Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell’Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Immagine che contiene cerchio, simbolo, schizzo

Descrizione generata automaticamente

Elaborato di Architettura dei Sistemi Digitali

*Prof.ssa Alessandra De Benedictis*

a.a. 2023-24

Studenti:

Boccarossa Antonio M63001643

Brunello Francesco M63001655

Cangiano Salvatore M63001647

Sommario

[Capitolo 1: Reti Combinatorie Elementari 4](#_Toc157196867)

[Esercizio 1.1: Multiplexer 16:1 4](#_Toc157196868)

[Progetto e architettura 4](#_Toc157196869)

[Implementazione 4](#_Toc157196870)

[Simulazione 6](#_Toc157196871)

[Esercizio 1.2: Rete di interconnessione 16:4 7](#_Toc157196872)

[Progetto e architettura 7](#_Toc157196873)

[Implementazione 8](#_Toc157196874)

[Simulazione 9](#_Toc157196875)

[Esercizio 1.3: Implementazione su board dell’esercizio 1.2 10](#_Toc157196876)

[Progetto e architettura 10](#_Toc157196877)

[Implementazione 10](#_Toc157196878)

[Simulazione 10](#_Toc157196879)

[Esercizio 2.1: ROM combinatoria con sistema M. 11](#_Toc157196880)

[Progetto e architettura 11](#_Toc157196881)

[Implementazione 11](#_Toc157196882)

[Simulazione 13](#_Toc157196883)

[Esercizio 2.2: Implementazione su board della ROM combinatoria con sistema M. 14](#_Toc157196884)

[Progetto e architettura 14](#_Toc157196885)

[Implementazione 14](#_Toc157196886)

[Simulazione 14](#_Toc157196887)

[Capitolo 2: Reti Sequenziali Elementari 14](#_Toc157196888)

[Esercizio 3.1: Riconoscitore di sequenza 101 (senza sovrapposizione/sovrapposizione parziale). 14](#_Toc157196889)

[Progetto e architettura 15](#_Toc157196890)

[Implementazione 16](#_Toc157196891)

[Simulazione 18](#_Toc157196892)

[Esercizio 3.2: Implementazione su board del riconoscitore di sequenza. 19](#_Toc157196893)

[Progetto e architettura 19](#_Toc157196894)

[Implementazione 19](#_Toc157196895)

[Simulazione 19](#_Toc157196896)

[Esercizio 4.1: Shift Register. 19](#_Toc157196897)

[Progetto e architettura 19](#_Toc157196898)

[Implementazione 20](#_Toc157196899)

[Simulazione 20](#_Toc157196900)

[Appendice 22](#_Toc157196901)

[Multiplexer 2:1 22](#_Toc157196902)

[Multiplexer 4:1 22](#_Toc157196903)

[Demultiplexer 1:4 22](#_Toc157196904)

[ROM da 16 locazioni da 8 bit 23](#_Toc157196905)

[Progetto e architettura 24](#_Toc157196906)

[Implementazione 24](#_Toc157196907)

# Capitolo 1: Reti Combinatorie Elementari

## Esercizio 1.1: Multiplexer 16:1

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.**

#### Progetto e architettura

L’esercizio richiede di implementare un multiplexer **16:1 indirizzabile**, l’approccio utilizzato è quello per composizione tramite [*multiplexer 4:1*](#_Multiplexer_4:1)*.*

Immagine che contiene diagramma, testo, Piano, linea

Descrizione generata automaticamenteIl multiplexer **16:1 è composto da 16 ingressi e 4 bit di selezione.** Per il design strutturale adottato, per il corretto funzionamento del multiplexer 16:1, si sono adottati *5 multiplexer 4:1 posti su due livelli:*

* Il **primo livello** è formato da 4 multiplexer, che accolgono gli ingressi a gruppi di 4; i bit di selezione sono i due bit meno significativi di *control\_16\_1.*
* Il **secondo livello** è composto da un solo multiplexer, che accoglie come ingressi le 4 uscite dei multiplexer del primo livello e viene controllato dai due bit più significativi di *control\_16\_1.*

#### Implementazione

Di seguito, è riportata l’implementazione del multiplexer 16:1.

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6. entity mux\_16\_1 is

7. port (

8. input\_16\_1: in STD\_LOGIC\_VECTOR (0 to 15);

9.

10. control\_16\_1: in STD\_LOGIC\_VECTOR (0 to 3);

11.

12. y\_16\_1: out STD\_LOGIC

13.

14. );

15. end entity mux\_16\_1;

16.

17.

18. architecture structural of mux\_16\_1 is

19.

20.

21.

22. signal outputs: STD\_LOGIC\_VECTOR (0 to 3) := (others => '0');

23. --uscite dei 4 multiplexer da interconnettere

24.

25. component mux\_4\_1 port(

26.

27. input\_4\_1 : in STD\_LOGIC\_VECTOR (0 to 3);

28.

29. control\_4\_1 : in STD\_LOGIC\_VECTOR (0 to 1);

30.

31. y\_4\_1 : out STD\_LOGIC

32.

33. );

34. end component;

35.

36. begin

37.

38. mux0: mux\_4\_1

39. Port map (

40. input\_4\_1(0) => input\_16\_1(0),

41. input\_4\_1(1) => input\_16\_1(1),

42. input\_4\_1(2) => input\_16\_1(2),

43. input\_4\_1(3) => input\_16\_1(3),

44.

45. control\_4\_1(0) => control\_16\_1(0),

46. control\_4\_1(1) => control\_16\_1(1),

47.

48.

49. y\_4\_1 => outputs(0)

50. );

51.

52. mux1: mux\_4\_1

53. Port map (

54. input\_4\_1(0) => input\_16\_1(4),

55. input\_4\_1(1) => input\_16\_1(5),

56. input\_4\_1(2) => input\_16\_1(6),

57. input\_4\_1(3) => input\_16\_1(7),

58.

59. control\_4\_1(0) => control\_16\_1(0),

60. control\_4\_1(1) => control\_16\_1(1),

61.

62. y\_4\_1 => outputs(1)

63. );

64.

65.

66. mux2: mux\_4\_1

67. Port map (

68. input\_4\_1(0) => input\_16\_1(8),

69. input\_4\_1(1) => input\_16\_1(9),

70. input\_4\_1(2) => input\_16\_1(10),

71. input\_4\_1(3) => input\_16\_1(11),

72.

73. control\_4\_1(0) => control\_16\_1(0),

74. control\_4\_1(1) => control\_16\_1(1),

75.

76. y\_4\_1 => outputs(2)

77. );

78.

79. mux3: mux\_4\_1

80. Port map (

81. input\_4\_1(0) => input\_16\_1(12),

82. input\_4\_1(1) => input\_16\_1(13),

83. input\_4\_1(2) => input\_16\_1(14),

84. input\_4\_1(3) => input\_16\_1(15),

85.

86.

87. control\_4\_1(0) => control\_16\_1(0),

88. control\_4\_1(1) => control\_16\_1(1),

89.

90. y\_4\_1 => outputs(3)

91.

92. );

93.

94.

95. mux4: mux\_4\_1

96. Port map (

97. input\_4\_1(0) => outputs(0),

98. input\_4\_1(1) => outputs(1),

99. input\_4\_1(2) => outputs(2),

100. input\_4\_1(3) => outputs(3),

101.

102.

103. control\_4\_1(0) => control\_16\_1(2),

104. control\_4\_1(1) => control\_16\_1(3),

105.

106. y\_4\_1 => y\_16\_1

107.

108. );

109.

110.

111.

112.

113.

114. end architecture structural;

115.

#### Simulazione

Di seguito è riportato il testbench utilizzato per effettuare i test sul componente.

1. library ieee;

2. use ieee.std\_logic\_1164.all;

3.

4. entity testbench is

5. end testbench;

6.

7. architecture tb of testbench is

8.

9. component mux\_16\_1

10. port (input\_16\_1 : in std\_logic\_vector (0 to 15);

11. control\_16\_1 : in std\_logic\_vector (0 to 3);

12. y\_16\_1 : out std\_logic);

13. end component;

14.

15. signal input\_16\_1 : std\_logic\_vector (0 to 15);

16. signal control\_16\_1 : std\_logic\_vector (0 to 3);

17. signal y\_16\_1 : std\_logic;

18.

19. begin

20.

21. dut : mux\_16\_1

22. port map (input\_16\_1 => input\_16\_1,

23. control\_16\_1 => control\_16\_1,

24. y\_16\_1 => y\_16\_1);

25.

26. stimuli : process

27. begin

28.

29. input\_16\_1 <= (others => '0');

30. control\_16\_1 <= (others => '0');

31.

32. wait for 10 ns;

33. input\_16\_1 <= "1000000000000000";

34. wait for 100 ns;

35.

36.

37. input\_16\_1 <= "0000000100000000";

38. control\_16\_1 <= "1110";

39.

40.

41. wait for 100 ns;

42.

43.

44. input\_16\_1 <= "0000000000000001";

45. control\_16\_1 <= "1111";

46.

47.

48. wait for 100 ns;

49. wait;

50. end process;

51.

52. end tb;

53.

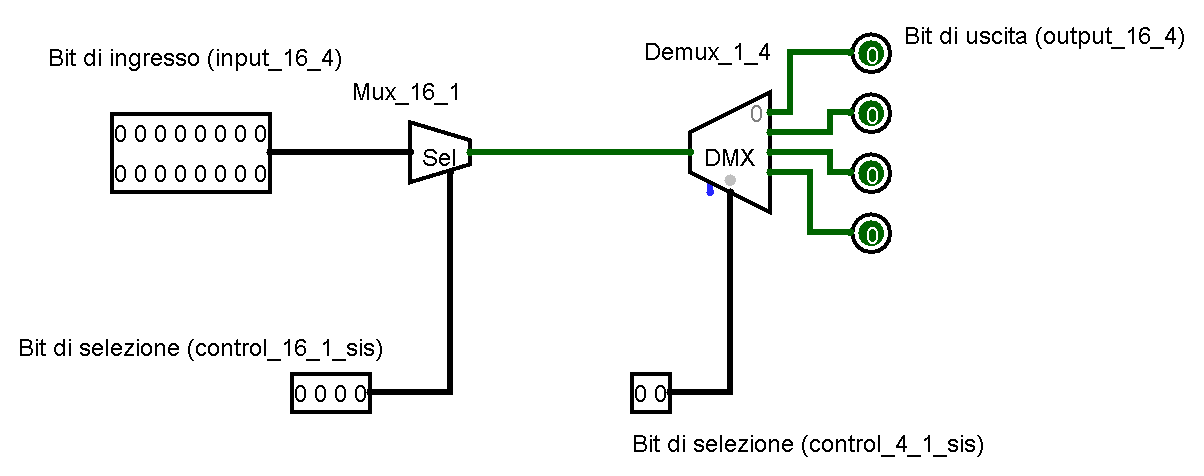
Nella sezione *architecture* del testbench viene definito dapprima il componente utilizzato con la definizione dei vari signal, poi viene istanziato il multiplexer mappando le corrispettive porte. Di seguito, tramite un process vengono simulati vari casi per asserire il funzionamento del componente.

## Esercizio 1.2: Rete di interconnessione 16:4

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.

#### Progetto e architettura

L’esercizio ci chiede di implementare, utilizzando il **multiplexer 16:1**, implementato nel punto precedente, una semplice rete di interconnessione formata da *16 ingressi e 4 destinazioni.* Come ulteriore componente, è stato aggiunto un [**Demultiplexer 1:4**](#_Demultiplexer_1:4).



Il top-module *Sistema,* contiene il **multiplexer 16:1 definito precedentemente,** in più abbiamo un **demux 1:4** di seguito, per permettere la comunicazione su 4 canali. Inoltre, sono presenti in totale 6 bit di selezione suddivisi in due vettori, *control\_16\_1\_sis* per controllare il multiplexer 16:1, e *control\_4\_1\_sis* per controllare il *demultiplexer 1:4.*

#### Implementazione

Di seguito, abbiamo l’implementazione del top-module *Sistema.* (Il demultiplexer è presente in **Appendice**.)

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6. entity Sistema is

7. port (

8. input\_16\_4 : in std\_logic\_vector(0 to 15);

9.

10. output\_16\_4: out std\_logic\_vector(0 to 3);

11.

12. control\_16\_1\_sis: in std\_logic\_vector(0 to 3);

13.

14. control\_1\_4\_sis: in std\_logic\_vector(0 to 1)

15.

16. );

17. end entity Sistema;

18.

19. architecture structural of Sistema is

20.

21. signal output : std\_logic;

22.

23. component mux\_16\_1 port(

24. input\_16\_1: in STD\_LOGIC\_VECTOR (0 to 15);

25.

26. control\_16\_1: in STD\_LOGIC\_VECTOR (0 to 3);

27.

28. y\_16\_1: out STD\_LOGIC

29.

30. );

31. end component;

32.

33. component Demux\_1\_4 port(

34. input\_1\_4: in STD\_LOGIC;

35.

36. output\_1\_4: out std\_logic\_vector(0 to 3);

37.

38. control\_1\_4 : in std\_logic\_vector(0 to 1)

39.

40. );

41. end component;

42.

43. begin

44.

45.

46.

47. mux: mux\_16\_1 port map(

48. input\_16\_1(0) => input\_16\_4(0),

49. input\_16\_1(1) => input\_16\_4(1),

50. input\_16\_1(2) => input\_16\_4(2),

51. input\_16\_1(3) => input\_16\_4(3),

52. input\_16\_1(4) => input\_16\_4(4),

53. input\_16\_1(5) => input\_16\_4(5),

54. input\_16\_1(6) => input\_16\_4(6),

55. input\_16\_1(7) => input\_16\_4(7),

56. input\_16\_1(8) => input\_16\_4(8),

57. input\_16\_1(9) => input\_16\_4(9),

58. input\_16\_1(10) => input\_16\_4(10),

59. input\_16\_1(11) => input\_16\_4(11),

60. input\_16\_1(12) => input\_16\_4(12),

61. input\_16\_1(13) => input\_16\_4(13),

62. input\_16\_1(14) => input\_16\_4(14),

63. input\_16\_1(15) => input\_16\_4(15),

64.

65.

66. --output e controllo

67.

68. y\_16\_1 => output,

69.

70. control\_16\_1(0)=> control\_16\_1\_sis(0),

71. control\_16\_1(1)=> control\_16\_1\_sis(1),

72. control\_16\_1(2)=> control\_16\_1\_sis(2),

73. control\_16\_1(3)=> control\_16\_1\_sis(3)

74.

75. );

76.

77. demux: Demux\_1\_4 port map(

78.

79. input\_1\_4 => output,

80.

81.

82. output\_1\_4(0)=> output\_16\_4(0),

83. output\_1\_4(1) => output\_16\_4(1),

84. output\_1\_4(2) => output\_16\_4(2),

85. output\_1\_4(3) => output\_16\_4(3),

86.

87.

88. control\_1\_4(0) => control\_1\_4\_sis(0),

89. control\_1\_4(1) => control\_1\_4\_sis(1)

90.

91. );

92.

93.

94.

95.

96. end architecture structural;

97.

La definizione del top-module *Sistema* è formata principalmente dalla definizione e istanziazione del **multiplexer 16:1 e del multiplexer 1:4,** con la definizione del singolo*signal* ***output*** per legare le due componenti.

#### Simulazione

Per la simulazione è stato utilizzato il seguente testbench.

1. library ieee;

2. use ieee.std\_logic\_1164.all;

3.

4. entity testbench is

5. end testbench;

6.

7. architecture tb of testbench is

8.

9. component Sistema

10. port (input\_16\_4 : in std\_logic\_vector (0 to 15);

11. output\_16\_4 : out std\_logic\_vector (0 to 3);

12. control\_16\_1\_sis : in std\_logic\_vector (0 to 3);

13. control\_1\_4\_sis : in std\_logic\_vector (0 to 1));

14. end component;

15.

16. signal input\_16\_4 : std\_logic\_vector (0 to 15);

17. signal output\_16\_4 : std\_logic\_vector (0 to 3);

18. signal control\_16\_1\_sis : std\_logic\_vector (0 to 3);

19. signal control\_1\_4\_sis : std\_logic\_vector (0 to 1);

20.

21. begin

22.

23. dut : Sistema

24. port map (input\_16\_4 => input\_16\_4,

25. output\_16\_4 => output\_16\_4,

26. control\_16\_1\_sis => control\_16\_1\_sis,

27. control\_1\_4\_sis => control\_1\_4\_sis);

28.

29. stimuli : process

30. begin

31.

32. input\_16\_4 <= (others => '0');

33. control\_16\_1\_sis <= (others => '0');

34. control\_1\_4\_sis <= (others => '0');

35.

36.

37.

38. wait for 100 ns;

39.

40.

41. input\_16\_4 <= "0000000010000000";

42. control\_16\_1\_sis <= "0001";

43. control\_1\_4\_sis <= "01";

44.

45. wait for 100 ns;

46.

47. input\_16\_4 <= "0000000000000010";

48. control\_16\_1\_sis <= "1111";

49. control\_1\_4\_sis <= "11";

50.

51.

52. wait;

53. end process;

54.

55. end tb;

56.

Nel primo caso, l’output risulta essere **1 sulla terza uscita,** nel secondo, risulta essere **0 sull’ultima uscita**.

## Esercizio 1.3: Implementazione su board dell’esercizio 1.2

Sintetizzare ed implementare su board il progetto della rete di interconnessione sviluppato al punto 1.2, utilizzando gli switch per fornire gli input di selezione e i led per visualizzare i 4 bit di uscita. Per quanto riguarda i 16 bit dato in input, essi devono essere immessi mediante switch, 8 bit alla volta, sviluppando un’apposita “rete di controllo” per l’acquisizione che utilizzi due bottoni della board per caricare rispettivamente la prima e la seconda metà del dato in ingresso.

#### Progetto e architettura

#### Implementazione

#### Simulazione

## Esercizio 2.1: ROM combinatoria con sistema M.

Progettare, implementare in VHDL e testare mediante simulazione un sistema S composto da una ROM puramente combinatoria di 16 locazioni da 8 bit ciascuna e da una macchina combinatoria M che opera come segue: fornito al sistema un indirizzo A di 4 bit, il sistema restituisce il valore contenuto nella ROM all’indirizzo A opportunamente “trasformato” attraverso la macchina M. Il comportamento della macchina M è totalmente a scelta dello studente, l’unico vincolo è che essa prenda in ingresso 8 bit e ne fornisca in uscita 4

#### Progetto e architettura

L’esercizio richiede di implementare un sistema con una [**ROM combinatoria**](#_ROM_da_16) **da 16 locazioni da 8 bit, con un sistema M qualsiasi che pone in uscita 4 bit.**

**Immagine che contiene testo, diagramma, linea, schermata

Descrizione generata automaticamente**

Il sistema M, tramite un bit di selezione sceglie i 4 bit da portare in uscita:

* Se 0: i bit da 0 a 3
* Se 1: i bit da 4 a 7

#### Implementazione

Di seguito, troviamo l’implementazione del *top-module Sistema e del sistema M.* L’implementazione della ROM è consultabile in Appendice.

**Componente M**

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6. entity componentM is

7. port (

8.

9. input\_M : in std\_logic\_vector(0 to 7);

10. output\_M: out std\_logic\_vector(0 to 3);

11. selection: in std\_logic

12.

13. );

14. end entity componentM;

15.

16. architecture dataflow of componentM is

17.

18. begin

19.

20. output\_M <= input\_M(0 to 3) when selection = '0' else --i primi valori

21. input\_M (4 to 7) when selection = '1' else -- i secondi

22. "----";

23.

24. end architecture dataflow;

**Top-Module**

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6. entity Sistema is

7. port (

8.

9. controller\_mem\_sis: in std\_logic\_vector(0 to 3);

10.

11. controller\_M\_sis : in std\_logic;

12.

13. output\_sis : out std\_logic\_vector(0 to 3)

14.

15. );

16. end entity Sistema;

17.

18.

19. architecture structural of Sistema is

20.

21. signal output\_mem : std\_logic\_vector(0 to 7) := (others => '0');

22.

23. component ROM port(

24. output : out std\_logic\_vector(0 to 7);

25.

26. address : in std\_logic\_vector(0 to 3)

27.

28. );

29. end component;

30.

31. component componentM port(

32.

33. input\_M : in std\_logic\_vector(0 to 7);

34. output\_M: out std\_logic\_vector(0 to 3);

35. selection: in std\_logic

36.

37. );

38. end component;

39.

40. begin

41.

42. ROM\_1 : ROM port map(

43. address(0) => controller\_mem\_sis(0),

44. address(1) => controller\_mem\_sis(1),

45. address(2) => controller\_mem\_sis(2),

46. address(3) => controller\_mem\_sis(3),

47.

48. output(0) => output\_mem(0),

49. output(1) => output\_mem(1),

50. output(2) => output\_mem(2),

51. output(3) => output\_mem(3),

52. output(4) => output\_mem(4),

53. output(5) => output\_mem(5),

54. output(6) => output\_mem(6),

55. output(7) => output\_mem(7)

56.

57. );

58.

59. componentM\_1: componentM port map(

60.

61. input\_M(0) => output\_mem(0),

62. input\_M(1) => output\_mem(1),

63. input\_M(2) => output\_mem(2),

64. input\_M(3) => output\_mem(3),

65. input\_M(4) => output\_mem(4),

66. input\_M(5) => output\_mem(5),

67. input\_M(6) => output\_mem(6),

68. input\_M(7) => output\_mem(7),

69.

70.

71. output\_M(0)=> output\_sis(0),

72. output\_M(1)=> output\_sis(1),

73. output\_M(2)=> output\_sis(2),

74. output\_M(3)=> output\_sis(3),

75.

76.

77. selection => controller\_M\_sis

78.

79. );

80.

81.

82.

83. end architecture structural;

84.

#### Simulazione

Per la simulazione, abbiamo considerato il seguente testbench.

1. library ieee;

2. use ieee.std\_logic\_1164.all;

3.

4. entity testbench is

5. end testbench;

6.

7. architecture tb of testbench is

8.

9. component Sistema

10. port (controller\_mem\_sis : in std\_logic\_vector (0 to 3);

11. controller\_M\_sis : in std\_logic;

12. output\_sis : out std\_logic\_vector (0 to 3));

13. end component;

14.

15. signal controller\_mem\_sis : std\_logic\_vector (0 to 3);

16. signal controller\_M\_sis : std\_logic;

17. signal output\_sis : std\_logic\_vector (0 to 3);

18.

19. begin

20.

21. dut : Sistema

22. port map (controller\_mem\_sis => controller\_mem\_sis,

23. controller\_M\_sis => controller\_M\_sis,

24. output\_sis => output\_sis);

25.

26. stimuli : process

27. begin

28.

29. controller\_mem\_sis <= (others => '0');

30. controller\_M\_sis <= '0';

31.

32.

33. wait for 100 ns;

34.

35. controller\_M\_sis <= '1';

36. controller\_mem\_sis <= "1000";

37.

38. wait for 100 ns;

39.

40. controller\_M\_sis <= '1';

41. controller\_mem\_sis <= "1100";

42.

43. wait for 100 ns;

44.

45. controller\_M\_sis <= '0';

46.

47.

48. wait for 100 ns;

49. wait;

50. end process;

51.

52. end tb;

Nel testbench riportato, abbiamo provato a utilizzare vari indirizzi per portare al sistema M dei dati prelevati dalla ROM, ulteriormente *filtrati* dal selezionatore del sistema M. Nel primo caso, abbiamo in uscita **il numero 8, nel secondo caso, la lettera c, e nel terzo caso 0. (E’ possibile visualizzare i contenuti della ROM in Appendice)**

## Esercizio 2.2: Implementazione su board della ROM combinatoria con sistema M.

Sintetizzare ed implementare su board il progetto del sistema ROM+M sviluppato al punto 2.1, utilizzando gli switch per fornire l’indirizzo della ROM da cui leggere i valori da trasformare e i led per visualizzare i 4 bit di uscita.

#### Progetto e architettura

#### Implementazione

#### Simulazione

# Capitolo 2: Reti Sequenziali Elementari

## Esercizio 3.1: Riconoscitore di sequenza 101 (senza sovrapposizione/sovrapposizione parziale).

Progettare, implementare in VHDL e testare mediante simulazione una macchina in grado di riconoscere la sequenza 101. La macchina prende in ingresso un segnale binario i che rappresenta il dato, un segnale A di tempificazione e un segnale M di modo, che ne disciplina il funzionamento, e fornisce un’uscita Y alta quando la sequenza viene riconosciuta. In particolare,

- se M=0, la macchina valuta i bit seriali in ingresso a gruppi di 3 (sequenze non sovrapposte),

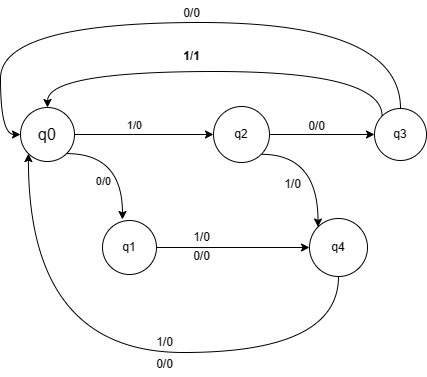
- se M=1, la macchina valuta i bit seriali in ingresso uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta (sequenze parzialmente sovrapposte).

#### Progetto e architettura

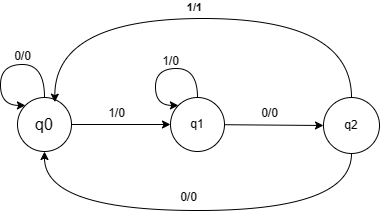
L’esercizio in questione richiede di implementare un riconoscitore di sequenza **101**, che tramite un flag M si comporti come un riconoscitore *con sovrapposizione parziale* oppure *senza sovrapposizione.*

Il riconoscitore di sequenza nelle due modalità, possono essere progettate come delle *macchine a stati finiti,* come di seguito riportato:

**Senza sovrapposizione**

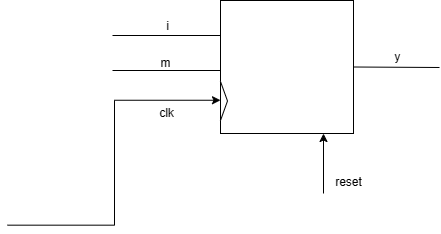
****Questa evoluzione della macchina a stati permette di valutare i bit in ingresso a gruppi di tre.

**Sovrapposizione parziale**

****

Questa evoluzione della macchina a stati, ci permette di avere nel riconoscitore una sovrapposizione *parziale.*

L’architettura può essere vista come un sistema che prende in ingresso il selezionatore M, il segnale di input, il clock e un reset; invece, come uscita abbiamo solo il segnale y.



#### Implementazione

Di seguito, è riportata l’implementazione del riconoscitore richiesto.

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6. entity riconoscitoristotsparz is

7. port (

8. y : out std\_logic;

9. i : in std\_logic;

10. clk : in std\_ulogic;

11. reset: in std\_logic;

12. m: in std\_logic

13. );

14. end entity riconoscitoristotsparz;

15.

16.

17.

18. architecture oneprocesses of riconoscitoristotsparz is

19.

20.

21.

22. --stati

23.

24. type stati is (q0, q1, q2, q3, q4);

25.

26. signal stato\_corrente : stati := q0; --signal stato corrente

27.

28. signal m\_lock : std\_logic :='0';

--variabile per bloccare il selezionatore di riconoscitore di sequenza

29.

30.

31.

32. begin

33.

34.

35.

36. combinatory: process(clk)

37.

38.

39. begin

40. if(rising\_edge(clk)) then

41.

42. if (reset = '1') then --reset

43. stato\_corrente<=q0;

44. y<='0';

45. end if;

46.

47. if(stato\_corrente = q0) then

48. m\_lock<=m;

49. end if;

50.

51.

52. if (m\_lock='1' ) then --sovrapposizione parziale

53. if ( stato\_corrente = q0 AND i ='1') then

54. stato\_corrente <= q1;

55. y<='0';

56. elsif (stato\_corrente = q0 AND i='0') then

57. stato\_corrente<= q0;

58. y<='0';

59. elsif (stato\_corrente = q1 AND i='0') then

60. stato\_corrente<= q2;

61. y<= '0';

62. elsif (stato\_corrente = q1 AND i = '1') then

63. stato\_corrente <= q1;

64. y<='0';

65. elsif(stato\_corrente = q2 AND i='1') then

66. stato\_corrente<= q0;

67. y<='1';

68. elsif(stato\_corrente=q2 AND i='0') then

69. stato\_corrente <= q0;

70. y<='0';

71. end if;

72.

73. elsif (m\_lock='0') then

74. if(stato\_corrente = q0 AND i ='1') then

75. stato\_corrente <= q2; --corretto

76. y<='0';

77. elsif (stato\_corrente = q0 AND i='0') then

78. stato\_corrente <= q1; --sempre sbagliato!

79. y<='0';

80. elsif(stato\_corrente = q1 AND (i='0' OR i='1')) then

81. stato\_corrente <= q4; --sempre sbagliato

82. y<='0';

83. elsif(stato\_corrente=q4 AND (i='0' OR i='1')) then

84. stato\_corrente <= q0;

85. y<='0';

86. elsif( stato\_corrente = q2 AND i ='1') then

87. stato\_corrente <= q4;

88. y<='0';

89. elsif(stato\_corrente = q2 AND i='0') then

90. stato\_corrente <= q3;

91. y<='0';

92. elsif(stato\_corrente=q3 AND i='1') then

93. stato\_corrente<= q0;

94. y<='1';

95. elsif(stato\_corrente=q3 AND i='0') then

96. stato\_corrente<=q0;

97. y<='0';

98. end if;

99.

100. end if;

101.

102. end if;

103. end process combinatory;

104.

105. end architecture oneprocesses;

106.

La macchina a stati è implementata come fedelmente visto a lezione, con la presenza di un unico **process** per la gestione dello stato e delle uscite in base al fronte di salita del clock. Unica cosa da notare è l’utilizzo del *signal* **m\_lock**, che permette di cambiare modalità del riconoscitore SOLO se ci troviamo nello stato q0, questo previene le possibili oscillazioni tra stati del riconoscitore con *sovrapposizione parziale a quello senza sovrapposizioni.*

#### Simulazione

Per la simulazione, abbiamo stilato il seguente *testbench.*

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5. entity tb is

6. end entity tb;

7.

8. architecture tb\_architecture of tb is

9. signal y\_tb, i\_tb, clk\_tb, reset\_tb, m\_tb : std\_logic := '0';

10. begin

11. uut: entity work.riconoscitoristotsparz

12. port map (

13. y => y\_tb,

14. i => i\_tb,

15. clk => clk\_tb,

16. reset => reset\_tb,

17. m => m\_tb

18. );

19.

20.

21. process

22. begin

23. clk\_tb <= '0';

24. wait for 5 ns;

25. clk\_tb <= '1';

26. wait for 5 ns;

27. end process;

28.

29.

30. process

31. begin

32.

33.

34.

35. wait for 10 ns;

36. reset\_tb <= '0';

37. m\_tb <= '1';

38.

39. wait for 10 ns;

40.

41. i\_tb <= '1';

42. wait for 15 ns;

43. i\_tb <= '0';

44. wait for 15 ns;

45. i\_tb <= '1';

46.

47.

48. wait for 50 ns;

49. m\_tb <= '0';

50. wait for 10 ns;

51.

52. i\_tb <= '1';

53. wait for 10 ns;

54. i\_tb <= '0';

55. wait for 10 ns;

56. i\_tb <= '1';

57.

58.

59.

60.

61. wait for 1000 ns; -- Simulate for a while

62. report "Simulation finished" severity note;

63. wait;

64. end process;

65. end architecture tb\_architecture;

66.

In particolare, abbiamo testato il funzionamento di entrambi i riconoscitori, dando in uscita due volte il valore **1** al riconoscimento della sequenza.

## Esercizio 3.2: Implementazione su board del riconoscitore di sequenza.

Sintetizzare e implementare su board la rete sviluppata al punto precedente, utilizzando uno switch S1 per codificare l’input i e uno switch S2 per codificare il modo M, in combinazione con due bottoni B1 e B2 utilizzati rispettivamente per acquisire l’input da S1 e S2 in sincronismo con il segnale di tempificazione A, che deve essere ottenuto a partire dal clock della board. Infine, l’uscita Y può essere codificata utilizzando un led.

#### Progetto e architettura

#### Implementazione

#### Simulazione

## Esercizio 4.1: Shift Register.

Progettare, implementare in VHDL e testare mediante simulazione un registro a scorrimento di N bit in grado di shiftare a destra o a sinistra di un numero Y variabile di posizioni a seconda di una opportuna selezione. In particolare, i valori possibili di Y sono 1 e 2. L’utente tramite selezione deve scegliere di quante posizioni shiftare. Il componente deve essere realizzato utilizzando sia un a) approccio comportamentale sia un b) approccio strutturale.

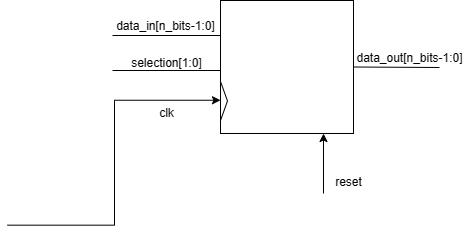
Nota: il numero di bit del registro deve essere implementato come un generic, e dall’esterno deve poter essere scelta la modalità di funzionamento mediante opportuni segnali di selezione.

#### Progetto e architettura

In questo esercizio, ci è stato chiesto di implementare uno shift register in due modi:

1. Con approccio **comportamentale,**
2. Con approccio **strutturale.**

**Approccio comportamentale**

In questo caso, tramite la descrizione **comportamentale, *il tool di sintesi sintetizza automaticamente il comportamento di uno shift register*,** la sua struttura si mostra nel seguente modo:

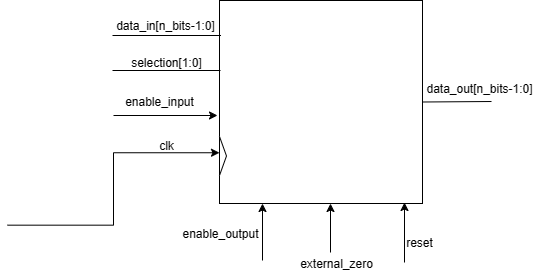
Dallo schema si denotano i bit di ingresso (data\_in di lunghezza n\_bits), una selezione di due bit (selection), l’ingresso per il clock, un segnale di reset e un segnale di uscita (data\_out di lunghezza n\_bits).

**Approccio strutturale**

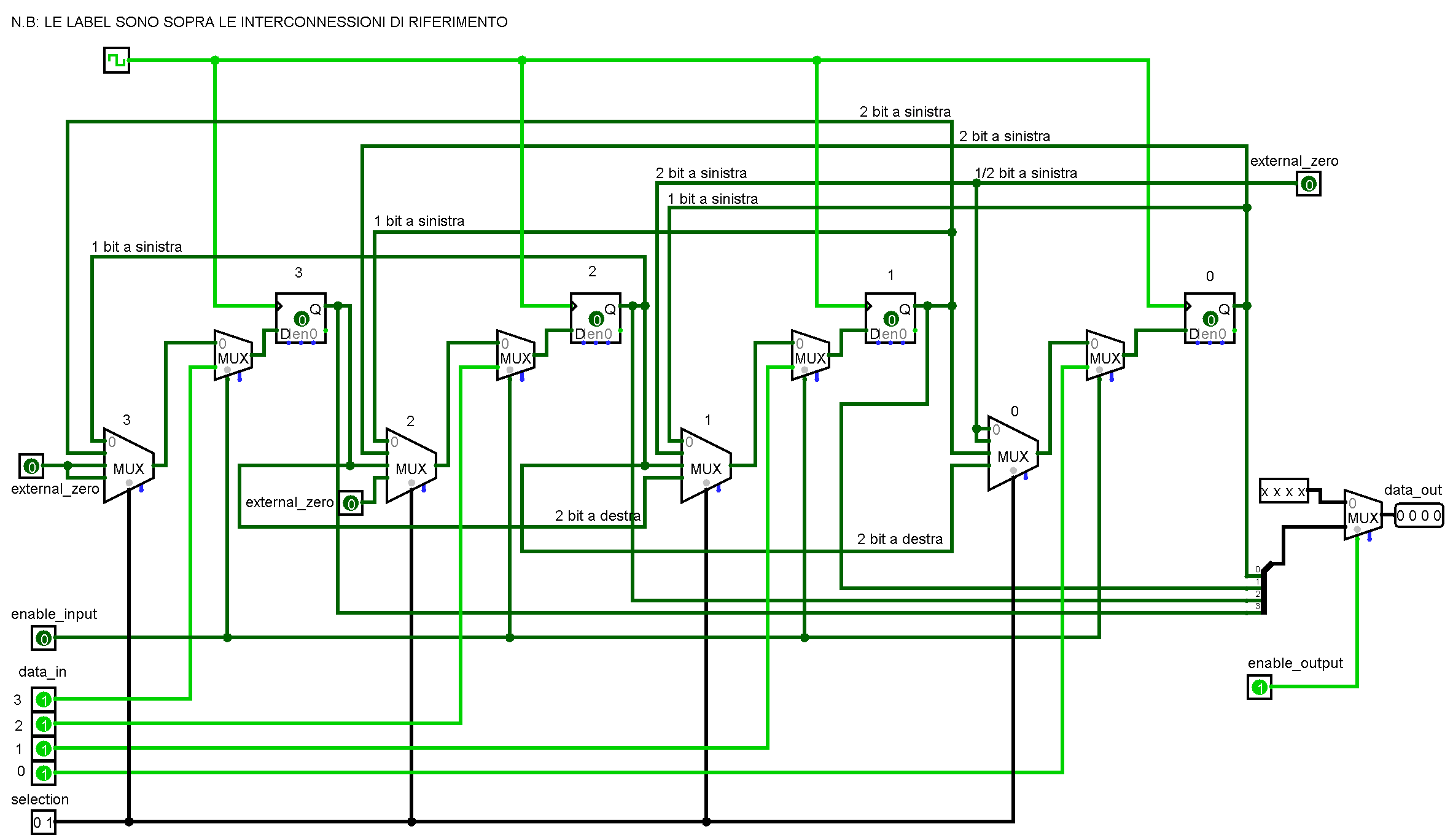
Per l’approccio strutturale, abbiamo bisogno di implementare singolarmente *n\_bits* flip-flop e una serie di multiplexer per permettere di realizzare le funzioni richieste. Nello specifico, abbiamo scelto di utilizzare **flip-flop D, multiplexer 4:1** (*abbiamo utilizzato una versione leggermente modificata del multiplexer presente in Appendice, utilizzando una convenzione* ***downto****)* **e** [**multiplexer 2:1**](#_Multiplexer_2:1)**.**

Prima di tutto, l’entity è stata leggermente modificata con l’aggiunta di **tre** segnali di input:

1. **external\_zero:** segnale che permette di riempire con uno 0 (per scelta progettuale) i registri che restano “vuoti” a causa dello shift.
2. **enable\_input:** abilita la lettura parallela dell’input.
3. **enable\_output:** abilita la scrittura parallela dei valori presenti nei registri in uscita.



Di seguito, è riportato lo schema strutturale dello shift register, con un numero di *n\_bits* pari a 4.

****

Come si può evincere dalla struttura, i **multiplexer 4:1** sono stati utilizzati per scegliere lo shift desiderato tramite il segnale **selection** (*00 di un bit a sinistra, 01 di due bit a sinistra, 10 di un bit a destra, 11 di due bit a desta*). I **multiplexer 2:1,** sono stati inseriti per permettere di scegliere l’ingresso di ogni registro: se quello del multiplexer, o se scegliere di caricare un certo input in **parallelo (data\_in)**, tramite il selezionatore **enable\_input;** inoltre**, c**ome già detto precedentemente, **external\_zero** è utilizzato laddove l’azione di shift lascia il registro “vuoto”. Infine, il selezionatore **enable\_output,** permette di attivare il flusso dei dati verso l’uscita **data\_out.** (N.B nel codice vivado non è presente un multiplexer effettivo, ma solo un assegnazione concorrente con una condizione sul selezionatore *enable\_output*)**.**

#### Implementazione

#### Simulazione

# Appendice

## Multiplexer 2:1

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5. entity multi is

6. port (

7. a0 : in STD\_LOGIC;

8. a1 : in STD\_LOGIC;

9.

10. s : in STD\_LOGIC;

11. y : out STD\_LOGIC --filo singolo, possiamo anche definire un bus di fili (STD\_LOGIC\_VECTOR)

12. );

13. end entity multi;

14.

15. architecture intermediate of multi is

16.

17. begin

18. y <= a0 when s='0' else --input mux precedente

19. a1 when s='1' else --input

20. '-';

21. end architecture intermediate;

22.

## Multiplexer 4:1

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6. entity mux\_4\_1 is

7. port (

8.

9. input\_4\_1 : in STD\_LOGIC\_VECTOR (0 to 3);

10.

11. control\_4\_1 : in STD\_LOGIC\_VECTOR (0 to 1);

12.

13. y\_4\_1 : out STD\_LOGIC

14.

15. );

16. end entity mux\_4\_1;

17.

18.

19. architecture dataflow of mux\_4\_1 is

20.

21.

22.

23. begin

24. y\_4\_1<= input\_4\_1(0) when (control\_4\_1(1)='0' AND control\_4\_1(0)='0') else

25. input\_4\_1(1) when (control\_4\_1(1)='0' AND control\_4\_1(0)='1') else

26. input\_4\_1(2) when (control\_4\_1(1)='1' AND control\_4\_1(0)='0') else

27. input\_4\_1(3) when (control\_4\_1(1)='1' AND control\_4\_1(0)='1') else

28. '-';

29.

30.

31. end architecture dataflow;

32.

## Demultiplexer 1:4

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6. entity Demux\_1\_4 is

7. port (

8. input\_1\_4: in STD\_LOGIC;

9.

10. output\_1\_4: out std\_logic\_vector(0 to 3);

11.

12. control\_1\_4 : in std\_logic\_vector(0 to 1)

13. );

14. end entity Demux\_1\_4;

15.

16.

17. architecture dataflow of Demux\_1\_4 is

18.

19. begin

20. output\_1\_4(0) <= input\_1\_4 when control\_1\_4(1)='0' AND control\_1\_4(0)='0' else

21. '-';

22. output\_1\_4(1) <= input\_1\_4 when control\_1\_4(1)='0' AND control\_1\_4(0)='1' else

23. '-';

24. output\_1\_4(2) <= input\_1\_4 when control\_1\_4(1)='1' AND control\_1\_4(0)='0' else

25. '-';

26. output\_1\_4(3) <= input\_1\_4 when control\_1\_4(1)='1' AND control\_1\_4(0)='1' else

27. '-‘;

28. end architecture dataflow;

## ROM da 16 locazioni da 8 bit

1. library IEEE;

2. use IEEE.std\_logic\_1164.all;

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity ROM is

8. port (

9. output : out std\_logic\_vector(0 to 7);

10.

11. address : in std\_logic\_vector(0 to 3)

12. );

13. end entity ROM;

14.

15.

16.

17. architecture dataflow of ROM is

18.

19. TYPE ROM\_16\_8 IS ARRAY (0 to 15) of std\_logic\_vector(0 to 7); --dichiaro il tipo ROM formato da un array di array

20.

21. constant MEMORY\_16\_4 : ROM\_16\_8 := (

22. x"00",

23. x"01",

24. x"02",

25. x"03",

26. x"04",

27. x"05",

28. x"06",

29. x"07",

30. x"08",

31. x"09",

32. x"0a",

33. x"0b",

34. x"0c",

35. x"0d",

36. x"0e",

37. x"0f"

38.

39. );

40.

41.

42. begin

43.

44.

45. main: process(address)

46. begin

47. output <= MEMORY\_16\_4(to\_integer(unsigned(address)));

48. end process main;

49.

50.

51. end architecture dataflow;

52.

### Progetto e architettura

<descrizione dell’approccio di progetto utilizzato, disegno architetturale del sistema e dei suoi componenti, descrizione delle funzionalità>

### Implementazione

<codice VHDL dei componenti significativi: componenti elementari riutilizzati in diversi progetti vanno inseriti in un’appendice>