contatore rx
    signal cnt: std_logic;
    signal cnt_val: std_logic_vector(4 downto 0);

    component ContModN is

```

```

generic(
    N:natural:=4
);
Port ( clock : in STD_LOGIC;
       reset : in STD_LOGIC;
       enable : in STD_LOGIC; --questo è l'enable del clock, insieme danno l'impulso di
       counter : out STD_LOGIC_VECTOR (N downto 0)
);
end component;

type stato is (idle,q0,q1, q2, q3, q4a, q4b, q5);
signal stato_corrente : stato := idle;
signal stato_next: stato;
signal reset_mem: std_logic;

begin

--Process evoluzione stati
process (ClockB, ResetB)
begin
    if ResetB = '1' then
        stato_corrente <= idle;
    elsif(ClockB'event and ClockB = '1') then
        stato_corrente <= stato_next;
    end if;

    case stato_corrente is
    when idle =>
        if (ReqA = '0') then
            stato_next <= idle;
        elsif (ReqA = '1') then
            stato_next <= q0;
        end if;
    when q0 =>
        if ReqA = '0' then
            stato_next <= idle;
        elsif ReqA = '1' then
            stato_next <= q1;
        end if;
    when q1 =>
        stato_next <= q2;
    when q2 =>
        if cnt_val < "11000" then
            stato_next <= idle;
        else
            stato_next <= q3;
        end if;
    when q3 =>
        if res = '1' then
            stato_next <= q4a;
        elsif res = '0' then
            stato_next <= q4b;
        end if;
    when q4a =>
        if res = '1' then
            stato_next <= q5;
        elsif res = '0' then
            stato_next <= q4b;
        end if;
    end case;
end process;

```

```

        end if;
when q4b =>          stato_next <= idle;
when q5 =>           stato_next <= idle;
end case;
end process;

--OUTPUT
process(stato_corrente)
begin
  case stato_corrente is
    when idle =>
      EnRx     <= '0';
      okA      <= '0';
      cnt       <= '0';
      en_write <= '0';
      reset_mem <= '0';
    when q0 =>
      okA      <= '1';
    when q1 =>
      EnRx     <= '1';
      cnt       <= '1';
    when q2 =>
      EnRx     <= '0';
      cnt       <= '0';
      sel       <= '0';
    when q3 =>
      if res = '1' then
        sel <= '1';
      end if;
    when q4a =>
      sel <= '0';
    when q4b =>
      reset_mem <= '1';
      val_out  <= "11";
      en_write <= '1';
    when q5  =>
      reset_mem <= '1';
      val_out  <= "00";
      en_write <= '1';
  end case;
end process;

cont: ContModN
port map(
  clock => ClockB,
  reset => reset_mem,
  enable => cnt,
  counter => cnt_val
);
end Behavioral;

```