Indice

1 Esercizio 1 - Multiplexer **2**
1.1 Parte 1 2
1.2 Parte 2 7
1.3 Parte 3 10

2 Esercizio 2 - Encoder BCD **11**
2.1 Traccia 11
2.2 Soluzione 11
2.3 Codice 13

3 Esercizio 3 - Riconoscitore di Sequenze **16**

4 Esercizio 4 - Shift Register **17**

5 Esercizio 5 - Cronometro **18**

6 Esercizio 6 - Sistema di Testing **19**

7 Esercizio 7 - Comunicazione con Handshaking **20**

8 Esercizio 8 - Processor **21**

9 Esercizio 9 - Interfaccia UART **22**

10 Esercizio 10 - Switch Multistadio **23**

11 Esercizio 11 - Divisore Restoring **24**

12 Esercizio 12 - Interfaccia VGA **25**

1 Esercizio 1 - Multiplexer

1.1 Parte 1

L'esercizio 1.1 richiede la rappresentazione di un multiplexer 16:1 tramite la composizione di multiplexer 4:1, quindi definiamo il module del componente mux_4_1 definito in modo Dataflow, e poi per le proprietà della modularità definiamo il module mux_16_1 come composizione dei precedenti, tramite il costrutto *for..generate*.
Il multiplexer 4:1 ha come ingressi quattro bit, identificati col vettore a(0 to 3), e $\lceil \log_2(n) \rceil$ segnali di abilitazione, dove n è il numero di segnali di ingresso. In questo caso, quindi, due segnali di abilitazione ed uno di uscita.

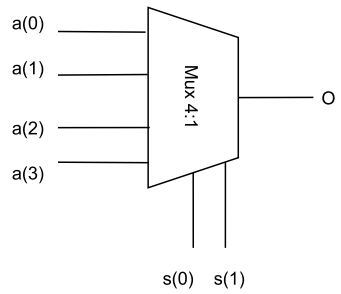

Figure 1: Mux 4:1

Il codice in VHDL per descrivere il comportamento di questo componente è il seguente:

```
architecture DataFlow of mux_4_1 is

begin
    o <= a(0) when s = "00" else
         a(1) when s = "01" else
         a(2) when s = "10" else
         a(3) when s = "11" else
         '-';

end DataFlow;
```

Figure 2: Mux 4:1 Dataflow

Composto da un costrutto *when...else* che suddivide i diversi casi e gestisce anche tutti i casi non definiti con l'ultima clausola else senza alcuna condizione.
Definito l'elemento base del progetto, si passa a comporre il multiplexer 16:1 tramite un approccio strutturale, nel quale generiamo cinque mux_4_1: i primi quattro avranno gli ingressi interfacciati con l'esterno e, tramite segnali interni, le loro uscite sono collegate come ingressi dell'ultimo multiplexer che costituisce l'uscita del sistema. La macchina completa presenta quindi 16 segnali di ingresso, 4 segnali di selezione ed un unico segnale di uscita:

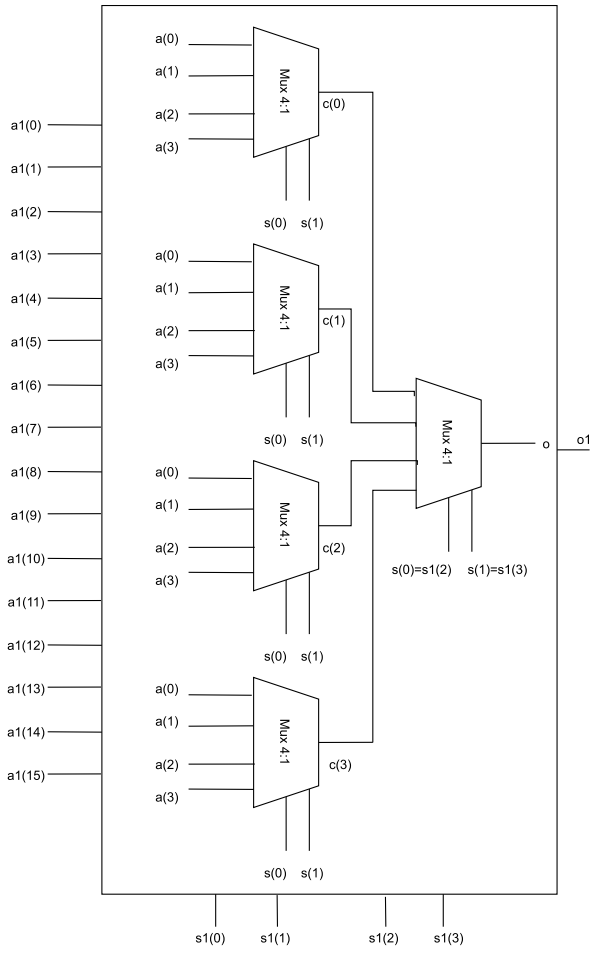

Figure 3: Mux 16:1

Lo schema sopra rappresentato è tradotto in linguaggio VHDL dal seguente codice, dove l'entità mux_16_1 è rappresentata con approccio strutturale e, tramite i segnali interni c(0 to 3), colleghiamo le uscite parziali dei primi quattro mux_4_1, identificati con la label mux_4_1_in, con i quattro ingressi dell'ultimo mux_4_1, identificato con la label mux_4_1_fin.

```
entity mux_16_1 is
    Port ( al : in STD_LOGIC_VECTOR (0 to 15);
           sl : in STD_LOGIC_VECTOR (0 to 3);
           ol : out STD_LOGIC);
end mux_16_1;

architecture Structural of mux_16_1 is
    COMPONENT mux_4_1 PORT (a : in STD_LOGIC_VECTOR (0 to 3); s : in STD_LOGIC_VECTOR (0 to 1); o : out STD_LOGIC);
    END COMPONENT;
    FOR ALL: mux_4_1 USE ENTITY WORK.mux_4_1 (Behavioral);
    signal c : STD_LOGIC_VECTOR (0 to 3);
begin
    mux_4_1_in : FOR i in 0 to 3 GENERATE m: mux_4_1
        port map (
        a(0 to 3) => al(i*4 to i*4+3),
        s(0 to 1) => sl(0 to 1),
        o => c(i)
        );
    end GENERATE;

    mux_4_1_fin : mux_4_1
        port map(
        a(0 to 3) => c(0 to 3),
        s(0 to 1) => sl(2 to 3),
        o => ol
    );
end Structural;
```

Figure 4: Mux 16:1 Dataflow

Progettato il mux_16_1, è possibile testarlo attraverso un testbench. La prima cosa che bisogna specificare è che il corpo dell'entity è vuoto, questo perché non si tratta di oggetto che realizziamo, ma serve solo per effettuare la simulazione e verificare se il sistema realizzato funziona correttamente. Il testbench effettivamente non ha né segnali d'ingresso né d'uscita, ma sfrutta per i test i segnali interni definiti nel codice. Per testare il mux_16_1 definito precedentemente, abbiamo istanziato una uut (Unit Under Test) in cui colleghiamo le varie porte ai segnali (input, selection, output).

```
SIGNAL input : STD_LOGIC_VECTOR (0 TO 15);
SIGNAL selection : STD_LOGIC_VECTOR (0 TO 3);
SIGNAL output : STD_LOGIC;

begin
mux_16_1 : mux_16 PORT MAP (a1(0 to 15)=>input(0 to 15), s1(0to 3)=>selection(0 to 3), o1=>output);

mux_16_2 : input <= "0000000000000000",
"0000000000000001" AFTER 100 NS,
"0000000000000011" AFTER 200 NS,
"1100000000000000" AFTER 300 NS;

mux_16_3 : selection <= "0000",
"1111" AFTER 100 NS,
"0011" AFTER 200 NS,
"1100" AFTER 300 NS;
end structural;
```

Figure 5: Mux 16:1 Testbench

Dopo aver effettuato queste assegnazioni, compreso di costrutto after per permettere l'evoluzione del sistema durante il tempo, si passa alla schermata di simulazione nella quale si può analizzare e studiare l'evoluzione nel tempo di ogni segnale presente nel codice, compresi eventuali segnali intermedi.

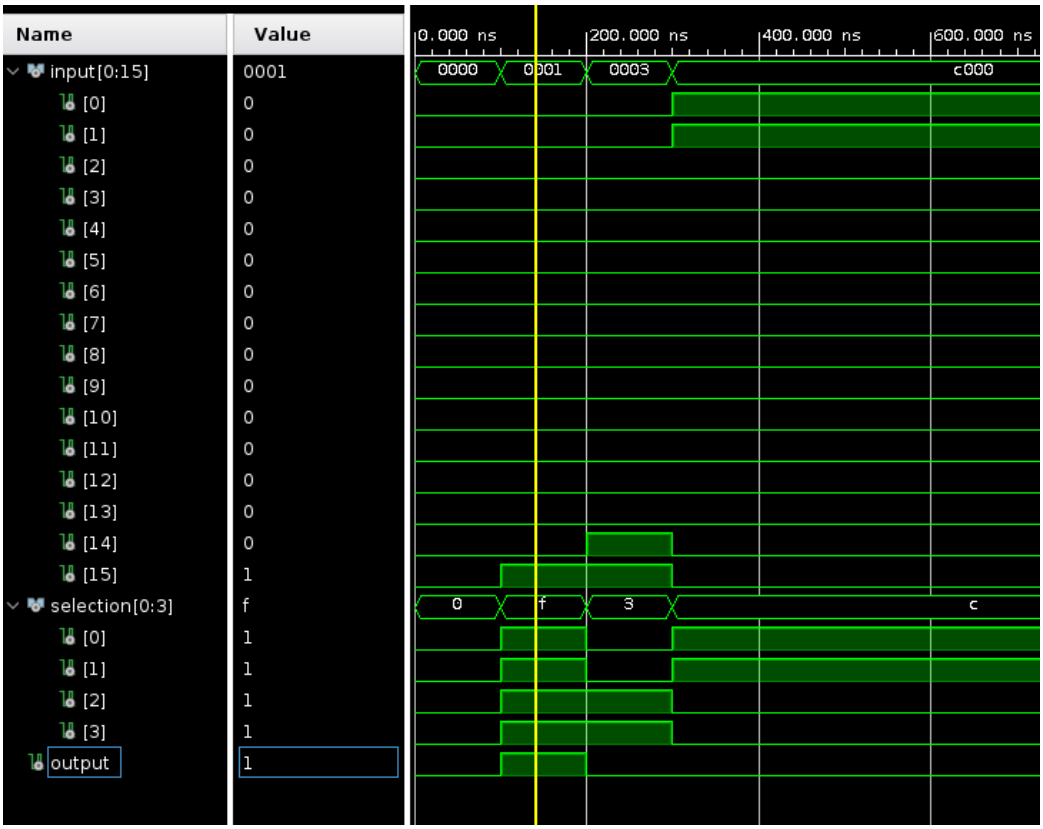

Figure 6: Mux 16:1 Simulazione

Possiamo analizzare l'evoluzione del programma: dopo circa 150ns, l'ingresso è posto a "0000000000000001" e contemporaneamente la selezione è posta a "1111", ottenendo come uscita del sistema "1". Questo è effettivamente il comportamento atteso.

1.2 Parte 2

L'esercizio 1.2 è in parte riconducibile all'esercizio precedente, in quanto la rappresentazione di una rete 16:4 può essere scomposta da una sottorete 16:1 connessa ad un demux 1:4. Basta quindi aggiungere un demultiplexer 1:4 alla rete precedente. Il demux 1:4 è realizzato con approccio Dataflow.

```
architecture DataFlow of demux_4_1 is
begin
    output <= input&"000" when enable="00" else
              '0'&input&"00" when enable="01" else
              "00"&input&'0' when enable="10" else
              "000"&input when enable="11" else
              "----";

end DataFlow;
```

Figure 7: Demux 1:4 Dataflow

Il Demux 1:4 presenta come uscita un segnale di 4 bit, di cui 3 pari a zero ed uno pari al valore in ingresso, la cui posizione è determinata a seconda dei segnali di abilitazione.

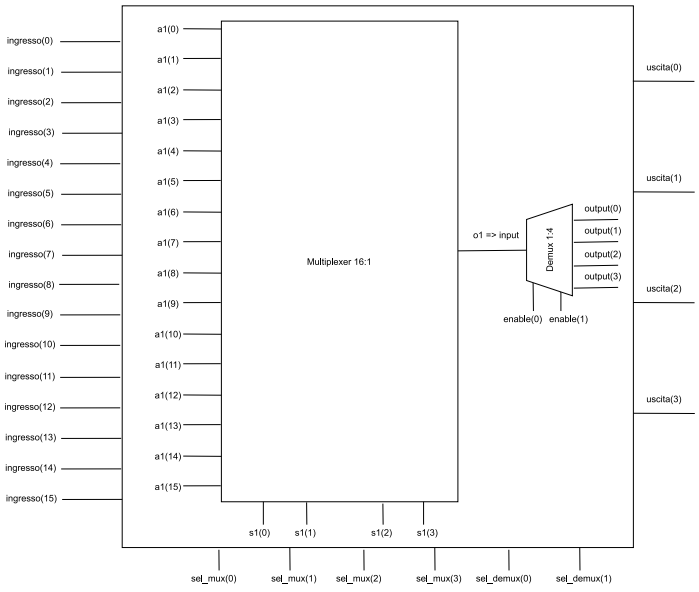

Figure 8: Rete 16:1

Una volta eseguite le interconnessioni tramite un unico segnale interno, utilizzato per collegare l'uscita del mux_16_1 con l'ingresso del demux_1_4, si ottiene la rete 16:4: tale rete presenta 16 segnali di ingressi totali, 6 di selezione (di cui 2 utilizzati per il demux) e 4 segnali di uscita.

```
signal interco : STD_LOGIC;
begin
    mux : mux_16_1
    PORT MAP (
    a1(0 to 15) => ingresso(0 to 15),
    s1( 0 to 3) => sel_mux(0 to 3),
    o1 => interco
    );
    demux : demux_4_1
    PORT MAP (
    input => interco,
    enable( 0 to 1) => sel_demux(0 to 1),
    output(0 to 3) => uscita ( 0 to 3)
    );


end Structural;
```

Figure 9: Rete 16:4 Structural

Anche dopo aver testato il singolo componente, è comunque necessario ripetere il test per la macchina completa, sia perché potrebbero essere presenti errori e problemi derivanti da implementazione di nuove funzioni, sia perché anche nella composizione di una macchina più complessa sono presenti intrinsecamente problemi legati alla coesione dei vari moduli.

```
begin
rete_16_4_1: rete_16_4 port map(
    ingresso (0 to 15) => i(0 to 15),
    uscita (0 to 3) => u (0 to 3),
    sel_mux(0 to 3) => s_mux(0 to 3),
    sel_demux(0 to 1) => s_demux(0 to 1));

rete_16_4_2: i <= "0000001000000000",
"0000000000000000" after 100ns,
"1000000000000000" after 200ns,
"1110111111111111" after 300ns;

rete_16_4_3: s_mux <= "1001",
"1111" after 100ns,
"0000" after 200ns,
"1100" after 300ns;

rete_16_4_4: s_demux <= "00",
"10" after 100ns;
```

Figure 10: Rete 16:4 Testbench

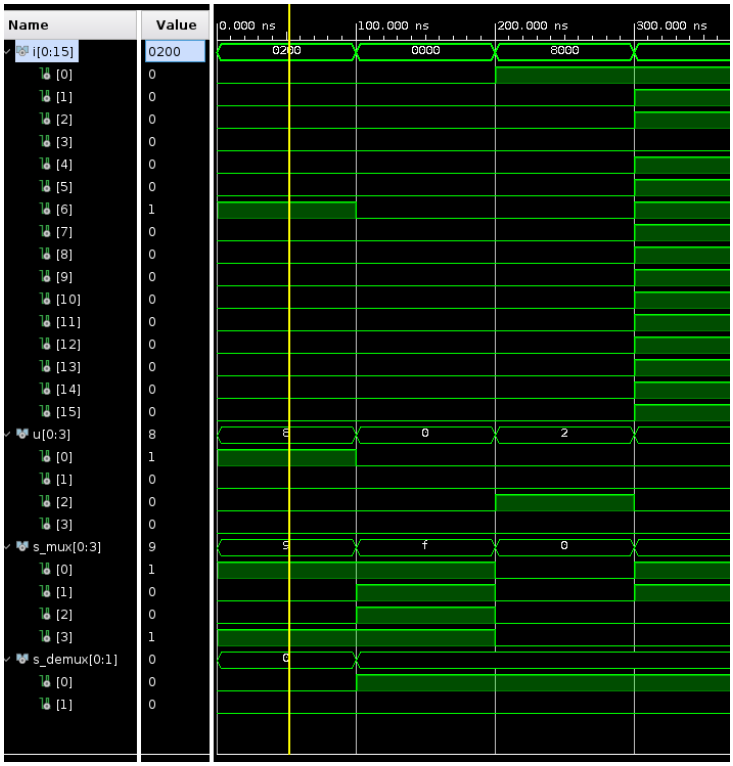

Figure 11: Rete 16:4 Simulazione

1.3 Parte 3

Come esercizio finale, è stato necessario adattare la rete per la sintesi sulla FPGA. Attraverso l'utilizzo di un file di constraint ideato per la board Nexys A7-50t, possiamo definire i collegamenti da effettuare sulla scheda tra le diverse periferiche disponibili e le componenti presenti all'interno della rete.
In questo caso sono necessari 6 switch per le linee di abilitazione dei multiplexer e demultiplexer. Poichè gli switch sono in totale 16, è stato necessario dare un input predefinito alla rete ed utilizzare gli switch unicamente per la selezione. Inoltre, sempre dal file di costraint, sono stati abilitati anche 4 led e connessi ai quattro bit di uscita, come indicato nel seguente file.

```
##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { sel_mux[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sel_mux[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sel_mux[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { sel_mux[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { sel_demux[0] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { sel_demux[1] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { input[0] }]; #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 { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[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_A09_D25_VREF_14 Sch=sw[14]
#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 { uscita[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { uscita[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { uscita[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { uscita[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { output[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { output[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
```

Figure 12: Costraints

Quindi, gli switch ***J15-L16-M13-R15*** sono stati mappati ai quattro bit di selezione dei multiplexer, mentre gli switch ***R17-T18*** ai due bit di selezione dei multiplexer, ed, infine, i led H17-K15-J13-N14 come rappresentazione visiva dei quattro bit di uscita al sistema complessivo.
Il sistema finale è quello raffigurato nella seguente figura

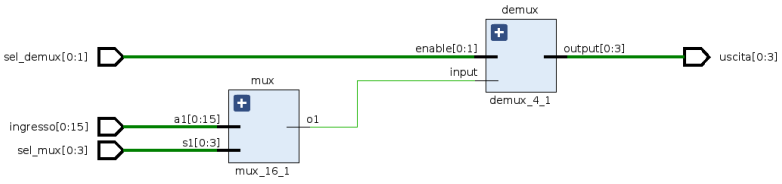

Figure 13: Rete 16:4 Analysis

2 Esercizio 2 - Encoder BCD

2.1 Traccia

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 Decimali (BCD).
Input: 0000000001 ⇒ Output: 0000 (cifra 0)
Input: 0000000010 ⇒ Output: 0001 (cifra 1)
Input: 0000000100 ⇒ Output: 0010 (cifra 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).
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 Soluzione

La rete è stata realizzata con vari componenti secondo un approccio strutturale. Essa ha un ingresso, cioè il valore della stringa X da 10 bit, e 2 uscite, utili per visualizzare la cifra codificata su un display a 7 segmenti. La rete utilizza i seguenti componenti:

- Un encoder: a sua volta composto da un arbitro a priorità e da un encoder 10:4;
- Un display manager per la visualizzazione dell'output;

Si è partiti, dunque, da una descrizione dataflow dei componenti base, per poi procedere con una descrizione strutturale dell'encoder ed una descrizione comportamentale del display manager. Infine, mettendo insieme questi ultimi 2 componenti, si è descritto l'intero sistema, chiaramente a livello strutturale.

Arbitro a priorità:

Il componente relativo all'arbitro di priorità dispone di un vettore di ingresso di 10 bit e di un vettore di uscita di altrettanti bit, in cui l'uscita avrà tutti 0 e un solo bit alto nella prima posizione in cui è stato trovato un 1 (a partire dalla posizione più significativa).

Encoder 10:4:

Il componente relativo all'encoder 10:4 presenta in ingresso un vettore di 10 bit ed in uscita un vettore di 4 bit, che rappresenta il numero in binario della prima posizione con bit alto in ingresso (valore compreso nel range [0,9]).

Facendo uso dei due componenti appena descritti, si è realizzato un encoder, il quale facendo uso di un segnale interno che fa da interconnessione tra l'uscita dell'arbitro e l'ingresso dell'encoder 10:4, prende in ingresso un vettore di 10 bit e restituisce in uscita un vettore di 4 bit effettuando la codifica Binary-Coded Decimal (BCD).

Display Manager:

Il componente display manager prende in ingresso un vettore di 4 bit e lo rappresenta sul display a 7 segmenti con 2 uscite. Al suo interno vengono definite delle costanti su 7 bit relative ai segmenti di una cifra del display da illuminare, in questo modo si visualizza un determinato valore in esadecimale. La prima uscita è fissa, al fine di illuminare costantemente solo la prima cifra del display, poiché il valore da mostrare è rappresentabile con una sola cifra. La seconda uscita determina i segmenti della cifra da illuminare, pertanto dipende dall'ingresso.
La rete complessiva dispone dell'encoder e del display manager. Il vettore di ingresso di 10 bit andrà nell'encoder e la sua uscita su 4 bit, mediante un segnale di interconnessione, andrà in ingresso al display manager che mostra il valore di uscita sul display.

2.3 Codice

Arbitro

L'architettura è stata descritta a livello dataflow e, scorrendo un vettore a partire dalla posizione 9 fino a 0, se tra i 10 bit uno solo è alto, l'uscità sarà una stringa con tutti 0 e solo un bit alto nella posizione in cui lo era nel vettore di ingresso. Nel caso in cui il vettore di ingresso presenta più bit uguali a 1, la stringa in uscita avrà il solo bit alto nella prima posizione, partendo dalla 9, in cui trova un 1.

```
entity arbitro is
    Port (
        x: in std_logic_vector(9 downto 0);
        y: out std_logic_vector(9 downto 0)
    );
end arbitro;

architecture dataflow of arbitro is

begin
        y <= "1000000000" when x(9) = '1' else
             "0100000000" when x(8) = '1' else
             "0010000000" when x(7) = '1' else
             "0001000000" when x(6) = '1' else
             "0000100000" when x(5) = '1' else
             "0000010000" when x(4) = '1' else
             "0000001000" when x(3) = '1' else
             "0000000100" when x(2) = '1' else
             "0000000010" when x(1) = '1' else
             "0000000001" when x(0) = '1' else
             "----------";

end dataflow;
```

Figure 14: Codice Arbitro

Encoder 10:4

L'encoder 10:4 è stato descritto a livello dataflow. L'architettura, partendo da un vettore con solo un bit alto, restituisce in uscita il valore su 4 bit della posizione in cui il bit è alto.

```
entity encoder10_4 is
    Port (
        x: in std_logic_vector(9 downto 0);
        y: out std_logic_vector(3 downto 0)
    );
end encoder10_4;

architecture dataflow of encoder10_4 is

begin
       with x select
            y <= "0000" when "0000000001",
                 "0001" when "0000000010",
                 "0010" when "0000000100",
                 "0011" when "0000001000",
                 "0100" when "0000010000",
                 "0101" when "0000100000",
                 "0110" when "0001000000",
                 "0111" when "0010000000",
                 "1000" when "0100000000",
                 "1001" when "1000000000",
                 "----" when others;

end dataflow;
```

Figure 15: Codice Encoder 10:4

Encoder complessivo

L'encoder complessivo è descritto a livello strutturale utilizzando i componenti "arbitro" e "encoder10_4". All'interno del sistema è definito un segnale t di tipo std_logic_vector(9 downto 0), che fa da interconnessione e viene utilizzato come uscita dell'arbitro e come ingresso dell'encoder 10:4.

```
entity encoder is
    Port (
        x: in std_logic_vector(9 downto 0);
        y: out std_logic_vector(3 downto 0)
    );
end encoder;

architecture structural of encoder is

    component arbitro port(
        x: in std_logic_vector(9 downto 0);
        y: out std_logic_vector(9 downto 0)
    );
    end component;

    component encoder10_4 port (
         x: in std_logic_vector(9 downto 0);
         y: out std_logic_vector(3 downto 0)
    );
    end component;

    signal t: std_logic_vector(9 downto 0);
```

Figure 16: Codice Encoder 10:4

```
begin
   arbitro_1: arbitro port map(
         x => x,
         y => t
   );
   encoder10_4_1:  encoder10_4 port map(
         x => t,
         y => y
   );

end structural;
```

Figure 17: Codice Encoder 10:4

3 Esercizio 3 - Riconoscitore di Sequenze

4 Esercizio 4 - Shift Register

5 Esercizio 5 - Cronometro

6 Esercizio 6 - Sistema di Testing

7 Esercizio 7 - Comunicazione con Handshaking

8 Esercizio 8 - Processor

9 Esercizio 9 - Interfaccia UART

10 Esercizio 10 - Switch Multistadio

11 Esercizio 11 - Divisore Restoring

12 Esercizio 12 - Interfaccia VGA

List of Figures

1 Mux 4:1 2
2 Mux 4:1 Dataflow 2
3 Mux 16:1 3
4 Mux 16:1 Dataflow 4
5 Mux 16:1 Testbench 5
6 Mux 16:1 Simulazione 6
7 Demux 1:4 Dataflow 7
8 Rete 16:1 7
9 Rete 16:4 Structural 8
10 Rete 16:4 Testbench 9
11 Rete 16:4 Simulazione 9
12 Costraints 10
13 Rete 16:4 Analysis 10
14 Codice Arbitro 13
15 Codice Encoder 10:4 14
16 Codice Encoder 10:4 15
17 Codice Encoder 10:4 15