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](#_Toc157355643)

[Esercizio 1.1: Multiplexer 16:1 4](#_Toc157355644)

[Progetto e architettura 4](#_Toc157355645)

[Implementazione 4](#_Toc157355646)

[Simulazione 6](#_Toc157355647)

[Esercizio 1.2: Rete di interconnessione 16:4 7](#_Toc157355648)

[Progetto e architettura 7](#_Toc157355649)

[Implementazione 8](#_Toc157355650)

[Simulazione 9](#_Toc157355651)

[Esercizio 1.3: Implementazione su board dell’esercizio 1.2 11](#_Toc157355652)

[Progetto e architettura 11](#_Toc157355653)

[Implementazione 11](#_Toc157355654)

[Simulazione 11](#_Toc157355655)

[Esercizio 2.1: ROM combinatoria con sistema M. 11](#_Toc157355656)

[Progetto e architettura 11](#_Toc157355657)

[Implementazione 12](#_Toc157355658)

[Simulazione 14](#_Toc157355659)

[Esercizio 2.2: Implementazione su board della ROM combinatoria con sistema M. 15](#_Toc157355660)

[Progetto e architettura 15](#_Toc157355661)

[Implementazione 15](#_Toc157355662)

[Simulazione 15](#_Toc157355663)

[Capitolo 2: Reti Sequenziali Elementari 15](#_Toc157355664)

[Esercizio 3.1: Riconoscitore di sequenza 101 (senza sovrapposizione/sovrapposizione parziale). 15](#_Toc157355665)

[Progetto e architettura 16](#_Toc157355666)

[Implementazione 17](#_Toc157355667)

[Simulazione 19](#_Toc157355668)

[Esercizio 3.2: Implementazione su board del riconoscitore di sequenza. 20](#_Toc157355669)

[Progetto e architettura 20](#_Toc157355670)

[Implementazione 20](#_Toc157355671)

[Simulazione 20](#_Toc157355672)

[Esercizio 4.1: Shift Register. 20](#_Toc157355673)

[Progetto e architettura 21](#_Toc157355674)

[Implementazione 23](#_Toc157355675)

[Simulazione 27](#_Toc157355676)

[Esercizio 5.1: Cronometro 31](#_Toc157355677)

[Progetto e architettura 31](#_Toc157355678)

[Implementazione 33](#_Toc157355679)

[Simulazione 38](#_Toc157355680)

[Esercizio 5.2: Implementazione su board del componente 5.1 40](#_Toc157355681)

[Progetto e architettura 40](#_Toc157355682)

[Implementazione 40](#_Toc157355683)

[Simulazione 41](#_Toc157355684)

[Esercizio 5.3: Estensione del componente sviluppato nei punti precedenti. 41](#_Toc157355685)

[Progetto e architettura 41](#_Toc157355686)

[Implementazione 41](#_Toc157355687)

[Simulazione 41](#_Toc157355688)

[Appendice 42](#_Toc157355689)

[Multiplexer 2:1 42](#_Toc157355690)

[Multiplexer 4:1 42](#_Toc157355691)

[Demultiplexer 1:4 43](#_Toc157355692)

[Flip-Flop D 43](#_Toc157355693)

[ROM da 16 locazioni da 8 bit 44](#_Toc157355694)

[Progetto e architettura 44](#_Toc157355695)

[Implementazione 44](#_Toc157355696)

# 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.

**Forma d’onda**

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

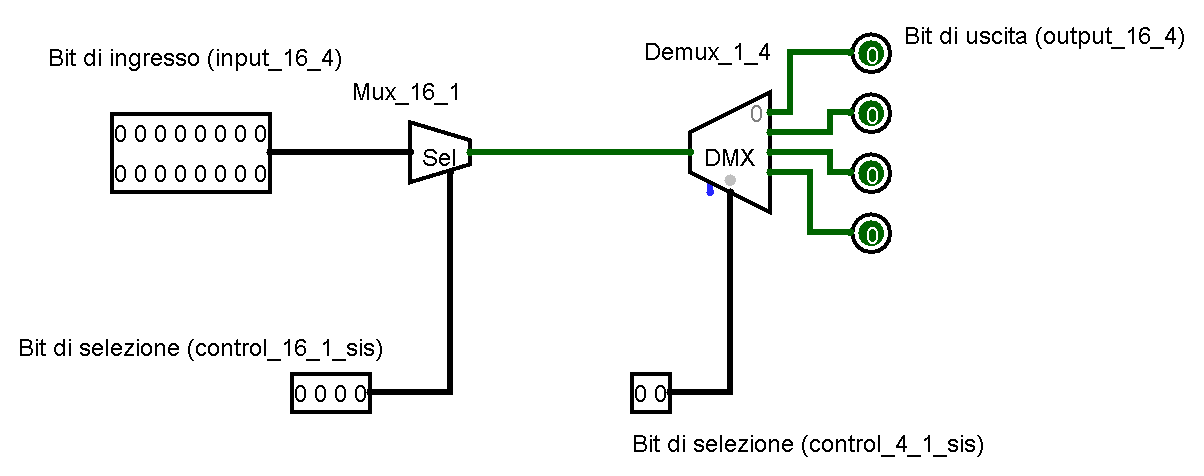
Descrizione generata automaticamente**

## 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.

**Forma d’onda**

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

Descrizione generata automaticamente**

## 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**](#_ROM_da_16_1)**)**

**Forma d’onda:**

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

Descrizione generata automaticamente**

## 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),

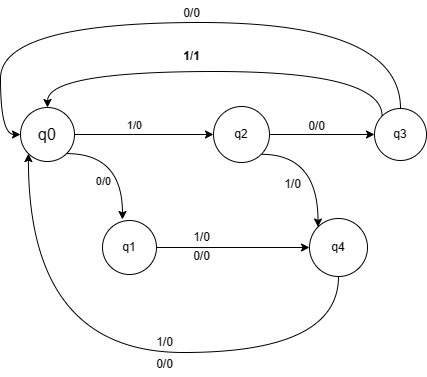
- se M=1, la macchina valuta i bit seriali in ingresso uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta (sequenze parzialmente sovrapposte).

#### Progetto e architettura

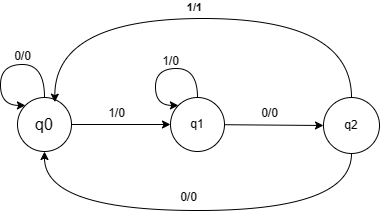
L’esercizio in questione richiede di implementare un riconoscitore di sequenza **101**, che tramite un flag M si comporti come un riconoscitore *con sovrapposizione parziale* oppure *senza sovrapposizione.*

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

**Senza sovrapposizione**

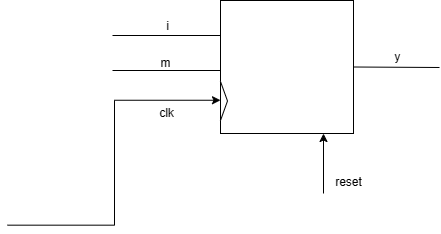
****Questa evoluzione della macchina a stati permette di valutare i bit in ingresso a gruppi di tre.

**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.

**Forma d’onda**

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

Descrizione generata automaticamente**

Il primo riconoscitore è quello con sovrapposizione parziale, il secondo quello con sovrapposizione totale.

## 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.

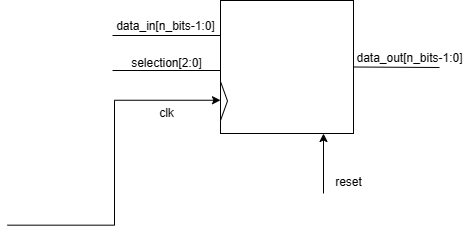
Nota: il numero di bit del registro deve essere implementato come un generic, e dall’esterno deve poter essere scelta la modalità di funzionamento mediante opportuni segnali di selezione.

#### 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 tre bit (*selection*), l’ingresso per il clock, un segnale di reset e un segnale di uscita (*data\_out* di lunghezza n\_bits).

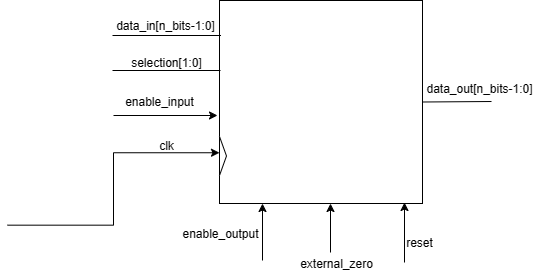
La **selection** da tre bit permette allo shift register di avere tre modalità di selezione (*000, 101, 111*) in più che permettono di porre il dato di ingresso, direttamente in uscita senza alcuno shift.

**Approccio strutturale**

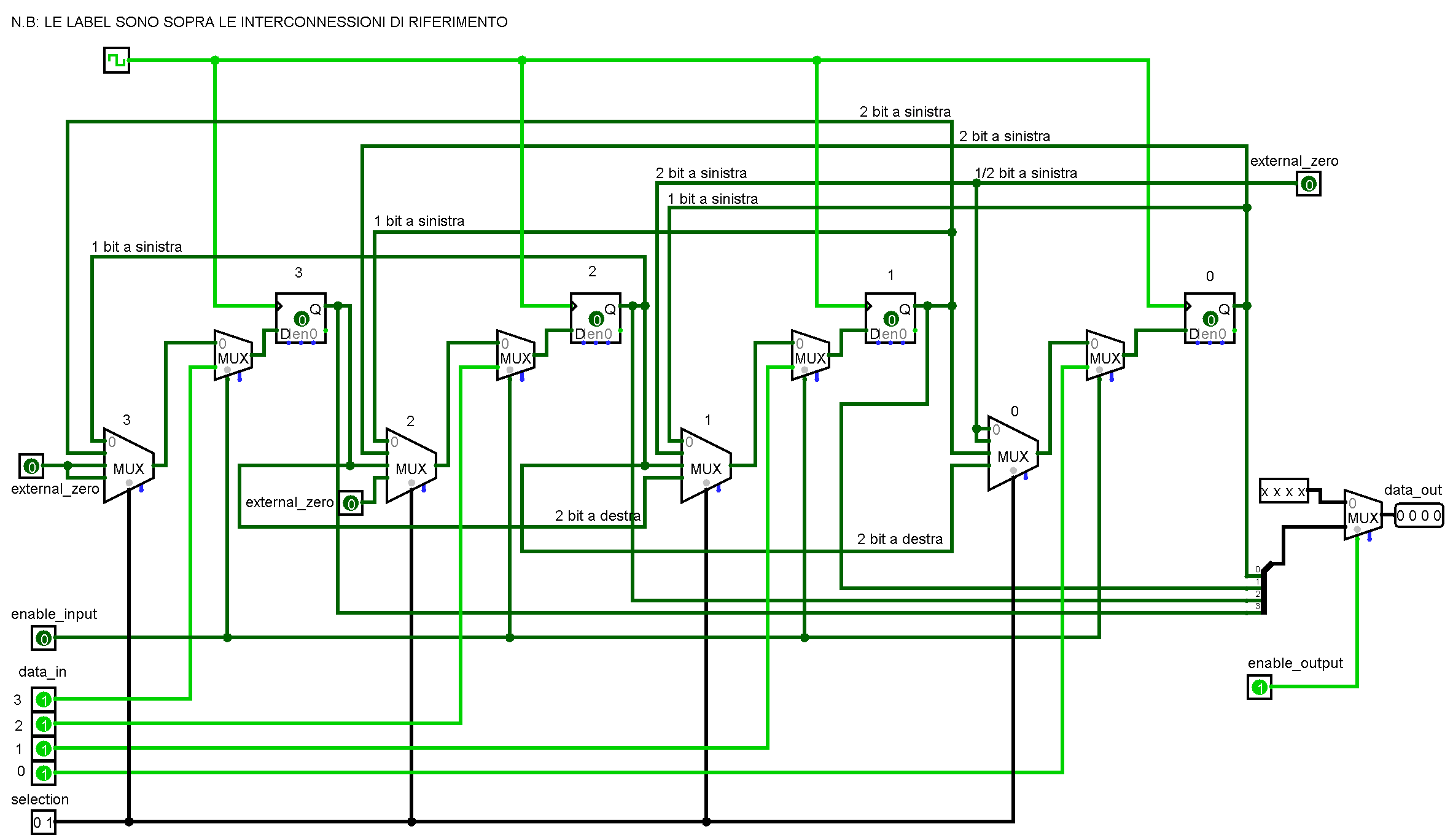
Per l’approccio strutturale, abbiamo bisogno di implementare singolarmente *n\_bits* flip-flop e una serie di multiplexer per permettere di realizzare le funzioni richieste. Nello specifico, abbiamo scelto di utilizzare **flip-flop D, multiplexer 4:1** (*abbiamo utilizzato una versione leggermente modificata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

L’*entity* è stata leggermente modificata con l’aggiunta di **tre** segnali di input e modificando la **selection**:

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.
4. **selection:** lunga due bit invece che tre.



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 l’usita del **multiplexer 4:1**, o se scegliere di caricare un l’input (***data\_in***) in **parallelo,** 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 non è presente l’istanziazione di un multiplexer effettivo, ma solo un’assegnazione concorrente sul segnale **data\_out** con una condizione ***when***sul selezionatore *enable\_output*)**.**

#### Implementazione

Di seguito sono riportate le implementazioni dei due approcci.

**Approccio comportamentale**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity shiftregc is

8. generic (

9. n\_bits : positive :=4

10. );

11. port (

12. data\_in : in std\_logic\_vector(n\_bits-1 downto 0);

13. --dati in ingresso

14. selection : in std\_logic\_vector(2 downto 0); --slezione dello shift, imposto 3 bit per permettere di avere 000 come "stato di quiete" del registro

15. clk : in std\_logic; --segnale per la sincronizzazione

16. reset : in std\_logic; --segnale di reset

17.

18.

19. data\_out : out std\_logic\_vector(n\_bits-1 downto 0) --dati in uscita

20. );

21. end entity shiftregc;

22.

23.

24.

25. architecture comportamentale of shiftregc is

26.

27.

28. shared variable data\_mem : std\_logic\_vector(n\_bits-1 downto 0);

29.

30. begin

31.

32.

33. proc\_name: process(clk)

34. begin

35. if rising\_edge(clk) then

36.

37. if reset = '1' then

38. data\_out <= (others => '0');

--questo ci permette di porre tutti i bit a 0 se il reset è alto

39. data\_mem <= (others => '0');

40. end if;

41.

42. if (selection = "001") then

43. data\_mem := data\_mem(n\_bits-2 downto 0) & '0';

44. data\_out <= data\_mem; --shift a sinistra di 1

45.

46. elsif (selection = "011") then

47. data\_mem := data\_mem(n\_bits-3 downto 0) & "00";

48. data\_out <= data\_mem; --shift a sinistra di 2

49.

50. elsif selection = "010" then

51. data\_mem := '0' & data\_mem(n\_bits-1 downto 1);

52. data\_out <= data\_mem; --shift a destra di 1

53.

54. elsif selection = "100" then

55. data\_mem := "00" & data\_mem(n\_bits-1 downto 2);

56. data\_out <= data\_mem; --shift a destra di 2

57.

58. else

59. data\_out<= data\_mem;

60. end if;

61.

62. end if;

63. end process proc\_name;

64.

65. proc\_data\_in: process(data\_in)

66. begin

67.

68. data\_mem:=data\_in;

69.

70. end process;

71.

72.

73. end architecture comportamentale;

74.

Nell’implementazione riportata sono presenti due ***process***:

1. Il *primo, sensibile solo al clock*, permette di implementare la logica dello **shift-register,** utilizzando anche una variabile d’appoggio chiamata ***data\_mem*** che permette di salvare il risultato e porlo in uscita, oltre che tenere lo stato del dato dopo un certo numero di shift.
2. Il *secondo, sensibile solo a* ***data\_in,*** aggiorna ***data\_mem*** (in un solo *delta cycle*) quando **data\_in** varia.

**Approccio strutturale**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5. entity shiftregs is

6. generic (

7. n\_bits : positive :=4

8. );

9. port (

10. data\_in : in std\_logic\_vector(n\_bits-1 downto 0);

11. --dati in ingresso

12. selection : in std\_logic\_vector(1 downto 0);

--slezione dello shift, imposto 3 bit per permettere di avere 000 come "stato di quiete" del registro

13.

clk : in std\_logic; --segnale per la sincronizzazione

14. reset : in std\_logic; --segnale di reset

15.

16. external\_zero: in std\_logic; --segnale con valore 0 per shift

17.

enable\_input: in std\_logic;

--enable per far scorrere l'input nei registri in maniera parallela

18.

enable\_output: in std\_logic; --enable per far scorrere l'ouput in maniera parallela

19.

20. data\_out : out std\_logic\_vector(n\_bits-1 downto 0) --dati in uscita

21. );

22. end entity shiftregs;

23.

24.

25.

26. architecture structural of shiftregs is

27.

28.

29. component ffd is

30. port(

31. D : in std\_logic;

32. clk: in std\_logic;

33. reset: in std\_logic;

34.

35. Q : out std\_logic

36. );

37. end component;

38.

39. component mux\_4\_1 is

40. port(

41.

42. input\_4\_1 : in STD\_LOGIC\_VECTOR (3 downto 0);

43.

44. control\_4\_1 : in STD\_LOGIC\_VECTOR (1 downto 0);

45.

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

47. );

48. end component;

49.

50. component multi is

51. port(

52.

53. a0 : in STD\_LOGIC;

54. a1 : in STD\_LOGIC;

55.

56. s : in STD\_LOGIC;

57. y : out STD\_LOGIC

58.

59. );

60. end component;

61.

62.

63. -- SEGNALI DI INTERCONNESSIONE --

64. signal memOuts: std\_logic\_vector(n\_bits downto 0) := (others => '0');

--segnale di uscita dei registri in parallelo, serve anche per shift a destra di 1

65. signal muxToMux: std\_logic\_vector(n\_bits-1 downto 0);

--collegamento tra multiplexer 4:1 e multiplexer 2:1

66. signal muxToMem: std\_logic\_vector(n\_bits-1 downto 0);

--collegamento tra multiplexer 2:1 e registro

67.

68.

69.

70. begin

71.

72.

73.

74. gen: for i in 0 to n\_bits-1 generate

75.

76. ff: ffd port map(

77. D => muxToMem(i),

78. clk => clk,

79. reset => reset,

80.

81. Q => memOuts(i)

82. );

83.

84.

85. end generate;

86.

87.

88. gen2: for i in 0 to n\_bits-1 generate

89.

90. muxs\_2\_1: multi port map(

91.

92. a0 => muxToMux(i),

93. a1 => data\_in(i),

94. s=> enable\_input,

95. y=> muxToMem(i)

96.

97. );

98.

99. end generate;

100.

101.

102. mux0: mux\_4\_1 port map(

103.

104. input\_4\_1(0) => external\_zero, --di uno a sinistra

105. input\_4\_1(1) => external\_zero, --di due a sinistra

106. input\_4\_1(2) => memOuts(1), --di uno a destra

107. input\_4\_1(3) => memOuts(2), --di due a destra

108.

109. control\_4\_1 => selection,

110.

111. y\_4\_1 => muxToMux(0)

112.

113. );

114.

115.

116. mux1: mux\_4\_1 port map(

117. input\_4\_1(0) => memOuts(0),

118. input\_4\_1(1) => external\_zero,

119. input\_4\_1(2) => memOuts(2),

120. input\_4\_1(3) => memOuts(3),

121.

122.

123. control\_4\_1 => selection,

124.

125. y\_4\_1 => muxToMux(1)

126.

127. );

128.

129.

130. muxn: mux\_4\_1 port map(

131.

132. input\_4\_1(0) => memOuts(n\_bits-2),

133. input\_4\_1(1) => memOuts(n\_bits-3),

134. input\_4\_1(2) => external\_zero,

135. input\_4\_1(3) => external\_zero,

136.

137. control\_4\_1 => selection,

138.

139. y\_4\_1 => muxToMux(n\_bits-1)

140. );

141.

142.

143.

144. gen3: for i in 2 to n\_bits-2 generate

145. mux3: mux\_4\_1 port map(

146.

147. input\_4\_1(0) => memOuts(i-1),

148. input\_4\_1(1) => memOuts(i-2),

149. input\_4\_1(2) => memOuts(i+1),

150. input\_4\_1(3) => memOuts(i+2),

151.

152. control\_4\_1 => selection,

153.

154. y\_4\_1 => muxToMux(i)

155. );

156. end generate;

157.

158.

159. --gestione output

160. data\_out <= memOuts(n\_bits-1 downto 0) when enable\_output = '1' else

161. (others => '-') ;

162.

163.

164.

165. end architecture structural;

166.

In questo approccio, sono stati instanziati *n\_bits* ***flip-flop D,*** *n\_bits* ***multiplexer\_2\_1 e*** *n\_bits* ***multiplexer\_4\_1.*** Questi ultimi sono stati differenziati a seconda della loro “posizione”, che richiedono degli ingressi diversi rispetto al peso dei bit e la lunghezza *n\_bits* scelta. (Notare che il primo blocco generato [con i=0] di **flip-flop/multiplexer\_4\_1/multiplexer\_2\_1** si riferiscono all’*n-1*-esimo bit della sequenza, il secondo al *n\_bits-2*-esimo e così via). Si noti, come il signal **memOuts** sia composto da un numero di bit pari a *n\_bits+1,* questo serve ad evitare errori nel collegamento durante i cicli *generate.*

#### Simulazione

**Approccio comportamentale**

Per l’approccio comportamentale abbiamo utilizzato il testbench di seguito riportato.

1. library ieee;

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

3. use ieee.numeric\_std.all;

4.

5. entity tb is

6. end tb;

7.

8. architecture behavior of tb is

9. constant n\_bits: positive := 4;

10. signal data\_in : std\_logic\_vector(n\_bits-1 downto 0);

11. signal selection : std\_logic\_vector(2 downto 0);

12. signal clk\_sig : std\_logic;

13. signal reset : std\_logic := '0';

14. signal data\_out : std\_logic\_vector(n\_bits-1 downto 0);

15.

16. component shiftregc is

17. generic (

18. n\_bits : in integer := 4

19. );

20. port (

21. data\_in : in std\_logic\_vector(n\_bits-1 downto 0);

22. selection : in std\_logic\_vector(2 downto 0);

23. clk : in std\_logic;

24. reset : in std\_logic;

25. data\_out : out std\_logic\_vector(n\_bits-1 downto 0)

26. );

27. end component;

28.

29. begin

30. dut: shiftregc

31. generic map (n\_bits => n\_bits)

32. port map (data\_in => data\_in, selection => selection, clk => clk\_sig, reset => reset, data\_out => data\_out);

33.

34. clk\_process : process

35. begin

36. while true loop

37. clk\_sig <= '0';

38. wait for 10 ns;

39. clk\_sig <= '1';

40. wait for 10 ns;

41. end loop;

42. end process;

43.

44. stim\_proc: process

45. begin

46. -- Shift left by 1

47. wait for 5 ns;

48. selection <= "001";

49. data\_in <= "1001";

50. wait for 20 ns;

51.

52. selection<="010";

53. wait for 20 ns;

54.

55. -- Shift right by 1

56. selection <= "010";

57. data\_in <= "1011";

58. wait for 20 ns;

59.

60. -- Shift left by 2

61. selection <= "011";

62. data\_in <= "1111";

63. wait for 20 ns;

64.

65. selection<="000";

66. wait for 10 ns; --qui lo shift register si ferma e pone solo data\_mem in uscita

67.

68. -- Shift right by 2

69. selection <= "100";

70. data\_in <= "1001";

71. wait for 20 ns;

72. selection<="000";

73.

74. -- End simulation

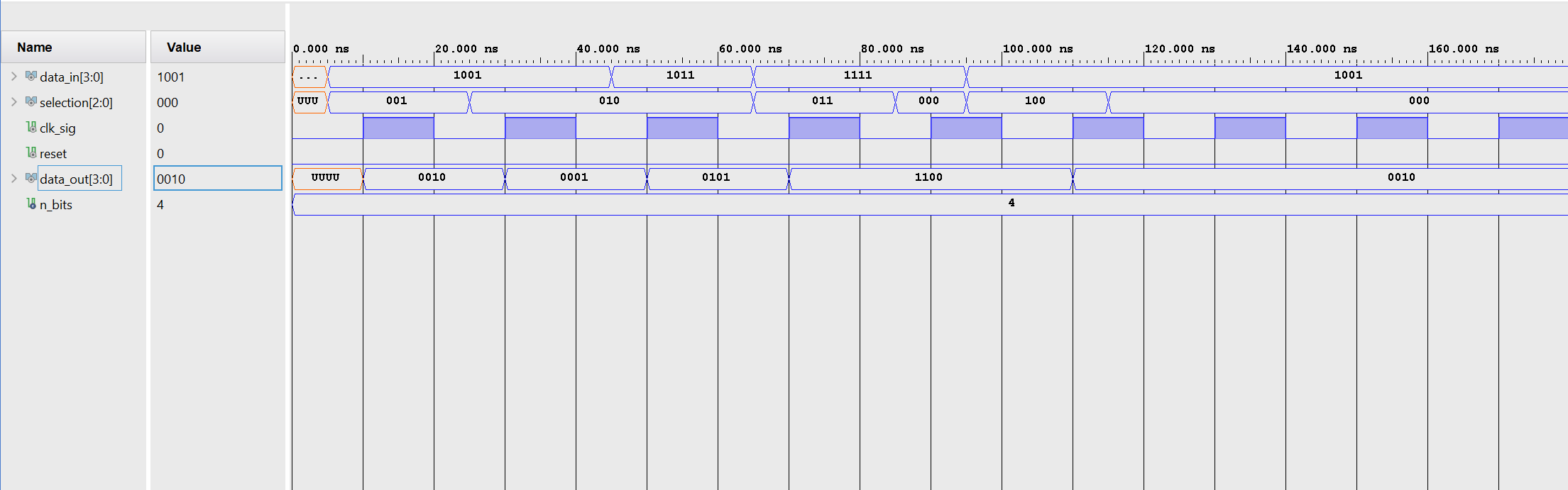
75. wait;

76. end process;

77. end behavior;

78.

**Forma d’onda:**

****

**Approccio strutturale**

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 testbench of tb is

9. -- Constants

10. constant n\_bits : positive := 4;

11.

12. signal data\_in\_sig : std\_logic\_vector(n\_bits-1 downto 0);

13. signal selection\_sig: std\_logic\_vector(1 downto 0);

14. signal clk\_sig, reset\_sig, external\_zero\_sig, enable\_input\_sig, enable\_output\_sig : std\_logic;

15. signal data\_out\_sig : std\_logic\_vector(n\_bits-1 downto 0);

16.

17. -- Instantiate the shiftregs component

18. component shiftregs

19. generic (

20. n\_bits : positive := 4

21. );

22. port (

23. data\_in : in std\_logic\_vector(n\_bits-1 downto 0);

24. selection : in std\_logic\_vector(1 downto 0);

25. clk : in std\_logic;

26. reset : in std\_logic;

27. external\_zero : in std\_logic;

28. enable\_input : in std\_logic;

29. enable\_output : in std\_logic;

30. data\_out : out std\_logic\_vector(n\_bits-1 downto 0)

31. );

32. end component;

33.

34. begin

35.

36.

37.

38.

39. -- Instantiate the shiftregs component

40. UUT: shiftregs

41. generic map (

42. n\_bits => n\_bits

43. )

44. port map (

45. data\_in => data\_in\_sig,

46. selection => selection\_sig,

47. clk => clk\_sig,

48. reset => reset\_sig,

49. external\_zero => external\_zero\_sig,

50. enable\_input => enable\_input\_sig,

51. enable\_output => enable\_output\_sig,

52. data\_out => data\_out\_sig

53. );

54.

55.

56. clk\_process :process

57. begin

58. while true loop

59. clk\_sig <= '0';

60. wait for 5 ns;

61. clk\_sig <= '1';

62. wait for 5 ns;

63. end loop;

64. end process;

65.

66. -- Stimulus process

67. stimulus\_process: process

68. begin

69. -- Initialize inputs

70.

71.

72. reset\_sig <= '0';

73. external\_zero\_sig <= '0';

74. enable\_input\_sig <= '0';

75. enable\_output\_sig <= '0';

76.

77. wait for 10 ns;

78.

79. -- Primo input

80. data\_in\_sig <= "1111"; -- Example input data

81. enable\_input\_sig <= '1';

82. wait for 30 ns;

83.

84. enable\_input\_sig <= '0';

85.

86. selection\_sig<= "00";

87. wait for 10 ns;

88. enable\_output\_sig <= '1';

89.

90. selection\_sig<="01";

91. wait for 10 ns;

92.

93. --secondo input

94. enable\_output\_sig<='0';

95. data\_in\_sig<="0110";

96. enable\_input\_sig<='1';

97.

98. wait for 10 ns;

99.

100. enable\_input\_sig<='0';

101.

102. selection\_sig <= "10";

103. wait for 10 ns;

104. enable\_output\_sig <='1';

105.

106. selection\_sig<="11";

107. wait for 10 ns;

108. enable\_output\_sig<='0';

109.

110.

111.

112.

113.

114. -- Add more test scenarios as needed

115.

116. wait;

117. end process stimulus\_process;

118.

119. end architecture testbench;

120.

Nel testbench sono previsti due input che prima vengono prelevati in maniera *parallela* grazie all’**enable\_input,** successivamente viene posto a 0 e viene scelto un particolare shift tramite **selection.** Grazie a **enable\_output,** possiamo permettere di portare il risultato in uscita in maniera parallela.

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

Descrizione generata automaticamenteForma d’onda:**

In questo caso, avendo maggiore granularità nella scelta dell’output, quando **enable\_output** è posto a 0, il dato in uscita è riempito con una serie di *don’t care*.

## Esercizio 5.1: Cronometro

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

#### Progetto e architettura

Il cronometro è caratterizzato dal conteggio di *ore, minuti e secondi*, e questo comportamento può essere implementato in maniera semplice grazie a dei contatori. Nello specifico abbiamo bisogno di:

1. Un contatore modulo 24 per le ore (*useremo un modulo 32, da 5 bit, che si azzerra quando il conteggio arriva a quota 23*).
2. Due contatori modulo 60 per i minuti e i secondi. (useremo un contatore modulo 64, da 6 bit, che si azzera quando il *conteggio arriva a quota 59*).

Gli schemi possibili per implementare sia i contatori stessi, che una catena di contatori possono essere due: in **parallelo** o in **serie.**

I contatori implementati in **parallelo** ricevono *tutti* contemporaneamente l’impulso di clock, e per permettere la sincronizzazione abbiamo un ulteriore impulso di conteggio, che consentedi far avanzare i contatopri posti nelle posizioni “più significative” grazie ad opportune condizioni. Questo approccio di solito non è preferito, anche essendo più veloce di quelli in serie, con meno incidenza da parte dei ritardi di propagazione, risulta instabile da gestire per contatori di modulo elevato, anche con la composizione di contatori di modulo minore.

I contatori implementati in **serie**, consistono nell’utilizzare solo un impulso di conteggio (che può essere anche il clock, o un segnale generato a partire da esso) che viene fornito come clock al contatore di peso minore (nel nostro caso a quello dei secondi). Quest’ultimo, una volta arrivato al valore massimo di conteggio o a quello desiderato, invia un inpulso di fine conteggio, collegato alla porta del clock del prossimo contatore. Questa metodologia è utilizzata per progettare i contatori di dimensioni più piccole, utilizzando *flip-flop T* o *JK* e collegandoli tra loro in serie. Il problema principale dell’approccio in serie, sta nei ritardi di propagazione del segnale di *fine conteggio* ai prossimi contatori, ma il sistema ne giova in fatto di stabilità.

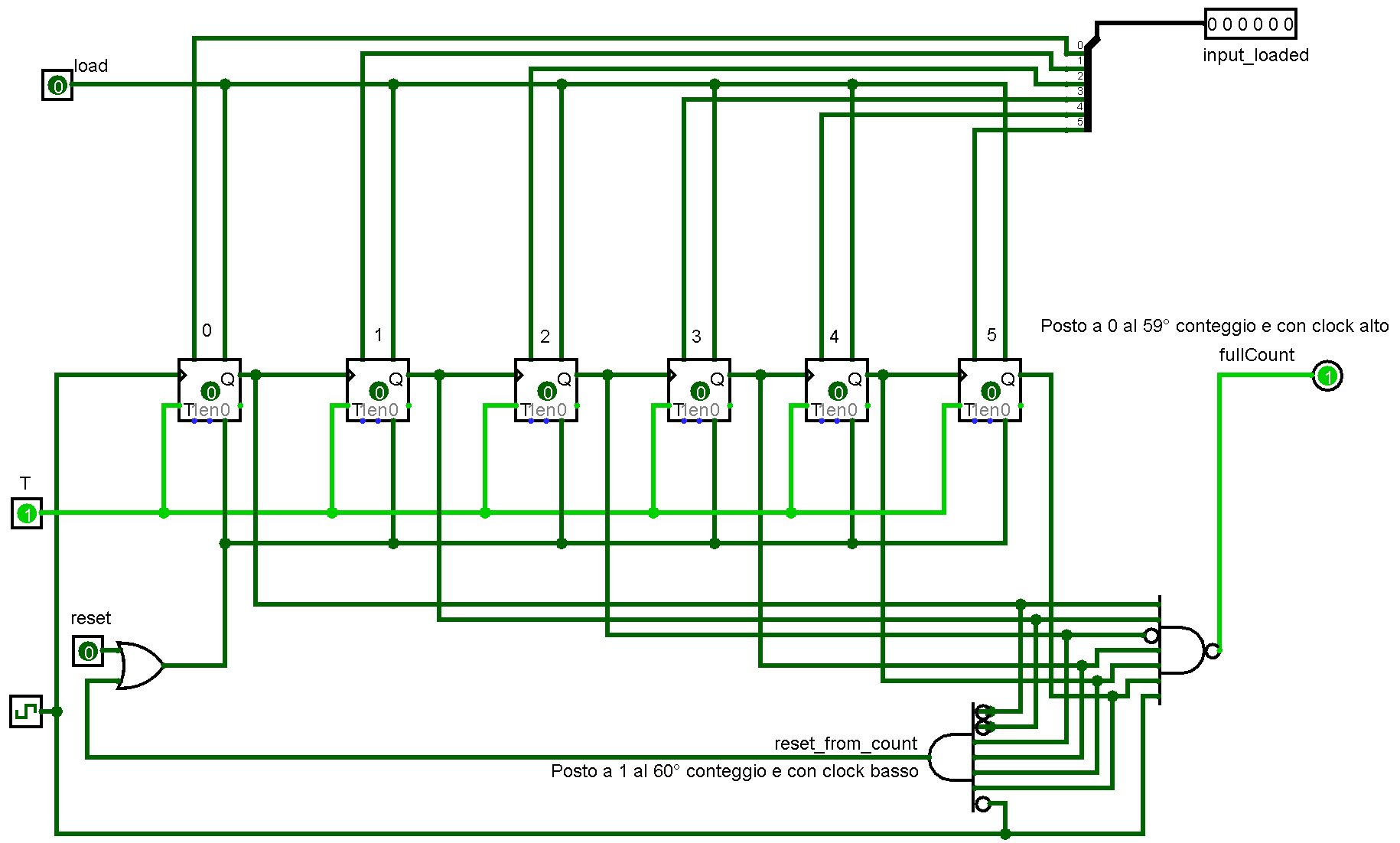
Inoltre, per tutti e due gli approcci, il conteggio deve essere attivato sul fronte di *discesa* per fare in modo che il contatore o la serie di contatori funzionino adeguatamente (Se volessi contare in maniera decrescente potrei decrementare il conteggio sul fronte di *salita* del segnale).

Per questo progetto è stato utilizzato un approccio ***seriale,*** infatti, il sistema compare come segue.

Immagine che contiene diagramma, linea, Disegno tecnico, Piano

Descrizione generata automaticamente

N.B. Nell’implementazione vhdl, il ***load*** è stato trattato in maniera *asincrona.*

Di seguito è riportato lo schema di un contatore modulo 64 utile per il conteggio di minuti e secondi. Per quanto riguarda quello *modulo 32* per il conteggio delle ore la struttura risulta la stessa ma con cinque *flip-flop T* invece che sei.

Dallo schema, si nota che per i contatori è stato usato sempre un approccio di tipo ***seriale***, che permette il corretto avanzamento del contatore senza complicare troppo la logica generale. Si noti che il ***load*** e ***input\_loaded****,* permettono di creare la funzione di *load* in maniera asincrona rispetto al clock, come fatto per il reset. La rete di porte in basso a destra permette di porre il segnale ***fullCount*** a 0 (impulso di conteggio per il prossimo contatore) quando si è raggiunto il 59-esimo conteggio e il clock risulta ***alto,*** appena il segnale di clock divenda basso (60-esimo conteggio), il contatore viene resettato.

#### Implementazione

Per le implementazioni abbiamo seguito un approccio:

1. *strutturale* per il cronometro
2. *strutturale* per i l’istanziazione dei *flip flop* all’interno dei contatori, e *comportamentale* per il reset asincrono, il reset dato dal conteggio e la gestione di ***fullCount,***
3. *comportamentale*, per i flip flop T*.*

**Flip Flop T**

Di seguito, riporteremo l’implementazione del *flip-flop* T, che non abbiamo messo in *Appendice* perché non rispecchia l’implementazione classica di questo *flip-flop* per via della funzionalità di set.

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity fft is

8. port (

9. clk : in std\_logic;

10. reset : in std\_logic; --reset dato esternamente

11. set: in std\_logic;

12. input\_loaded: in std\_logic;

13.

14. Y : out std\_logic

15.

16. );

17. end entity fft;

18.

19.

20. architecture rtl of fft is

21.

22. signal T : std\_logic :='0';

23.

24. begin

25. ff: process (clk, reset, set)

26. begin

27.

28. if(set='1') then

29. T<=input\_loaded;

30. end if;

31. if (reset='1') then

32. T<='0';

33. elsif falling\_edge(clk) then

34. if(set='0') then

35. T<= not T;

36. end if;

37. end if;

38. end process;

39.

40. Y<=T; --assegnazione di T a Y

41. end architecture rtl;

**Contatore modulo 60**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity contMod60 is

8. port (

9. clk : in std\_logic;

10. set: in std\_logic;

11. input\_loaded: in std\_logic\_vector(5 downto 0);

12. reset : in std\_logic;

13.

14. cont: out std\_logic\_vector(5 downto 0);

15. fullCount: out std\_logic

16.

17. );

18. end entity contMod60;

19.

20.

21. architecture rtl of contMod60 is

22.

23.

24.

25. component fft

26. port(

27. clk : in std\_logic;

28. reset : in std\_logic; --reset dato esternamente

29. set: in std\_logic;

30. input\_loaded: in std\_logic;

31.

32. Y : out std\_logic

33. );

34. end component;

35.

36.

37. signal wirings : std\_logic\_vector (5 downto 0);

38. signal reset\_count : std\_logic;

39.

40. begin

41.

42.

43. ff0: fft port map(

44. clk => clk,

45. reset => reset\_count,

46. set => set,

47. input\_loaded => input\_loaded(0),

48.

49. Y=> wirings(0)

50. );

51.

52.

53. gen : for i in 1 to 5 generate

54.

55. ffn: fft port map(

56. clk => wirings(i-1),

57. reset => reset\_count,

58. set => set,

59. input\_loaded => input\_loaded(i),

60.

61. Y=> wirings(i)

62. );

63. end generate;

64.

65.

66. cont <= wirings;

67.

68.

69. p: process (wirings, clk, reset)

70. begin

71. fullCount<='1';

72. reset\_count<='0';

73.

74. if(wirings = "111100" or reset='1') then

75. reset\_count<='1';

76.

77. end if;

78.

79. if(wirings = "111100") then

80. fullCount<='0';

81. end if;

82.

83. end process;

84.

85. end architecture rtl;

86.

Il contatore modulo 60 è realizzato con l’instanziazione di sei *flip-flop T,* legati l’uno all’altro come mostrato nella sezione [Progetto e architettura](#_Progetto_e_architettura). In più, abbiamo un *process* per la gestione del reset *asincrono* o dato dal conteggio, e del raggiungimento del **fullCount.**

**Contatore modulo 24**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity contMod24 is

8. port (

9. clk : in std\_logic;

10. set: in std\_logic;

11. input\_loaded: in std\_logic\_vector(4 downto 0);

12. reset : in std\_logic;

13.

14. cont: out std\_logic\_vector(4 downto 0);

15. fullCount: out std\_logic

16.

17. );

18. end entity contMod24;

19.

20.

21. architecture rtl of contMod24 is

22.

23.

24.

25. component fft

26. port(

27. clk : in std\_logic;

28. reset : in std\_logic; --reset dato esternamente

29. set: in std\_logic;

30. input\_loaded: in std\_logic;

31.

32. Y : out std\_logic

33. );

34. end component;

35.

36.

37. signal wirings : std\_logic\_vector (4 downto 0);

38. signal reset\_count : std\_logic;

39.

40. begin

41.

42.

43. ff0: fft port map(

44. clk => clk,

45. reset => reset\_count,

46. set => set,

47. input\_loaded => input\_loaded(0),

48.

49. Y=> wirings(0)

50. );

51.

52.

53. gen : for i in 1 to 4 generate

54.

55. ffn: fft port map(

56. clk => wirings(i-1),

57. reset => reset\_count,

58. set => set,

59. input\_loaded => input\_loaded(i),

60.

61. Y=> wirings(i)

62. );

63. end generate;

64.

65.

66. cont <= wirings;

67.

68.

69. p: process (wirings, clk, reset)

70. begin

71. fullCount<='1';

72. reset\_count<='0';

73.

74. if(wirings = "11000" or reset='1') then

75. reset\_count<='1';

76.

77. end if;

78.

79. if(wirings = "10111") then

80. fullCount<='0';

81. end if;

82.

83. end process;

84.

85. end architecture rtl;

86.

Per il contatore modulo 24, l’implemtazione cambia di poco, se non per il fatto che il valore per il reset dato dal conteggio è *11000,* ossia 24, invece per il **fullCount** abbassiamo il valore per tutta la durata della 23-esima ora, e risulta utile per la getione effettuata nel **Top-Module** (ossia il ***Cronometro*** vero e proprio) del ***fullCount*** più esterno, che indica il passaggio delle 24 ore.

**Top-Module: Cronometro**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity cronometro is

8. port (

9. clk : in std\_logic;

10. reset: in std\_logic;

11. set: in std\_logic;

12. input\_loaded: in std\_logic\_vector(16 downto 0); --ore, minuti, secondi

13.

14. hour: out std\_logic\_vector(4 downto 0);

15. min: out std\_logic\_vector(5 downto 0);

16. sec: out std\_logic\_vector(5 downto 0);

17. fullCount : out std\_logic

18.

19.

20. );

21. end entity cronometro;

22.

23.

24.

25. architecture rtl of cronometro is

26.

27.

28. component contMod60 is port(

29. clk : in std\_logic;

30. set: in std\_logic;

31. input\_loaded: in std\_logic\_vector(5 downto 0);

32. reset : in std\_logic;

33.

34. cont: out std\_logic\_vector(5 downto 0);

35. fullCount: out std\_logic

36. );

37. end component;

38.

39.

40. component contMod24 port(

41. clk : in std\_logic;

42. set: in std\_logic;

43. input\_loaded: in std\_logic\_vector(4 downto 0);

44. reset : in std\_logic;

45.

46. cont: out std\_logic\_vector(4 downto 0);

47. fullCount: out std\_logic

48. );

49. end component;

50.

51. signal sectoMin: std\_logic;

52. signal minToHour: std\_logic;

53. signal hourToexit:std\_logic;

54.

55. signal secMidi: std\_logic\_vector(5 downto 0);

56. signal minMidi: std\_logic\_vector(5 downto 0);

57. signal hourMidi: std\_logic\_vector(4 downto 0);

58.

59. begin

60.

61. seconds: contMod60 port map(

62. clk=>clk,

63. set => set,

64. input\_loaded => input\_loaded(5 downto 0),

65. reset => reset,

66.

67. cont => secMidi,

68. fullCount=> sectoMin

69.

70. );

71.

72. mins: contMod60 port map(

73. clk=> sectoMin,

74. set=> set,

75. input\_loaded => input\_loaded(11 downto 6),

76. reset => reset,

77.

78. cont=> minMidi,

79. fullCount=>minToHour

80.

81. );

82.

83. hours: contMod24 port map(

84. clk=> minToHour,

85. set => set,

86. input\_loaded => input\_loaded(16 downto 12),

87. reset => reset,

88.

89. cont => hourMidi,

90. fullCount => hourtoExit

91. );

92.

93. hour <= hourMidi;

94. min<=minMidi;

95. sec<=secMidi;

96.

97. fullCount<= '1' when (hourMidi="10111" AND minMidi="111011" AND secMidi="111011")

AND rising\_edge(clk) else

98. '0';

99.

100.

101. end architecture rtl;

102.

Il cronometro è formato per composizione di due contatori **modulo60** e un contatore **modulo24,** Inoltre, è presente la gestione del **fullCount** esterno che quando raggiunge *23 ore, 59 minuti e 59 secondi,* e abbiamo un***rising\_edge*** *del clock (di cui il suo corrispettivo* ***falling\_edge*** *corrisponde al 60-esimo secondo)* allora **fullCount** è posto alto, cioè “*mezzo-secondo”* prima che *ore, minuti e secondi,* diventino 0.

#### Simulazione

Di seguito è riportato il testbench utilizzato per testare il **Cronometro.**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5. entity tbCron is

6. end entity tbCron;

7.

8. architecture tb of tbCron is

9. signal clk : std\_logic := '0';

10. signal reset : std\_logic := '0';

11. signal set : std\_logic := '0';

12. signal input\_loaded : std\_logic\_vector(16 downto 0) := (others => '0');

13. signal hour : std\_logic\_vector(4 downto 0);

14. signal min : std\_logic\_vector(5 downto 0);

15. signal sec : std\_logic\_vector(5 downto 0);

16. signal fullCount : std\_logic;

17.

18. component cronometro is

19. port (

20. clk : in std\_logic;

21. reset: in std\_logic;

22. set: in std\_logic;

23. input\_loaded: in std\_logic\_vector(16 downto 0);

24. hour: out std\_logic\_vector(4 downto 0);

25. min: out std\_logic\_vector(5 downto 0);

26. sec: out std\_logic\_vector(5 downto 0);

27. fullCount : out std\_logic

28. );

29. end component cronometro;

30.

31. begin

32. dut: cronometro port map (

33. clk => clk,

34. reset => reset,

35. set => set,

36. input\_loaded => input\_loaded,

37. hour => hour,

38. min => min,

39. sec => sec,

40. fullCount => fullCount

41. );

42.

43. clk\_process :process

44. begin

45. clk <= '0';

46. wait for 3 ns;

47. clk <= '1';

48. wait for 3 ns;

49. end process;

50.

51. stim\_proc: process

52. begin

53. -- Aggiungi qui la tua sequenza di stimoli

54. wait for 30 ns;

55. input\_loaded <= "10111111010110000";

56. set <= '1';

57. wait for 3 ns;

58. set <= '0';

59. wait for 880 ns;

60. reset<= '1';

61. wait for 3 ns;

62. reset <= '0';

63.

64.

65.

66. wait;

67. end process;

68.

69. end architecture tb;

70.

**Forme d’onda**

**Generica**

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

Descrizione generata automaticamente**

**Focus sull’operazione di Set**

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

Descrizione generata automaticamente**

All’alzarsi del segnale di set, viene prontamemte caricato il valore posto in ***input\_loaded,*** assegnando opportunamente i *secondi, i minuti e le ore.* Si nota anche come alla fine del conteggio, **fullCount si alzi** e sul *falling\_edge* del clock si abbassi e si azzeri il conteggio.

**Focus sull’operazione di reset**

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

Descrizione generata automaticamente**

Qui è possibile apprezzare come l’operazione di reset *asincrono* azzeri il conteggio facendolo ripartire.

­

## Esercizio 5.2: Implementazione su board del componente 5.1

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).

#### Progetto e architettura

#### Implementazione

#### Simulazione

## Esercizio 5.3: Estensione del componente sviluppato nei punti precedenti.

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).

#### Progetto e architettura

#### 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;

## 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;

## Flip-Flop D

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity ffd is

8. port (

9. D : in std\_logic;

10. clk: in std\_logic;

11. reset: in std\_logic;

12.

13. Q : out std\_logic

14. );

15. end entity ffd;

16.

17.

18. architecture comportamentale of ffd is

19.

20. begin

21.

22. proc\_name: process(clk)

23. begin

24. if rising\_edge(clk) then

25. if reset = '1' then

26. Q<='0'; --reset sincrono

27. else

28. Q<=D;

29. end if;

30.

31. end if;

32. end process proc\_name;

33.

34.

35. end architecture comportamentale;

## 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>