

– Elettronica dei Sistemi Digitali –

Lab#2

Switches, Decoders, Numbers and Displays

Gruppo: D07

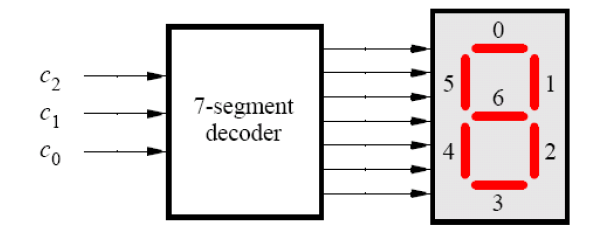
**1 – Controlling a 7-segments display**

Spiegazione teorica

Si vogliono utilizzare tre *switches* presenti sulla scheda DE2 () come input per pilotare il display a 7 segmenti e scrivere la parola “HELLO”.

Procedimento

Per il progetto, si parte da un approccio blackbox: il display è pilotato da un decoder con 3 bit di ingresso e 7 bit di uscita.



**Figura 1:** definizione della entity del decoder utilizzato

Si procede con la definizione delle funzioni logiche necessarie per la realizzazione del progetto. Nella seguente tabella, ciascun segmento è indicato per comodità con la notazione , dove *i* è il numero identificato in **Figura 1**.

|  |  |  |
| --- | --- | --- |
|  | Lettera |  |
| 000 | H | 1001000 |
| 001 | E | 0110000 |
| 010 | L | 1110001 |
| 011 | O | 0000001 |
| 100 |  | 1111111 |
| 101 |  | 1111111 |
| 110 |  | 1111111 |
| 111 |  | 1111111 |

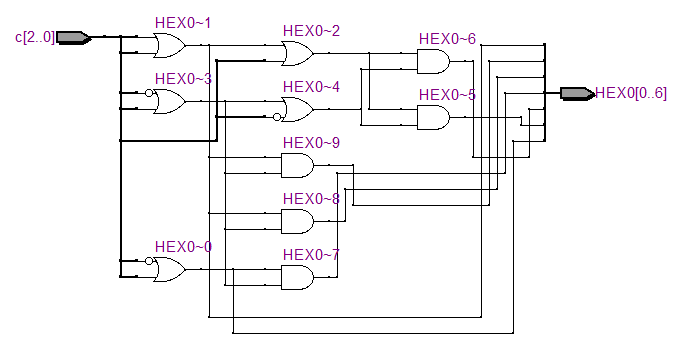
Poiché i segmenti luminosi si accendono in corrispondenza del valore logico ‘0’, le funzioni logiche vengono ricavate a partire dalle mappe di Karnaugh in forma POS (Product of Sum).

MAPPE DI KAZZAUGH

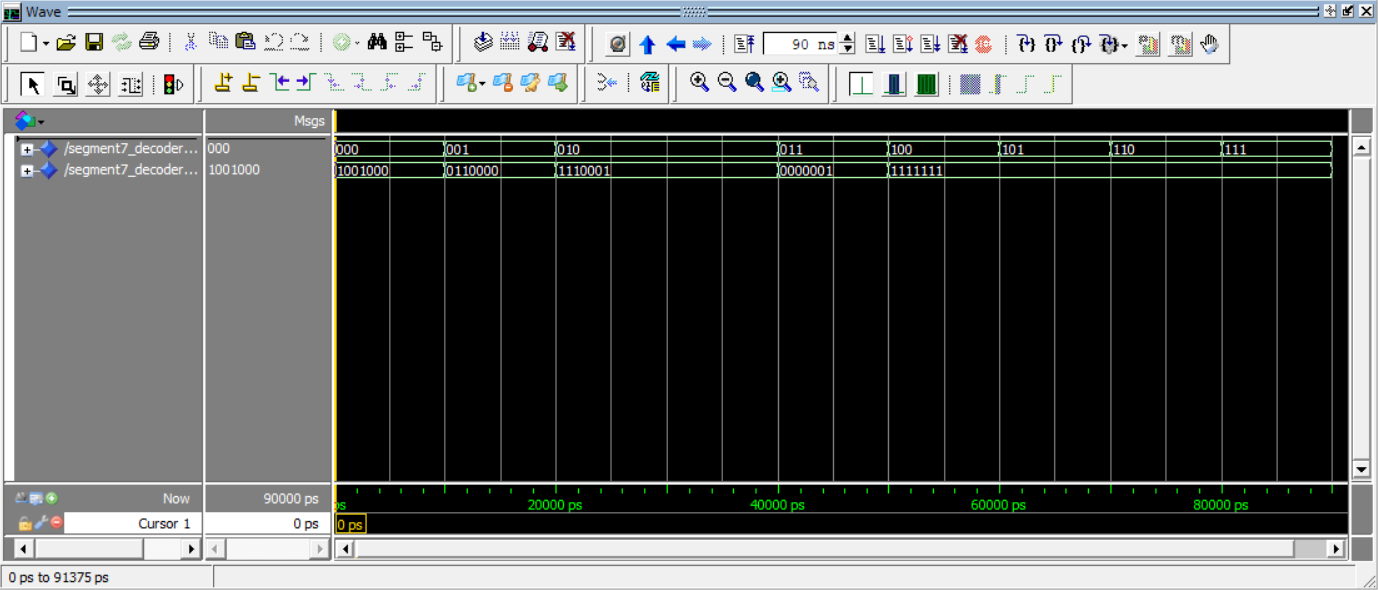
Una volta definite le funzioni logiche desiderate, nel file *segment7\_decoder.vhd* viene descritto il decoder utilizzato per controllare i 7 segmenti luminosi del display, facendo attenzione anche ad assegnare correttamente i pin. Per testare il funzionamento in assenza della scheda DE2, viene creato il file *segment7\_decoder\_tb.vhd* per effettuare una simulazione su ModelSim.

Risultati

Vengono riportati nelle figure seguenti i risultati ottenuti con Quartus II e ModelSim: la simulazione effettuata mostra che è possibile scrivere la parola “HELLO” con il display a 7 segmenti, un carattere per volta; alle combinazioni degli ingressi diverse dalle quattro necessarie per i singoli caratteri corrisponde il display spento.



**Figura 2:** vista RTL del decoder



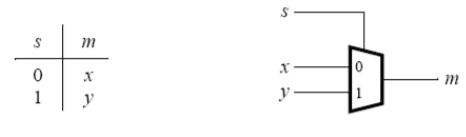
**Figura 3:** output della simulazione effettuata con Modelsim

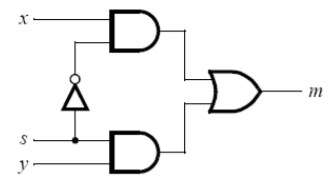
**2 - Multiplexing the 7-segments display output**

Spiegazione teorica

lll

Procedimento





**Figura 4:** schema a blocchi del MUX 2-to-1 al livello gate (SOP)

Risultati

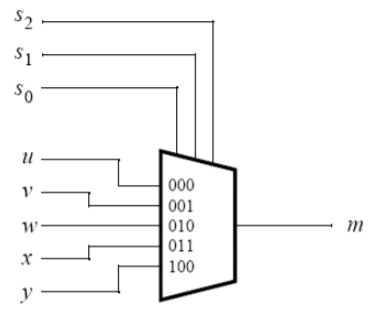
**3 – 5-to-1 Multiplexer**

Spiegazione teorica

In questo caso, l’uscita *m* del circuito deve essere scelta tra 5 possibili ingressi, indicati con *u*, *v*, *w*, *x*, e *y*. Per questo motivo, sono necessari 3 bit di selezione.

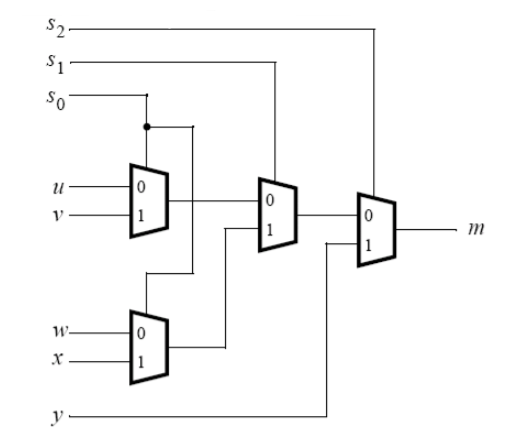
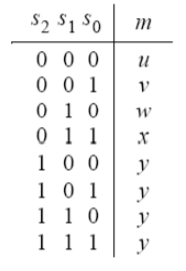
Procedimento

La *entity* che si vuole realizzare è un MUX 5-to-1, rappresentato a livello *blackbox* come segue. A seconda del numero di selezione, rappresentato come detto su 3 bit, l’uscita assumerà uno dei valori di ingresso.



**Figura 11:** simbolo del MUX 5-to-1

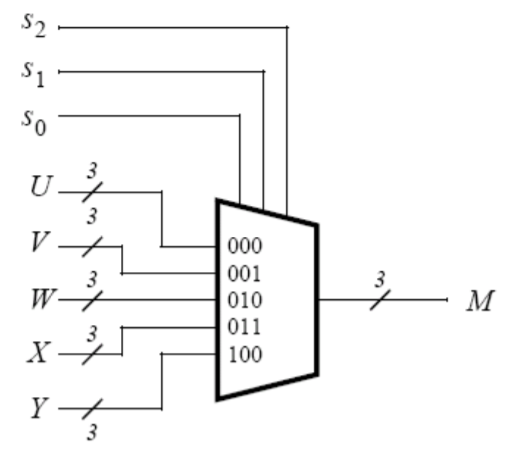
Il “blocco elementare” per la realizzazione del circuito richiesto è nuovamente il MUX 2-to-1 presentato in precedenza. Con il consueto approccio, è possibile rappresentare la funzione logica del circuito e il suo schema a blocchi.



**Figura 12:** tavola di verità e schema a blocchi

Il MUX 5-to-1 è descritto nel file *MUX5to1.vhd* e viene controllato il suo funzionamento con il relativo testbench (*MUX5to1\_tb.vhd*). Poiché viene richiesto un parallelismo su 3 bit per ingressi e uscite, anche in questo caso si procede mettendo in parallelo tre MUX 5-to-1, collegati allo stesso segnale di selezione *s* (sempre su 3 bit): nella *entity* riportata, il parallelismo è indicato come di consueto semplicemente aggiungendo una linea sugli ingressi e sull’uscita, specificando il numero di bit.

Si è scelto anche in questo caso di utilizzare il nome MUX 15-to-3 allo scopo di differenziare la notazione; descrizione in VHDL e relativo testbench sono nei files *MUX15to3.vhd* e *MUX15to3\_tb.vhd*.



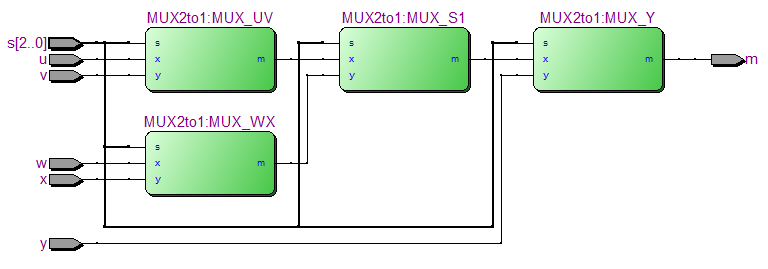
**Figura 13:** simbolo del MUX 5-to-1 su 3 bit

Risultati

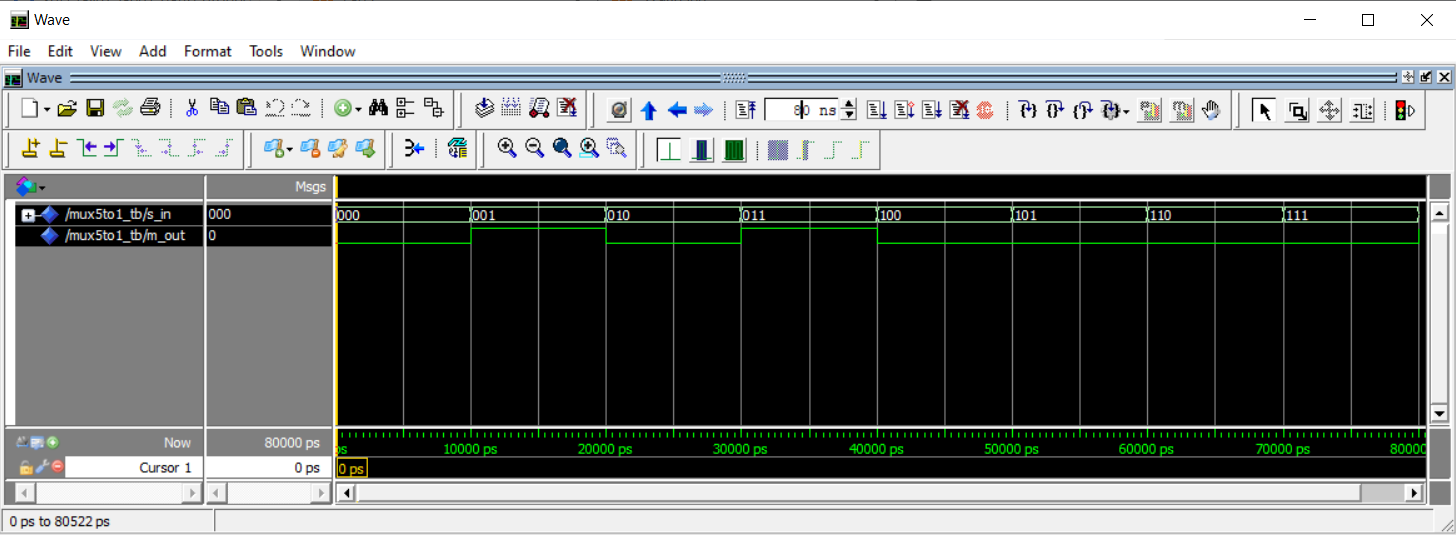
Si riportano vista RTL e risultati della simulazione con Modelsim dei due circuiti descritti nella sezione precedente. Per quanto riguarda la simulazione, nella vista Wave di Modelsim non sono stati riportati i valori degli ingressi, impostati come costanti per provare un approccio differente dal precedente. I valori scelti per la simulazione sono:

* *u* = 0, che compare in uscita quando *s* = “000”;
* *v* = 1, che compare in uscita quando *s* = “001”;
* *w* = 0, che compare in uscita quando *s* = “010”;
* *x* = 1, che compare in uscita quando *s* = “011”;
* *y* = 0, che compare in uscita per gli altri valori di *s*.

Il valore di *s* cambia ogni 10 ns, dunque si è scelto di simulare per 80 ns in modo da controllare tutte le combinazioni.



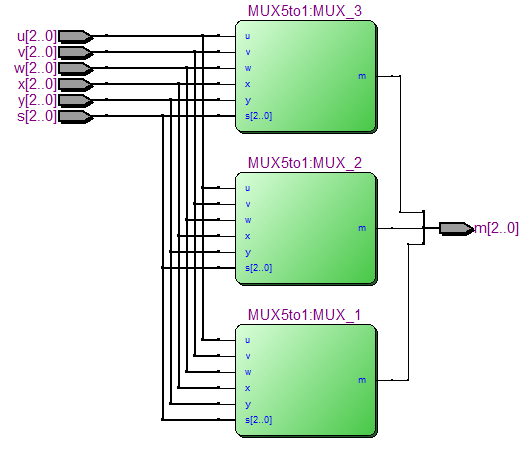
**Figura 14:** RTL view del MUX 5-to-1



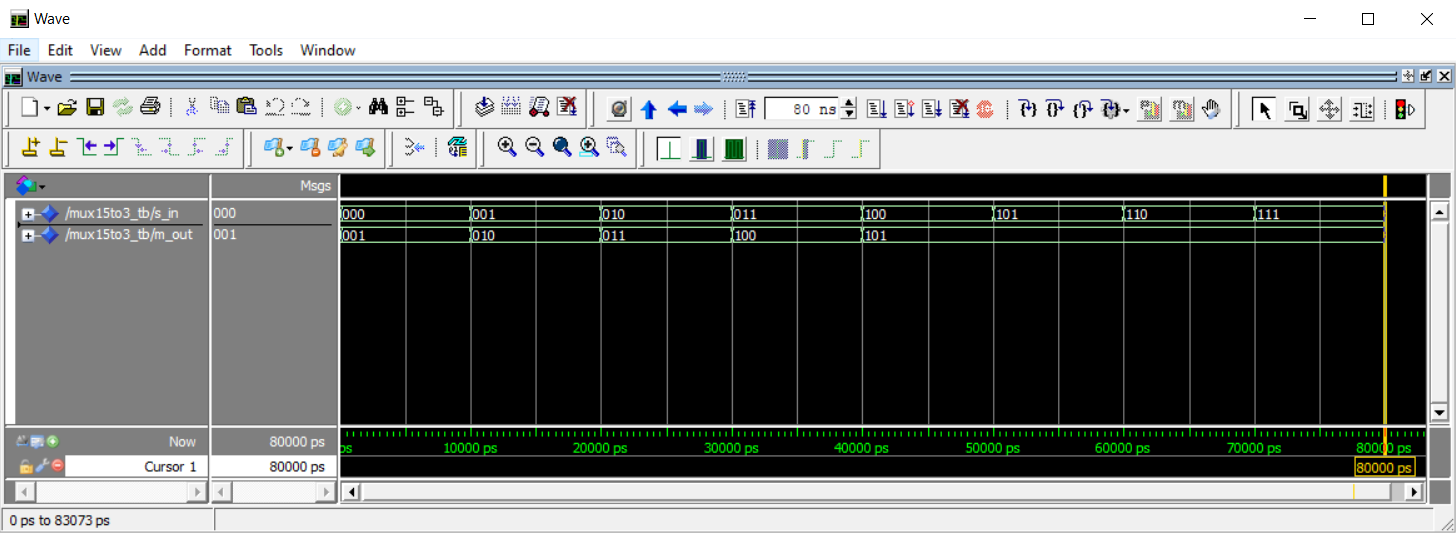
**Figura 15:** simulazione effettuata con Modelsim

Nel caso di MUX 5-to-1 con parallelismo a 3 bit (indicato come MUX 15-to-3) si impostano per la simulazione i seguenti valori dei segnali di ingresso:

* *u* = “001”, che compare in uscita quando *s* = “000”;
* *v* = “010”, che compare in uscita quando *s* = “001”;
* *w* = “011”, che compare in uscita quando *s* = “010”;
* *x* = “100”, che compare in uscita quando *s* = “011”;
* *y* = “101”, che compare in uscita per gli altri valori di *s*.



**Figura 16:** RTL view del MUX 15-to-3



**Figura 17:** simulazione effettuata con Modelsim

**Appendice**

**1 – Controlling a 7-segments display**

Codice

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity segment7\_decoder is  port(  c: in std\_logic\_vector(2 downto 0); --inputs  HEX0: out std\_logic\_vector(0 to 6) --segments  ); end segment7\_decoder;  architecture Behavior of segment7\_decoder is begin  HEX0(0) <= c(2) OR (NOT c(0));  HEX0(1) <= (c(2) OR c(1) OR c(0)) AND (c(2) OR (NOT c(1)) OR (NOT c(0)));  HEX0(2) <= (c(2) OR c(1) OR c(0)) AND (c(2) OR (NOT c(1)) OR (NOT c(0)));  HEX0(3) <= (c(2) OR ( NOT c(0))) AND ( c(2) OR ( NOT c(1)));  HEX0(4) <= (c(2) OR c(1)) AND (c(2) OR (NOT c(1)));  HEX0(5) <= (c(2) OR c(1)) AND (c(2) OR (NOT c(1)));  HEX0(6) <= c(2) OR c(1); end Behavior; |

Codice VHDL per decoder display a 7 segmenti

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity segment7\_decoder\_tb is end segment7\_decoder\_tb;  architecture Behavior of segment7\_decoder\_tb is  component segment7\_decoder  port(  c: in std\_logic\_vector(2 downto 0);  segs: out std\_logic\_vector(0 to 6)  ); end component;  signal c\_in: std\_logic\_vector(2 downto 0) := "111"; --all segs turned off signal segs\_out: std\_logic\_vector(0 to 6); begin  DUT : segment7\_decoder port map (c => c\_in, segs => segs\_out); process begin  c\_in <= "000"; -- "H => 1001000"  wait for 10 ns;  c\_in <= "001"; -- "E => 0110000"  wait for 10 ns;  c\_in <= "010"; -- "L => 1110001"  wait for 10 ns;  c\_in <= "010"; -- "L => 1110001"  wait for 10 ns;  c\_in <= "011"; -- "O => 0000001"  wait for 10 ns;  c\_in <= "100"; -- "blank from now on => 1111111"  wait for 10 ns;  c\_in <= "101";  wait for 10 ns;  c\_in <= "110";  wait for 10 ns;  c\_in <= "111";  wait for 10 ns; end process;   end Behavior; |

Testbench per il decoder

**2 - 2-to-1 Multiplexer**

Codice

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX2to1 is  port( x, y: in std\_logic; --inputs  s: in std\_logic; --selector  m: out std\_logic --output  ); end MUX2to1;  architecture Behavior of MUX2to1 is begin  m <= (NOT (s) AND x) OR (s AND y); end Behavior; |

Codice VHDL per MUX 2-to-1

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX2to1\_tb is end MUX2to1\_tb;  architecture Behavior of MUX2to1\_tb is  component MUX2to1  port(  x, y: in std\_logic; --inputs  s: in std\_logic; --selector  m: out std\_logic --output  ); end component;  signal x\_in: std\_logic; signal y\_in: std\_logic; signal s\_in, m\_out: std\_logic;  begin  x\_in <= '1';  y\_in <= '0'; --constant inputs in order to verify the behaviour of the circuit, emulating the switches  DUT: MUX2to1 port map (x => x\_in, y=> y\_in, s => s\_in, m => m\_out);   process begin  s\_in <= '0'; -- if s=0 then m=x that equals to 1 (m = 1)  wait for 4 ns;  s\_in <= '1'; -- if s=1 then m=y that equals to 0 (m = 0)  wait for 4 ns; end process; end Behavior; |

Testbench del MUX 2-to-1

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX16to8 is  port(x, y: in std\_logic\_vector(7 downto 0); --inputs  s: in std\_logic; --selectors  m: out std\_logic\_vector(7 downto 0) --outputs  ); end MUX16to8;  architecture Structure of MUX16to8 is  component MUX2to1  port(x, y: in std\_logic; --inputs  s: in std\_logic; --selector  m: out std\_logic --output  ); end component;  begin  GEN: for i in 7 downto 0 generate  MUX: MUX2to1 port map (x => x(i), y => y(i), s => s, m => m(i));  end generate GEN; end Structure; |

Codice VHDL per MUX 2-to-1 a 8 bit

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX16to8\_tb is end MUX16to8\_tb;  architecture Behavior of MUX16to8\_tb is  component MUX16to8  port(x, y: in std\_logic\_vector(7 downto 0); --inputs  s: in std\_logic; --selectors  m: out std\_logic\_vector(7 downto 0) --outputs  ); end component;  signal X\_in: std\_logic\_vector(7 downto 0); signal Y\_in: std\_logic\_vector(7 downto 0); signal s\_in: std\_logic; signal M\_out: std\_logic\_vector(7 downto 0);  begin  X\_in <= "11111111";  Y\_in <= "00000000";   DUT: MUX16to8 port map (x => X\_in, y=> Y\_in, s => s\_in, m(7 downto 0) => M\_out(7 downto 0));  process begin  s\_in <= '0'; -- if s=0 then M=X  wait for 4 ns;  s\_in <= '1'; -- if s=0 then M=Y  wait for 4 ns; end process; end Behavior; |

Testbench del MUX 2-to-1 a 8 bit

**3 – 5-to-1 Multiplexer**

Codice

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX5to1 is  port(  u, v, w, x, y: in std\_logic; --inputs  s: in std\_logic\_vector(2 downto 0); --selectors  m: out std\_logic --output  ); end MUX5to1;  architecture Structure of MUX5to1 is  component MUX2to1  port(  x, y: in std\_logic; --inputs  s: in std\_logic; --selector  m: out std\_logic --output  ); end component;  signal mth\_output: std\_logic\_vector(2 downto 0); begin  MUX\_UV: MUX2to1 port map (x => u, y => v, s => s(0), m => mth\_output(0));  MUX\_WX: MUX2to1 port map (x => w, y => x, s => s(0), m => mth\_output(1));  MUX\_S1: MUX2to1 port map (x => mth\_output(0), y => mth\_output(1), s => s(1), m => mth\_output(2));  MUX\_Y: MUX2to1 port map (x => mth\_output(2), y => y, s => s(2), m => m); end Structure; |

Codice VHDL per MUX 5-to-1

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX5to1\_tb is end MUX5to1\_tb;  architecture Behavior of MUX5to1\_tb is  component MUX5to1  port(  u, v, w, x, y: in std\_logic; --inputs  s: in std\_logic\_vector(2 downto 0); --selectors  m: out std\_logic --output  ); end component;  constant v\_in, x\_in: std\_logic := '1'; -- we set constants only for demonstration purposes: u,v,w,x,y constant u\_in, w\_in, y\_in: std\_logic:= '0'; -- could be any input value in the circuit signal s\_in: std\_logic\_vector(2 downto 0); signal m\_out: std\_logic;  begin  DUT: MUX5to1 port map (  u => u\_in, v => v\_in, w => w\_in, x => x\_in, y=> y\_in,  s => s\_in, m => m\_out  );   process begin  -- if s=0 then m=x that equals to 1 (m = 1)  -- if s=1 then m=y that equals to 0 (m = 0)    s\_in <= "000"; -- u (m=0)  wait for 10 ns;  s\_in <= "001"; -- v (m=1)  wait for 10 ns;   s\_in <= "010"; -- w (m=0)  wait for 10 ns;  s\_in <= "011"; -- x (m=1)  wait for 10 ns;  s\_in <= "100"; -- y (m=0)  wait for 10 ns;  s\_in <= "101"; -- y (m=0)  wait for 10 ns;  s\_in <= "110"; -- y (m=0)  wait for 10 ns;  s\_in <= "111"; -- y (m=0)  wait for 10 ns; end process; end Behavior; |

Testbench del MUX 5-to-1

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX15to3 is  port(u, v, w, x, y: in std\_logic\_vector(2 downto 0); --inputs  s: in std\_logic\_vector(2 downto 0); --selectors  m: out std\_logic\_vector(2 downto 0) --output  ); end MUX15to3;  architecture Structure of MUX15to3 is  component MUX5to1  port(u, v, w, x, y: in std\_logic; --inputs  s: in std\_logic\_vector(2 downto 0); --selectors  m: out std\_logic --output  ); end component;  begin  MUX\_1: MUX5to1 port map (u => u(0), v => v(0), w => w(0), x => x(0), y => y(0), s => s, m => m(0));  MUX\_2: MUX5to1 port map (u => u(1), v => v(1), w => w(1), x => x(1), y => y(1), s => s, m => m(1));  MUX\_3: MUX5to1 port map (u => u(2), v => v(2), w => w(2), x => x(2), y => y(2), s => s, m => m(2)); end Structure; |

Codice VHDL per MUX 5-to-1 a 3 bit

|  |
| --- |
| library ieee; use ieee.std\_logic\_1164.all;  entity MUX15to3\_tb is end MUX15to3\_tb;  architecture Behavior of MUX15to3\_tb is  component MUX15to3  port(  u, v, w, x, y: in std\_logic\_vector(2 downto 0); --inputs  s: in std\_logic\_vector(2 downto 0); --selectors  m: out std\_logic\_vector(2 downto 0) --output  ); end component;  -- we count from 1 to 5 constant u\_in: std\_logic\_vector(2 downto 0) := "001"; -- number 1 constant v\_in: std\_logic\_vector(2 downto 0) := "010"; -- number 2 constant w\_in: std\_logic\_vector(2 downto 0) := "011"; -- number 3 constant x\_in: std\_logic\_vector(2 downto 0) := "100"; -- number 4 constant y\_in: std\_logic\_vector(2 downto 0) := "101"; -- number 5  signal s\_in: std\_logic\_vector(2 downto 0); signal m\_out: std\_logic\_vector(2 downto 0);  begin  DUT: MUX15to3 port map (  u => u\_in, v => v\_in, w => w\_in, x => x\_in, y=> y\_in,  s => s\_in, m => m\_out  );   process begin  s\_in <= "000"; -- u (m=001 => "1")  wait for 10 ns;  s\_in <= "001"; -- v (m=010 => "2")  wait for 10 ns;   s\_in <= "010"; -- w (m=011 => "3")  wait for 10 ns;  s\_in <= "011"; -- x (m=100 => "4")  wait for 10 ns;  s\_in <= "100"; -- y (m=101 => "5")  wait for 10 ns;  s\_in <= "101"; -- y (m=101 => "5")  wait for 10 ns;  s\_in <= "110"; -- y (m=101 => "5")  wait for 10 ns;  s\_in <= "111"; -- y (m=101 => "5")  wait for 10 ns; end process; end Behavior; |

Testbench MUX 5-to-1 a 3 bit