Elettronica dei Sistemi Digitali
Lab 01

Multiplexers, Light Emitting Diodes, Switches and Testbench

Gruppo: A11

Pronesti, Massimiliano	S245831
Oropallo, Maria Vittoria	S245999
Nicolicchia, Riccardo	S244728
Miceli, Michelangelo	S245478

25 marzo 2020

Indice

1 Controlling the LEDs **2**
1.1 Spiegazione teorica 2
1.2 Procedimento 2
1.3 Risultati 2
1.4 Appendice 3

2 2-to-1 Multiplexer **5**
2.1 Spiegazione teorica 5
2.2 Procedimento 5
2.3 Risultati 6
2.4 Appendice 7

3 5-to-1 Multiplexer **10**
3.1 Spiegazione teorica 10
3.2 Procedimento 10
3.3 Risultati 12
3.4 Appendice 12

Capitolo 1

Controlling the LEDs

1.1 Spiegazione teorica

Lo scopo dell'esercizio 1 è quello di utilizzare gli interruttori della scheda DE2 e mostrare i loro stati attraverso l'accensione di LED rossi. In particolare, la scheda presente 18 switches, denominati SW_{17-0}, input del circuito, e 18 LED rossi, denominati $LEDR_{17-0}$, rappresentanti gli output.
La metodologia procedurale che si è seguita ha previsto la creazione di un opportuno file .vhd ,contenete il codice già fornito dal testo dell'esercitazione, ed includendo nel progetto il file .csv contenente le assegnazioni dei pin.

1.2 Procedimento

Sia gli input che gli output sono stati rappresentati in forma vettoriale. A ciascuna luce $LEDR_x$, è stato associato il corrispondente interruttore SW_x.

1.3 Risultati

Per verificare il funzionamento del circuito è stato realizzato un testbench che assegna dei bit casuali agli switch. La forma d'onda risultante, mostrata in **figura 1.1**, mostra che, per ogni switch a cui è assegnato il valore '1' segue l'accensione del corrispondente LED.

Viceversa, se lo switch vale zero, anche il corrispondente LED ha valore '0' (spento).

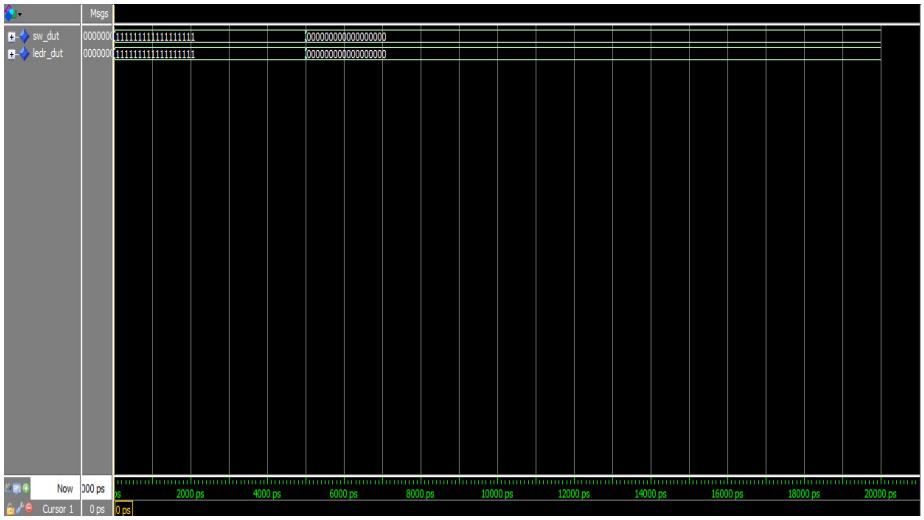


Figura 1.1: waveshot exercise no. 1

Figura 1.2: synth exercise no. 1

1.4 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  -- Simple module that connects the SW switches to the LEDR lights
5
6  entity light is
7  port (
8  SW : in std_logic_vector(17 downto 0);
```

```
LEDR : out std_logic_vector(17 downto 0) --red LEDs
);
end light;

architecture behavior of light is
begin
LEDR <= SW;
end behavior;
```

Code 1.1: VHDL code exercise no. 1

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

  entity light_tb is
  end light_tb;

  architecture Behavioral of light_tb is

  --device under test

  --signals
  signal sw_dut,ledr_dut : std_logic_vector (17 downto 0);

  begin
  -- implicit process testcase

  sw_dut <= "11111111111111111", "000000000000000000" after 5 ns;

  -- testing instance
  DUT: entity work.light port map (sw_dut , ledr_dut);

  end Behavioral;
```

Code 1.2: testbench exercise no. 2

Capitolo 2

2-to-1 Multiplexer

2.1 Spiegazione teorica

La seconda parte dell'esercitazione ha previsto l'implementazione di un multiplexer 2-to-1 ad 8 bit. Nell'ottica della riutilizzabilità del codice (ed alle successive implementazioni richieste dalla terza parte dell'esercitazione), si è proceduto, alla definizione di un multiplexer a 2 vie di parallelismo parametrico dw (data width). Settando opportunamente $dw = 8$ si ottiene, pertanto, il multiplexer desiderato, come mostrato nell'apposito testbench mostrato in **code 2.1**.
Il circuito riceve tre input, x, y e s (selettore), e ha output m su dw bit corrispondente ad x od ad y in base alla combinazione di valori assunta da s.

2.2 Procedimento

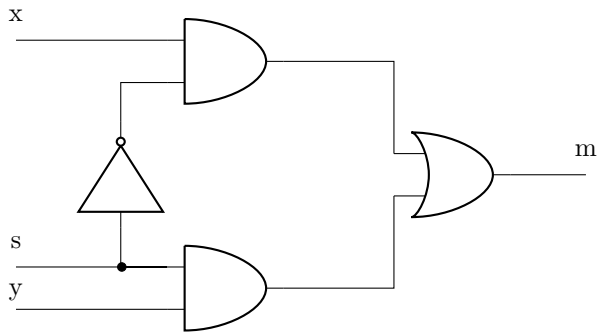
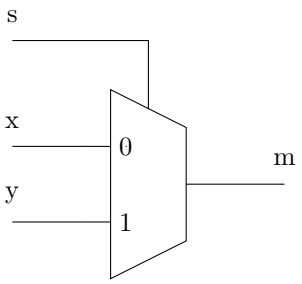


Figura 2.1: 2-to-1 MUX logic gates circuit

Si è instanziato il multiplexer passando come parametro $dw = 8$ (come mostrato nell'apposito testbench, **code 2.2**) ottenendo così ingressi del parallelismo desiderato.



s	m
0	x
1	y

Figura 2.2: 2-to-1 MUX and truth table

2.3 Risultati

Sono stati assegnati ad s i valori '0' e '1', e agli input i valori d'esempio x = "00000000" e y = "11111111". In **figura 2.3** si vede che, inizialmente, m = x e, dopo 20 ns, cambiando il valore del selettore, cambia il valore dell'uscita, i.e. m = y, come atteso.

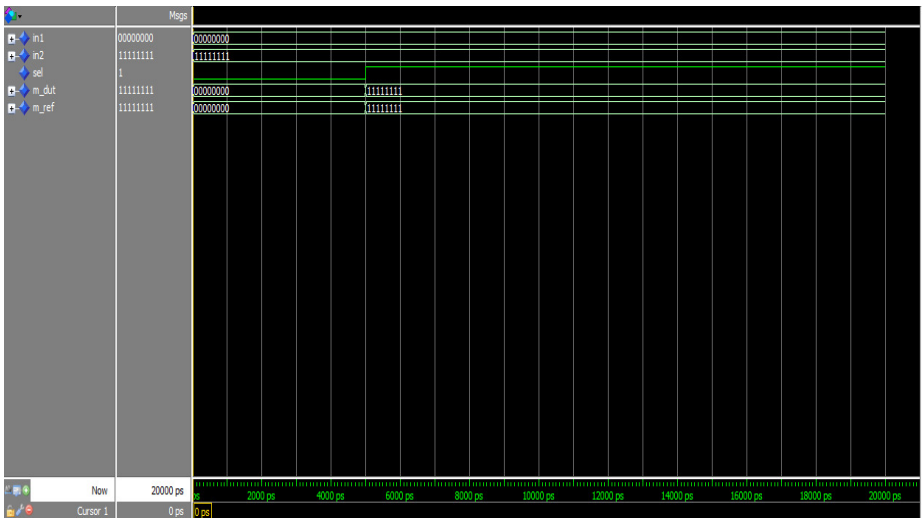


Figura 2.3: waveshot exercise no. 2

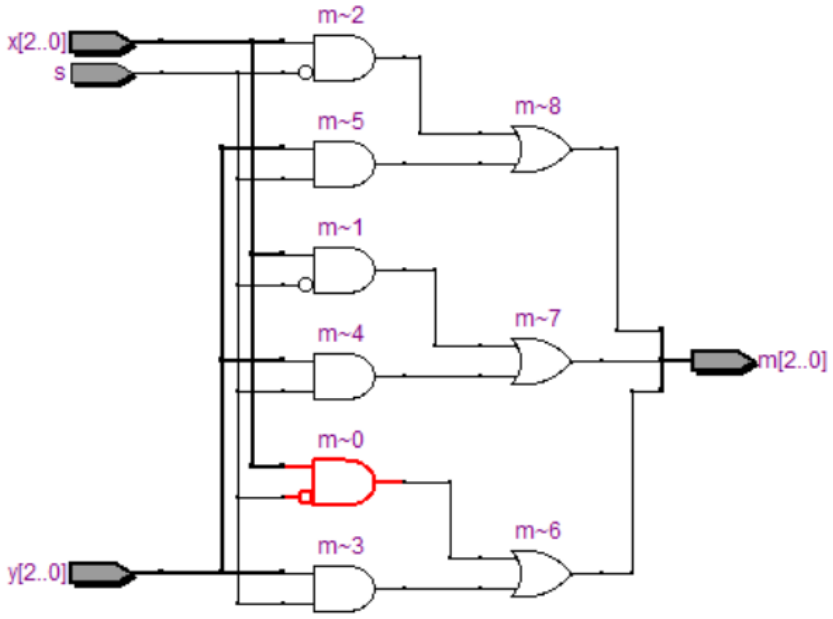


Figura 2.4: synth exercise no. 2

2.4 Appendice

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity mux2to1_gen is
5   generic ( dw : positive := 1 ); -- data width
6   port (
7   x,y : in std_logic_vector ( dw - 1 downto 0 );
8   s : in std_logic;
9   m : out std_logic_vector ( dw -1 downto 0 )
10  );
11  end mux2to1_gen;
12
13
14  architecture logic of mux2to1_gen is
15  signal s_vector : std_logic_vector ( dw -1 downto 0 );
16
17  begin
```

```
18    s_vector <= ( others => s );
19    -- generics-like bit a bit logic
20    m <= (NOT (s_vector) AND x) OR (s_vector AND y);
21    end architecture;
```

Code 2.1: VHDL code exercise no. 2

```
1     library ieee;
2     use ieee.std_logic_1164.all;
3
4     -- empty entity section
5     entity mux2to1_tb is
6     end mux2to1_tb;
7
8     architecture test of mux2to1_tb is
9     constant dw : positive := 8;
10    signal in1 : std_logic_vector ( dw - 1 downto 0);
11    signal in2 : std_logic_vector ( dw - 1 downto 0);
12    signal sel : std_logic;
13    signal m_dut, m_ref : std_logic_vector ( dw - 1 downto 0);
14
15    begin
16    -- example input
17    in1 <= "00000000";
18    in2 <= "11111111";
19
20    sel <= '0', '1' after 5 ns;
21    DUT: entity work.mux2to1_gen generic map (dw)
22    port map( x => in1, y => in2, s => sel, m => m_dut);
23
24    -- expected output
25    with sel select
26    m_ref <=
27    in2 when '1',
28    in1 when others;
29
30
31    end test;
```

Code 2.2: testbench exercise no. 2

```
1     library ieee;
2     use ieee.std_logic_1164.all;
3
4     entity mux2to1_8b_synth is
5     port (
6     SW   : in std_logic_vector ( 17 downto 0 );
7     LEDR : out std_logic;
8     LEDG : out std_logic_vector ( 7 downto 0 )
```

```
9     );
10    end mux2to1_8b_synth;
11
12
13    architecture structure of mux2to1_8b_synth is
14    begin
15
16    MUX: entity work.mux2to1_gen
17    generic map (8)
18    port map( X => SW( 7 downto 0 ),
19    Y => SW( 15 downto 8 ),
20    S => SW(17),
21    M => LEDG
22    );
23    LEDR <= SW(17);
24
25    end structure;
```

Code 2.3: Synch exercise no. 2

Capitolo 3

5-to-1 Multiplexer

3.1 Spiegazione teorica

La terza parte dell'esercitazione ha previsto l'implementazione di un multiplexer a 5 vie di parallelismo 3. Sfruttando i risultati conseguiti nello svolgimento del secondo esercizio, possiamo impiegare quattro multiplexer a due vie di parallelismo 3, passando opportunamente il parametro di tipo generic, come mostrato in **figura 3.1** e in appendice (**codice 3.1**). Si noti che, sempre in ottica di riutilizzabilità, si è implementato parametricamente il parallelismo *dw*.
Il circuito riceve in ingresso i 5 input u, v, w, x, y e i 3 selettori s_0, s_1, s_2. In base alla combinazione dei valori assunti da questi ultimi, l'output assume il valore di uno degli input secondo la tavola della verità. Essendo il numero di input minore di esistono delle combinazioni dei valori di selezione che portano in uscita il medesimo input. In particolare, per tutte le combinazioni tali per cui $s_2 = 1$, si ha y in uscita.

3.2 Procedimento

Le port map collegano, tramite dei segnali, gli input ai MUX di livello più basso, le loro uscite a quelli di livello più alto, insieme ai segnali di selezione, sino a giungere all'output m, come mostrato nello schema seguente.

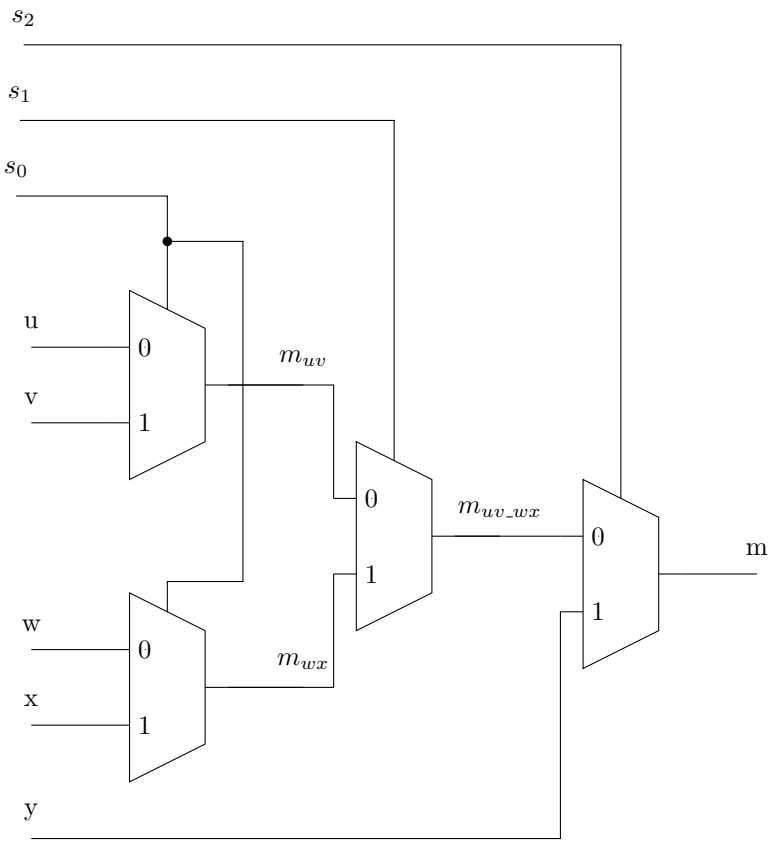


Figura 3.1: 5-to-1 MUX employing four 2-to-1 MUXs

s_0	s_1	s_2	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

Figura 3.2: 5-to-1 MUX table

3.3 Risultati

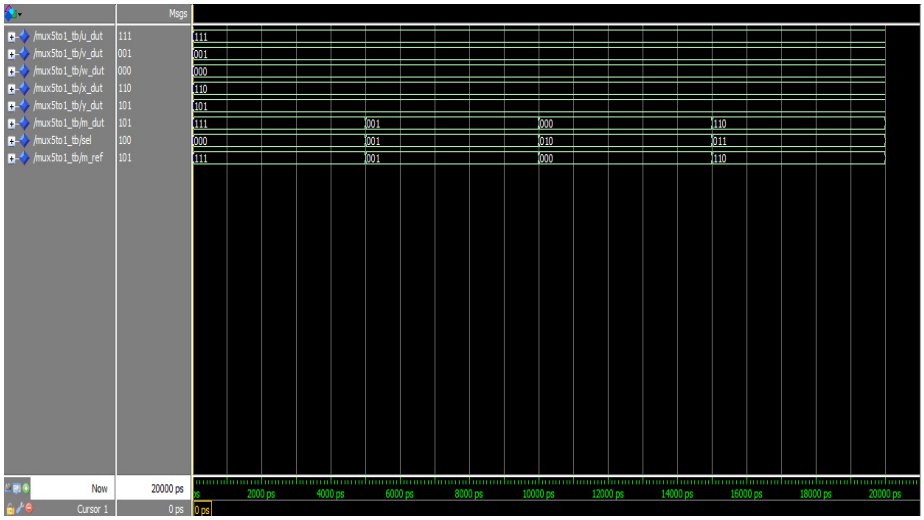


Figura 3.3: waveshot exercise no. 3

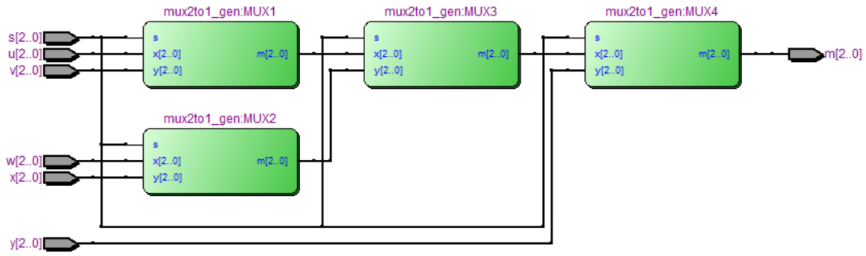


Figura 3.4: RTL exercise no. 3

3.4 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux5to1_gen is
5  generic ( dw : positive := 1);
6  port (
7  u,v,w,x,y : in std_logic_vector (dw - 1 downto 0);
8  s : in std_logic_vector (2 downto 0);
9  m : out std_logic_vector (dw -1 downto 0)
```

```
10 );
11 end mux5to1_gen;
12
13
14 architecture mux_structure of mux5to1_gen is
15 signal m_uv,m_wx,m_uv_wx : std_logic_vector(dw - 1 downto 0);
16
17 begin
18 -- employing generic 2to1mux
19 MUX1 : entity work.mux2to1_gen
20 generic map(dw)
21 port map (u, v, s(0), m_uv);
22
23 MUX2 : entity work.mux2to1_gen
24 generic map(dw)
25 port map (w, x, s(0), m_wx);
26
27 MUX3 : entity work.mux2to1_gen
28 generic map(dw)
29 port map (m_uv, m_wx, s(1), m_uv_wx);
30
31 MUX4 : entity work.mux2to1_gen
32 generic map(dw)
33 port map (m_uv_wx, y, s(2), m);
34 end mux_structure;
```

Code 3.1: VHDL code exercise no. 3

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  -- empty entity section
5  entity mux5to1_tb is
6  end mux5to1_tb;
7
8  architecture Behavioral of mux5to1_tb is
9
10 constant dw : positive := 3;
11 -- testing signals
12
13 signal u_dut, v_dut, w_dut, x_dut, y_dut, m_dut : std_logic_vector ( dw
       - 1 downto 0);
14 signal sel : std_logic_vector (2 downto 0) ;
15 signal m_ref: std_logic_vector ( dw - 1 downto 0); -- reference
       (expected) output
16
17 begin
18
19 -- implicit process testcase
```

```
20  sel <= "000",
21  "001" after 5 ns,
22  "010" after 10 ns,
23  "011" after 15 ns,
24  "100" after 20 ns,
25  "101" after 25 ns,
26  "100" after 30 ns,
27  "111" after 35 ns;
28
29  -- example input
30
31  u_dut <= "111";
32  v_dut <= "001";
33  w_dut <= "000";
34  x_dut <= "110";
35  y_dut <= "101";
36
37  -- testing instance
38  DUT : entity work.mux5to1_gen
39  generic map (dw)
40  port map (u_dut, v_dut, w_dut, x_dut, y_dut, sel, m_dut);
41
42
43  -- expected output def
44
45  with sel select
46  m_ref <= u_dut when "000",
47  v_dut when "001",
48  w_dut when "010",
49  x_dut when "011",
50  y_dut when others;
51
52
53  end architecture;
```

Code 3.2: testbench exercise no. 3

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  --Il programma associa ingressi e uscite del mux5to1 a
5  --periferiche di input e output identificabili della sheda DE2
6
7  entity mux5to1_3b_synth is
8  port (
9  SW : in std_logic_vector( 17 downto 0 ); --SW(0to7)->X, SW(8to15) ->Y,
       SW(17)->S
10 LEDR : out std_logic_vector(2 downto 0); --Permette di vedere il valore
       assunto dal segnale selettore
```

```
11  LEDG : out std_logic_vector( 2 downto 0) --LEDG->M
12  );
13  end mux5to1_3b_synth;
14
15  architecture structure of mux5to1_3b_synth is
16  begin
17  MUX: entity work.mux5to1_gen
18  generic map(3)
19  port map( u => SW( 14 downto 12 ),
20  v => SW( 11 downto 9 ),
21  w => SW( 8 downto 6 ),
22  x => SW( 5 downto 3 ),
23  y => SW( 2 downto 0 ),
24  s => SW( 17 downto 15 ),
25  M => LEDG
26  );
27
28
29  LEDR <= SW (17 downto 15); -- led to selec
30  end structure;
```

Code 3.3: Synth code exercise no. 3