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 5](#_Toc158815678)

[Esercizio 1.1: Multiplexer 16:1 5](#_Toc158815679)

[Progetto e architettura 5](#_Toc158815680)

[Implementazione 5](#_Toc158815681)

[Simulazione 7](#_Toc158815682)

[Esercizio 1.2: Rete di interconnessione 16:4 8](#_Toc158815683)

[Progetto e architettura 8](#_Toc158815684)

[Implementazione 9](#_Toc158815685)

[Simulazione 10](#_Toc158815686)

[Esercizio 1.3: Implementazione su board dell’esercizio 1.2 12](#_Toc158815687)

[Sintesi su board di sviluppo 12](#_Toc158815688)

[Esercizio 2.1: ROM combinatoria con sistema M. 18](#_Toc158815689)

[Progetto e architettura 18](#_Toc158815690)

[Implementazione 18](#_Toc158815691)

[Simulazione 20](#_Toc158815692)

[Esercizio 2.2: Implementazione su board della ROM combinatoria con sistema M. 21](#_Toc158815693)

[Sintesi su board di sviluppo 21](#_Toc158815694)

[Capitolo 2: Reti Sequenziali Elementari 23](#_Toc158815695)

[Esercizio 3.1: Riconoscitore di sequenza 101 (senza sovrapposizione/sovrapposizione parziale). 23](#_Toc158815696)

[Progetto e architettura 23](#_Toc158815697)

[Implementazione 24](#_Toc158815698)

[Simulazione 26](#_Toc158815699)

[Esercizio 3.2: Implementazione su board del riconoscitore di sequenza. 28](#_Toc158815700)

[Sintesi su board di sviluppo 28](#_Toc158815701)

[Timing analysis 32](#_Toc158815702)

[Esercizio 4.1: Shift Register. 33](#_Toc158815703)

[Progetto e architettura 33](#_Toc158815704)

[Implementazione 35](#_Toc158815705)

[Simulazione 39](#_Toc158815706)

[Esercizio 5.1: Cronometro 44](#_Toc158815707)

[Progetto e architettura 44](#_Toc158815708)

[Implementazione 46](#_Toc158815709)

[Simulazione 51](#_Toc158815710)

[Esercizio 5.2: Implementazione su board del componente 5.1 53](#_Toc158815711)

[Sintesi su board di sviluppo 54](#_Toc158815712)

[Esercizio 5.3: Estensione del componente sviluppato nei punti precedenti. 58](#_Toc158815713)

[Sintesi su board di sviluppo 58](#_Toc158815714)

[Esercizio 6.1: Sistema di lettura-elaborazione-scrittura PO\_PC 60](#_Toc158815715)

[Progetto e architettura 60](#_Toc158815716)

[Implementazione 61](#_Toc158815717)

[Simulazione 66](#_Toc158815718)

[Esercizio 6.2: Implementazione su board dell’esercizio 6.1 67](#_Toc158815719)

[Sintesi su board di sviluppo 67](#_Toc158815720)

[Capitolo 3: Macchine Aritmetiche 74](#_Toc158815721)

[Esercizio 7.1: Moltiplicatore di Booth 74](#_Toc158815722)

[Progetto e architettura 74](#_Toc158815723)

[Implementazione 75](#_Toc158815724)

[Simulazione 77](#_Toc158815725)

[Esercizio 7.2: Sintesi del Moltiplicatore di Booth 79](#_Toc158815726)

[Sintesi su board di sviluppo 79](#_Toc158815727)

[Capitolo 4: Comunicazione con Handshaking 81](#_Toc158815728)

[Esercizio 8.1: 81](#_Toc158815729)

[Progetto e architettura 81](#_Toc158815730)

[Implementazione 83](#_Toc158815731)

[Simulazione 90](#_Toc158815732)

[Capitolo 5: Processore 93](#_Toc158815733)

[Esercizio 9 93](#_Toc158815734)

[Punto A: Analisi dell’architettura, di due istruzioni a scelta e simulazione. 93](#_Toc158815735)

[Punto B: Modifica di un codice operativo. 103](#_Toc158815736)

[Capitolo 6: Interfaccia seriale 105](#_Toc158815737)

[Esercizio 10 105](#_Toc158815738)

[Progetto e architettura 105](#_Toc158815739)

[Implementazione 105](#_Toc158815740)

[Simulazione 109](#_Toc158815741)

[Capitolo 7: Switch multistadio 110](#_Toc158815742)

[Esercizio 11.1 110](#_Toc158815743)

[Progetto e architettura 110](#_Toc158815744)

[Implementazione 111](#_Toc158815745)

[Simulazione 113](#_Toc158815746)

[Appendice 117](#_Toc158815747)

[Multiplexer 2:1 117](#_Toc158815748)

[Multiplexer 4:1 117](#_Toc158815749)

[Demultiplexer 1:4 118](#_Toc158815750)

[Flip Flop T 118](#_Toc158815751)

[Flip-Flop D 119](#_Toc158815752)

[ROM da 16 locazioni da 8 bit 120](#_Toc158815753)

[Sistema M 120](#_Toc158815754)

[Contatore Modulo 16 121](#_Toc158815755)

[Contatore comportamentale 122](#_Toc158815756)

[Progetto e architettura 123](#_Toc158815757)

[Implementazione 123](#_Toc158815758)

# 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

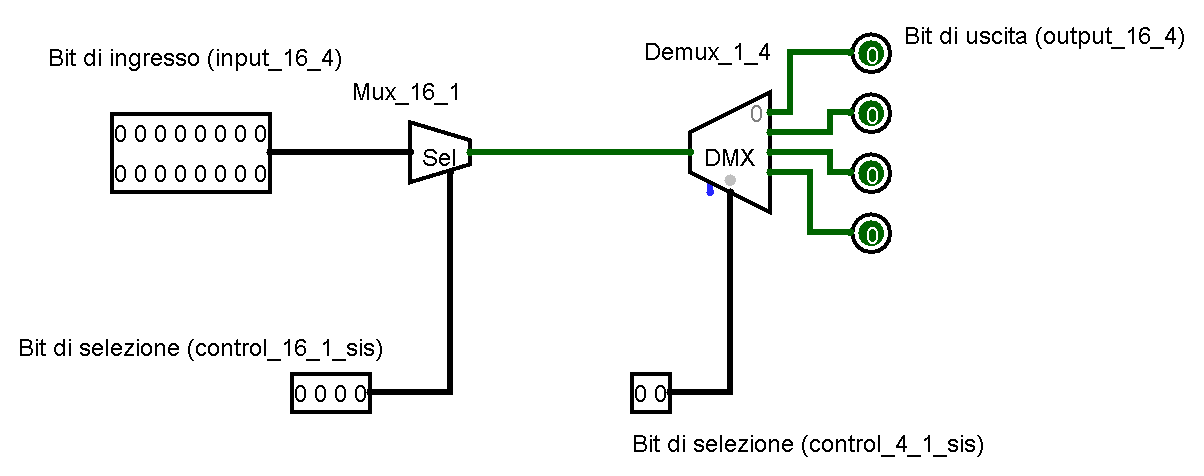
Descrizione generata automaticamente**

## 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.

#### Sintesi su board di sviluppo

Per permettere la sintesi su board dell’esercizio 1.2 abbiamo implementato un ulteriore componente chiamato *loader* che permette a seconda della pressione di due bottoni, di caricare il valore assunto dagli switch o nei primi otto bit (da **0 a 7**) di uscita o nei secondi otto (da **8 a 15**).

Immagine che contiene diagramma, schizzo, Piano, Disegno tecnico

Descrizione generata automaticamente

Riportiamo di seguito l’implementazione del componente ***loader***:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity loader is

port (

input\_switch : in std\_logic\_vector(0 to 7);

button1: in std\_logic;

button2: in std\_logic;

clk: in std\_logic;

output\_switch: out std\_logic\_vector(0 to 15)

);

end entity loader;

architecture rtl of loader is

begin

process (clk)

begin

if rising\_edge(clk) then

if(button1='1' AND button2='0') then

output\_switch(0 to 7) <= input\_switch;

elsif(button1='0' AND button2='1') then

output\_switch(8 to 15) <= input\_switch;

end if;

end if;

end process;

end architecture rtl;

Nel *top-module* ***Sistema*** abbiamo aggiunto e collegato opportunamente il componente loader:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity Sistema is

port (

clk: in std\_logic;

button1\_sis: in std\_logic;

button2\_sis: in std\_logic;

input\_16\_4 : in std\_logic\_vector(0 to 7); --ingressi dagli switch

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

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

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

);

end entity Sistema;

architecture structural of Sistema is

signal output : std\_logic;

signal loadertoMux: std\_logic\_vector(0 to 15);

component loader port (

input\_switch : in std\_logic\_vector(0 to 7);

button1: in std\_logic;

button2: in std\_logic;

clk: in std\_logic;

output\_switch: out std\_logic\_vector(0 to 15)

);

end component;

component mux\_16\_1 port(

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

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

y\_16\_1: out STD\_LOGIC

);

end component;

component Demux\_1\_4 port(

input\_1\_4: in STD\_LOGIC;

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

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

);

end component;

begin

load: loader port map(

input\_switch(0)=> input\_16\_4(0),

input\_switch(1)=> input\_16\_4(1),

input\_switch(2)=> input\_16\_4(2),

input\_switch(3)=> input\_16\_4(3),

input\_switch(4)=> input\_16\_4(4),

input\_switch(5)=> input\_16\_4(5),

input\_switch(6)=> input\_16\_4(6),

input\_switch(7)=> input\_16\_4(7),

button1=> button1\_sis,

button2=> button2\_sis,

clk=> clk,

output\_switch(0) => loadertoMux(0),

output\_switch(1) => loadertoMux(1),

output\_switch(2) => loadertoMux(2),

output\_switch(3) => loadertoMux(3),

output\_switch(4) => loadertoMux(4),

output\_switch(5) => loadertoMux(5),

output\_switch(6) => loadertoMux(6),

output\_switch(7) => loadertoMux(7),

output\_switch(8) => loadertoMux(8),

output\_switch(9) => loadertoMux(9),

output\_switch(10) => loadertoMux(10),

output\_switch(11) => loadertoMux(11),

output\_switch(12) => loadertoMux(12),

output\_switch(13) => loadertoMux(13),

output\_switch(14) => loadertoMux(14),

output\_switch(15) => loadertoMux(15)

);

mux: mux\_16\_1 port map(

input\_16\_1(0) => loadertoMux(0),

input\_16\_1(1) => loadertoMux(1),

input\_16\_1(2) => loadertoMux(2),

input\_16\_1(3) => loadertoMux(3),

input\_16\_1(4) => loadertoMux(4),

input\_16\_1(5) => loadertoMux(5),

input\_16\_1(6) => loadertoMux(6),

input\_16\_1(7) => loadertoMux(7),

input\_16\_1(8) => loadertoMux(8),

input\_16\_1(9) => loadertoMux(9),

input\_16\_1(10) => loadertoMux(10),

input\_16\_1(11) => loadertoMux(11),

input\_16\_1(12) => loadertoMux(12),

input\_16\_1(13) => loadertoMux(13),

input\_16\_1(14) => loadertoMux(14),

input\_16\_1(15) => loadertoMux(15),

--output e controllo

y\_16\_1 => output,

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

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

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

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

);

demux: Demux\_1\_4 port map(

input\_1\_4 => output,

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

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

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

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

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

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

);

end architecture structural;

Infine, riportiamo il file di *constraint* utilizzato:

## This file is a general .xdc for the Nexys A7-50T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { control\_1\_4\_sis[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { control\_1\_4\_sis[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { control\_16\_1\_sis[0] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { control\_16\_1\_sis[1] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { control\_16\_1\_sis[2] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { control\_16\_1\_sis[3] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

#set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { SW[6] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

#set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { SW[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { input\_16\_4[0] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { input\_16\_4[1] }]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[2] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[3] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[4] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[5] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[6] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[7] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { output\_16\_4[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { output\_16\_4[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { output\_16\_4[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { output\_16\_4[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

#set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { LED[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

#set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { LED[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

#set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { LED[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

#set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { LED[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

#set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { LED[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

#set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { LED[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { LED[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

#set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { LED[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

#set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { LED[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

#set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { LED[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

#set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { LED[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

#set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { LED[15] }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

## RGB LEDs

#set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_B }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

#set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_G }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

#set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_R }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

#set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_B }]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

#set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_G }]; #IO\_0\_14 Sch=led17\_g

#set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_R }]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##7 segment display

#set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { CA }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

#set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { CB }]; #IO\_25\_14 Sch=cb

#set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { CC }]; #IO\_25\_15 Sch=cc

#set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { CD }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

#set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { CE }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

#set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { CF }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

#set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { CG }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

#set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { DP }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

#set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { AN[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

#set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { AN[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

#set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { AN[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

#set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { AN[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

#set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { AN[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

#set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { AN[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

#set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { AN[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

#set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { AN[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

#set\_property -dict { PACKAGE\_PIN C12 IOSTANDARD LVCMOS33 } [get\_ports { CPU\_RESETN }]; #IO\_L3P\_T0\_DQS\_AD1P\_15 Sch=cpu\_resetn

#set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { BTNC }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

#set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { BTNU }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { button1\_sis }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { button2\_sis }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

#set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { BTND }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

…

I bottoni sono stati utilizzati per scegliere se caricare gli input nella prima parte o nella seconda parte degli ingressi del *multiplexer 16\_1 (****button1\_sis*** *per la prima parte,* ***button2\_sis*** *per la seconda).* I primi due *switch (****J15, L16****)* sono stati utilizzati come selezionatore del *demux\_1\_4,* invece quelli da **M13** a **T18** sono stati usati per il selzionatore del *multiplexer\_16\_1.* Il resto degli switch è stato utilizzato per immettere l’input. Infine, i led sono stati utilizzati per visualizzare *l’output.*

## 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**](#_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.

#### Sintesi su board di sviluppo

Per la sintesi dell’esercizio 2.1 non è stato necessario aggiungere ulteriori componenti al design dell’esercizio sviluppato precedentemente. Di seguito riportiamo il file di *constraint*:

## This file is a general .xdc for the Nexys A7-50T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

#set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

#create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { controller\_M\_sis}]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { controller\_mem\_sis[0] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { controller\_mem\_sis[1] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { controller\_mem\_sis[2] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { controller\_mem\_sis[3] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

#set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { control\_16\_1\_sis[3] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

#set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { SW[6] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

#set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { SW[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

#set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { input\_16\_4[0] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

#set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { input\_16\_4[1] }]; #IO\_25\_34 Sch=sw[9]

#set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[2] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

#set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[3] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

#set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[4] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

#set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[5] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

#set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[6] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

#set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { input\_16\_4[7] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { output\_sis[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { output\_sis[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { output\_sis[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { output\_sis[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

#set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { LED[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

#set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { LED[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

#set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { LED[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

#set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { LED[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

#set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { LED[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

#set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { LED[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { LED[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

#set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { LED[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

#set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { LED[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

#set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { LED[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

#set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { LED[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

#set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { LED[15] }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

…

Con il primo switch (**J15**) riusciamo a controllare il **sistemaM,** con i restanti switch posso scegliere l’indirizzo di memoria da prelevare e mettere in uscita. Con i led visualizzo i valori posti in uscita.

# 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),

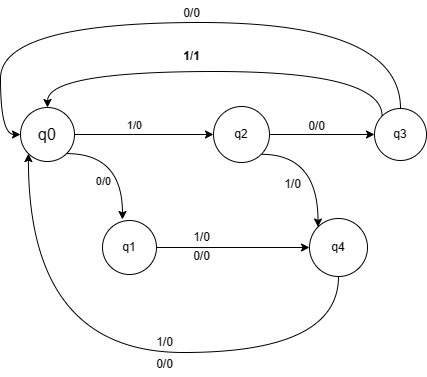
- se M=1, la macchina valuta i bit seriali in ingresso uno alla volta, tornando allo stato iniziale ogni volta che la sequenza viene correttamente riconosciuta (sequenze parzialmente sovrapposte).

#### Progetto e architettura

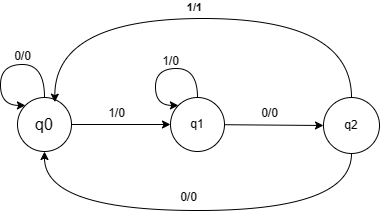
L’esercizio in questione richiede di implementare un riconoscitore di sequenza **101**, che tramite un flag M si comporti come un riconoscitore *con sovrapposizione parziale* oppure *senza sovrapposizione.*

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

**Senza sovrapposizione**

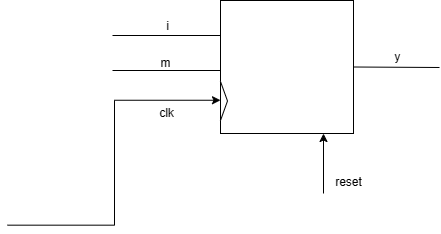
****Questa evoluzione della macchina a stati permette di valutare i bit in ingresso a gruppi di tre.

**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

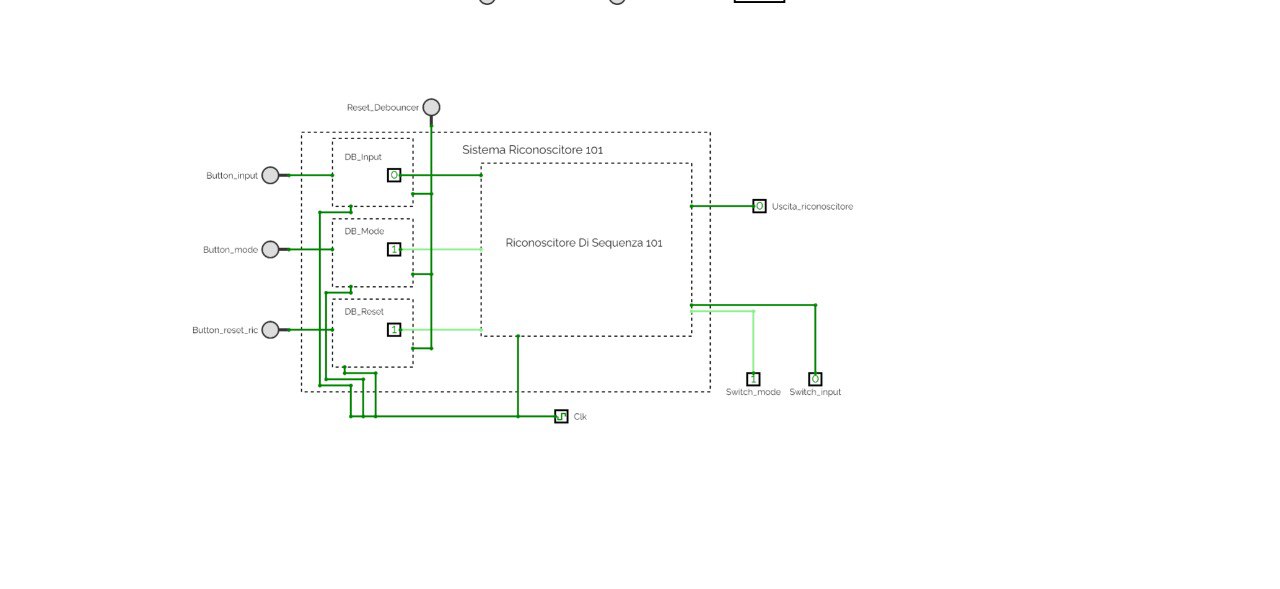
Descrizione generata automaticamente**

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.

#### Sintesi su board di sviluppo

Per la sintesi su board del riconoscitore di sequenza abbiamo dovuto utilizzare il *Debouncer* già fornito nel materiale didattico per permettere di “pulire” il segnale generato dal bottone quando viene premuto e rilasciato, consentendo inoltre di non essere acquisito di continuo se premuto per lungo tempo.

Dallo schema, si denota come abbiamo inserito tre componenti *Debouncer:*

1. **DB\_Input:** per il pulsante che acquisice l’input.
2. **DB\_Mode:** per il pulsante che permette di acquisire la modalità.
3. **DB\_Reset:** per il pulsante che permette di resettare il riconoscitore.

L’implementazione del *top\_module* cambia nel seguente modo:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity sistema\_recog is port(

bottone1\_sis: in std\_logic; --bottone input

bottone2\_sis: in std\_logic;--bottone modo

bottone3\_sis: in std\_logic; --bottone reset

bottone\_reset\_debouncer: in std\_logic;

clk: in std\_logic;

switchIN: in std\_logic;

switchMODE: in std\_logic;

led\_uscita: out std\_logic

);

end sistema\_recog;

architecture rtl of sistema\_recog is

component riconoscitoristotsparz is port(

y : out std\_logic;

i : in std\_logic;

clk : in std\_ulogic;

reset: in std\_logic;

m: in std\_logic;

button1\_ric: in std\_logic;

button2\_ric: in std\_logic

);

end component;

component ButtonDebouncer is

generic (

CLK\_period: integer := 10; -- periodo del clock (della board) in nanosecondi

btn\_noise\_time: integer := 10000000 -- durata stimata della oscillazione del bottone in nanosecondi

-- il valore di default è 10 millisecondi

);

port(

RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

BTN : in STD\_LOGIC;

CLEARED\_BTN : out STD\_LOGIC

);

end component;

signal debouncer\_to\_bottone\_recog: std\_logic\_vector(0 TO 2); --vettore di fili per completare i collegamenti tra i debouncer e il riconoscitore

begin

input: ButtonDebouncer port map(

RST => bottone\_reset\_debouncer,

CLK => clk,

BTN => bottone1\_sis,

CLEARED\_BTN => debouncer\_to\_bottone\_recog(0)

);

mode: ButtonDebouncer port map(

RST => bottone\_reset\_debouncer,

CLK => clk,

BTN => bottone2\_sis,

CLEARED\_BTN => debouncer\_to\_bottone\_recog(1)

);

reset: ButtonDebouncer port map(

RST => bottone\_reset\_debouncer,

CLK => clk,

BTN => bottone3\_sis,

CLEARED\_BTN => debouncer\_to\_bottone\_recog(2)

);

recog: riconoscitoristotsparz port map(

y => led\_uscita,

i => switchIN,

clk => clk,

reset => debouncer\_to\_bottone\_recog(2),

m => switchMODE,

button1\_ric=> debouncer\_to\_bottone\_recog(0),

button2\_ric=> debouncer\_to\_bottone\_recog(1)

);

end architecture rtl;

Infine, il file di *constraint* è impostato come segue:

## This file is a general .xdc for the Nexys A7-50T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

##Switches

#set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { SW[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

#set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { SW[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

#set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { SW[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

#set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { SW[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

#set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { SW[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

#set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { SW[5] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

#set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { SW[6] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

#set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { SW[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

#set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { SW[8] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

#set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { SW[9] }]; #IO\_25\_34 Sch=sw[9]

#set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { SW[10] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

#set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { SW[11] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

#set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { SW[12] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

#set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { SW[13] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { switchMODE }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { switchIN }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

#set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { bottone3\_sis }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

#set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { bottone\_reset\_debouncer }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

#set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { debouncer\_to\_bottone\_recog[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

#set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { LED[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

#set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { LED[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

#set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { LED[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

#set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { LED[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

#set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { LED[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

#set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { LED[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

#set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { LED[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { LED[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

#set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { LED[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

#set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { LED[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

#set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { LED[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

#set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { LED[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { led\_uscita }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

## RGB LEDs

#set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_B }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

#set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_G }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

#set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_R }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

#set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_B }]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

#set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_G }]; #IO\_0\_14 Sch=led17\_g

#set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_R }]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##7 segment display

#set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { CA }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

#set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { CB }]; #IO\_25\_14 Sch=cb

#set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { CC }]; #IO\_25\_15 Sch=cc

#set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { CD }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

#set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { CE }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

#set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { CF }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

#set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { CG }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

#set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { DP }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

#set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { AN[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

#set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { AN[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

#set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { AN[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

#set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { AN[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

#set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { AN[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

#set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { AN[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

#set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { AN[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

#set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { AN[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

#set\_property -dict { PACKAGE\_PIN C12 IOSTANDARD LVCMOS33 } [get\_ports { CPU\_RESETN }]; #IO\_L3P\_T0\_DQS\_AD1P\_15 Sch=cpu\_resetn

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { bottone1\_sis }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { bottone2\_sis }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { bottone3\_sis }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { bottone\_reset\_debouncer }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

#set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { BTND }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

Gli switch **U11** e **V10** sono utilizzati rispettivamente per la *modalità* e l’*input.* L’ultimo led (**V11**) è stato utilizzato per visualizzare se la sequenza è stata riconosciuta. Infine, il bottone **N17** è stato usato per campinare l’input, **M18** è stato utilizzato per selezionare la modalità, e gli ultimi due per i rispettivi *reset.*

#### Timing analysis

Per effettuare la timing analysis abbiamo prima definito tale constraint sul clock:

## Clock signal

set\_property -dict {PACKAGE\_PIN E3 IOSTANDARD LVCMOS33} [get\_ports {clk}]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

In tal caso, avendo specificato una sorgente per il clock, esso non è ***virtual*** ma è proprio il clock della board fornita: viene infatti chiamato ***primary clock*** ed ha un periodo di 10 nanosecondi e un duty cycle pari al 50%.

**Definizioni utili**

* **SLACK**: Si definisce *slack* la differenza tra il tempo richiesto per le uscite e il tempo di arrivo degli ingressi;
* **TEMPO DI SETUP**: è il tempo che impiega il segnale di input per stabilizzarsi *prima del prossimo fronte attivo del clock*, al fine di evitare errori di sincronizzazione;
* **TEMPO DI HOLD**: è il tempo richiesto al segnale di input affinché sia stabile *dopo il fronte del clock* per evitare problemi di sincronizzazione.

Con i parametri definiti nello script di cui sopra, abbiamo ottenuto questi parametri temporali del sistema in esame:

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

Come si evince dall’immagine ottenuta dall’analisi temporale, il tempo di setup risulta essere abbastanza alto (ciò vuol dire che i segnali devono stabilizzarsi 5,4 nanosecondi prima del prossimo fronte del clock).

Per poter ridurre tale parametro e velocizzare il sistema, andiamo a dimezzare il periodo del clock e contestualmente anche il parametro ***waveform*** (ovvero quel parametro in cui si specificano gli istanti in cui si presentano i fronti di salita e di discesa in un periodo del clock) dei constraint del clock:

## Clock signal

set\_property -dict {PACKAGE\_PIN E3 IOSTANDARD LVCMOS33} [get\_ports {clk}]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 5.00 -waveform {0 2.5} [get\_ports {clk}];

Da queste modifiche, segue la nuova timing analysis:

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

Si può notare che il tempo di setup è passato da circa 5 nanosecondi a circa 0,5 nanosecondi, aumentando la velocità del sistema, dato che i segnali devono stabilizzarsi più in prossimità del prossimo fronte di clock.

## 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.

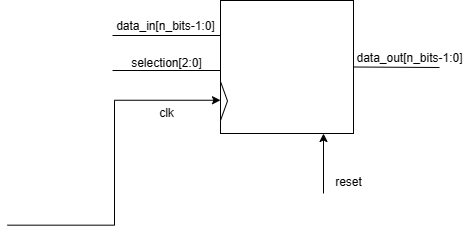
Nota: il numero di bit del registro deve essere implementato come un generic, e dall’esterno deve poter essere scelta la modalità di funzionamento mediante opportuni segnali di selezione.

#### 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).

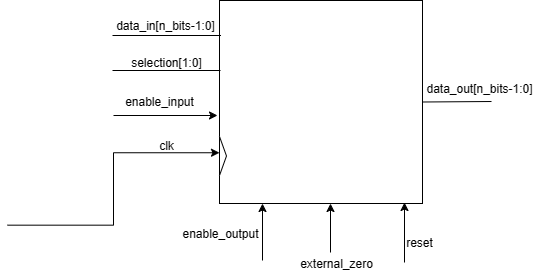
La **selection** da tre bit permette allo shift register di avere tre modalità di selezione (*000, 101, 111*) in più che permettono di porre il dato di ingresso, direttamente in uscita senza alcuno shift.

**Approccio strutturale**

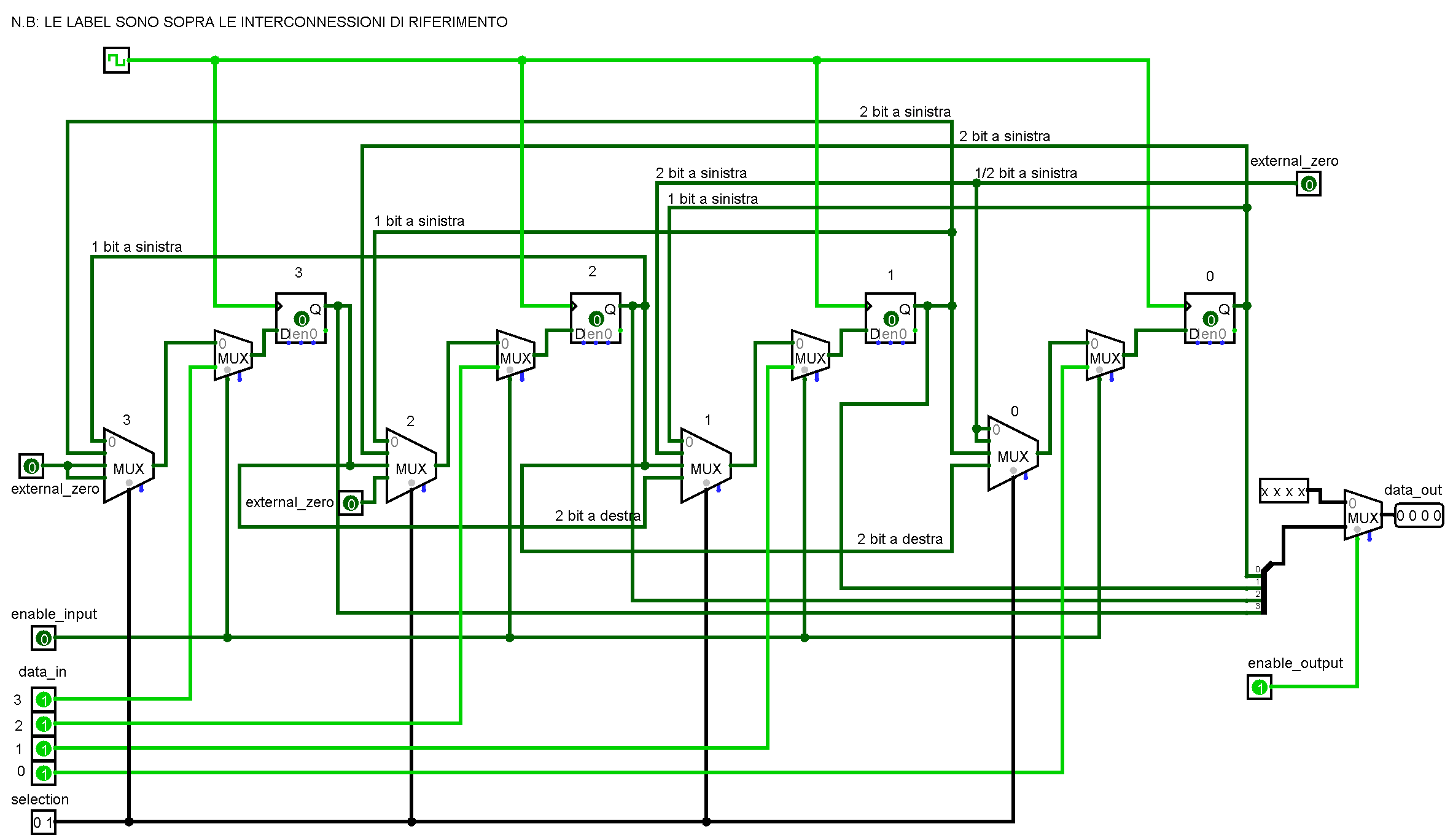
Per l’approccio strutturale, abbiamo bisogno di implementare singolarmente *n\_bits* flip-flop e una serie di multiplexer per permettere di realizzare le funzioni richieste. Nello specifico, abbiamo scelto di utilizzare **flip-flop D, multiplexer 4:1** (*abbiamo utilizzato una versione leggermente modificata 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

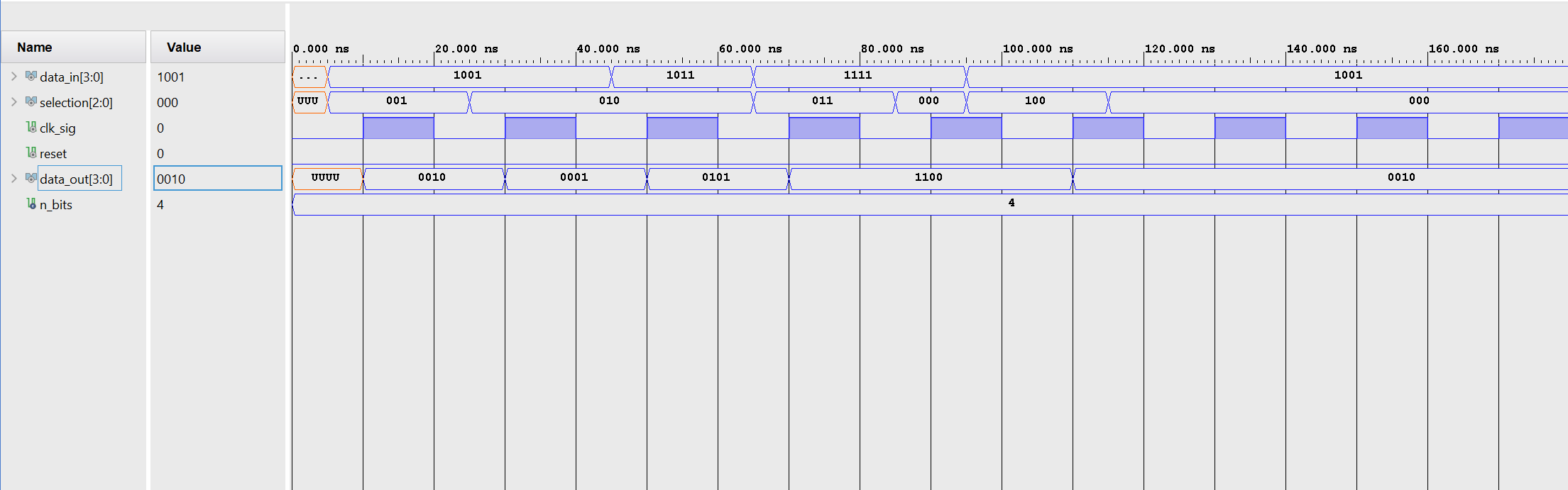
75. wait;

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.

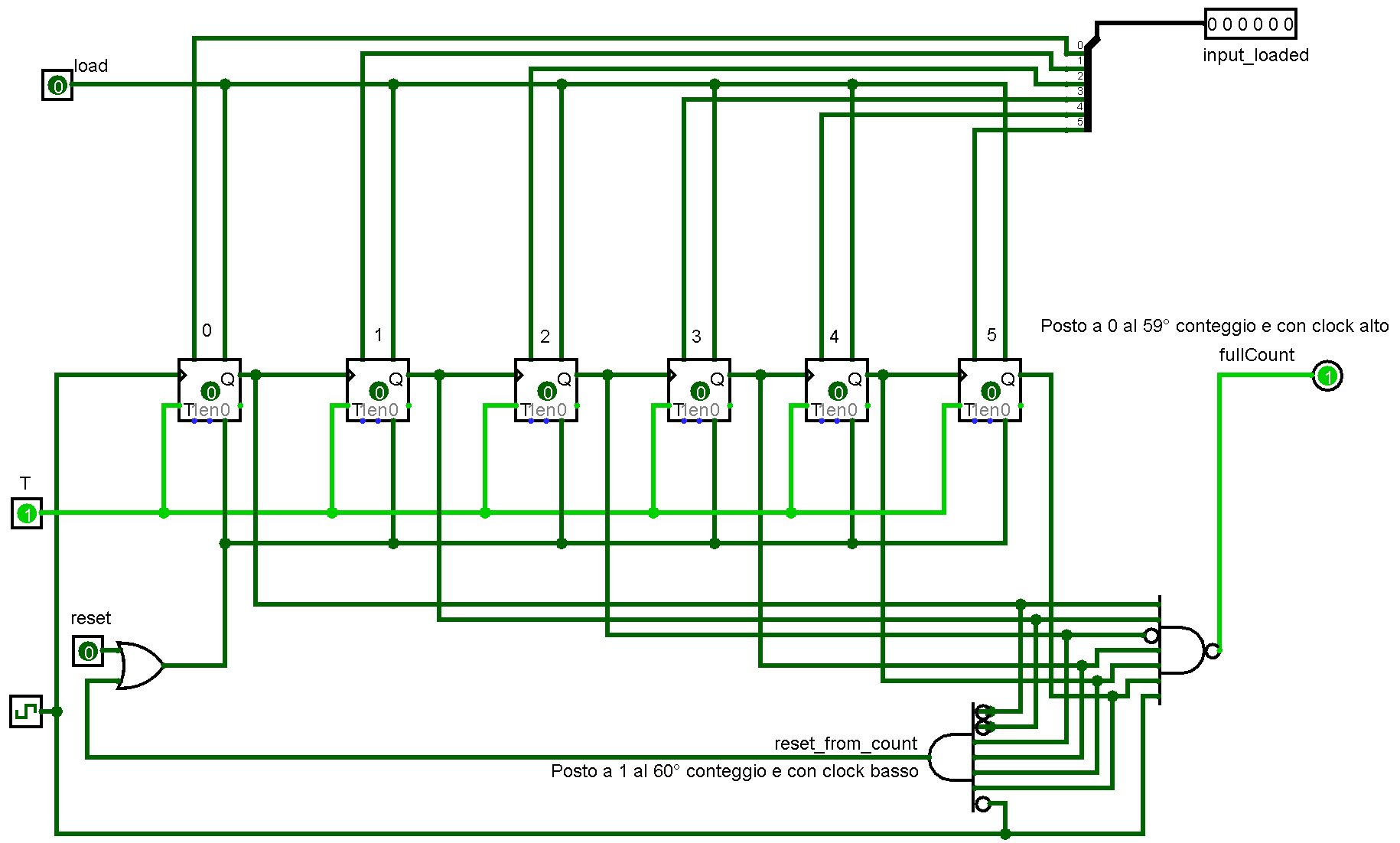
Inoltre, per tutti e due gli approcci, il conteggio deve essere attivato sul fronte di *discesa* per fare in modo che il contatore o la serie di contatori funzionino adeguatamente (Se volessi contare in maniera decrescente potrei decrementare il conteggio sul fronte di *salita* del segnale).

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).

#### Sintesi su board di sviluppo

Abbiamo aggiunto delle componenti a partire dall’esercizio 5.1 per sintetizzare correttamente il cronometro su board il cronometro. In particolare:

1. Un seven-segment-display-manager, per gestire il display a sette segmenti
2. Un Loader, che permette di caricare un tempo iniziale da cui far partire il cronometro
3. Un Clock Filter, che diminuisce la frequenza di clock della board
4. Vari Button Debouncer, per “pulire” l’input fornito dai bottoni della board
5. Un Multiplexer enable, per gestire lo stop e l’inizio del conteggio del cronometro
6. Una Control Unit, che è stata modificata dall’esercizio 5.1

NB: Sono stati utilizzati dei contatori comportamentali (consultabile in [*Appendice*](#_Contatore_comportamentale)) nella struttura del cronometro, il comportamento è analogo al cronometro dell’esercizio 5.1

Immagine che contiene schermata, diagramma

Descrizione generata automaticamenteI bottoni utilizzati sono quattro, Button Start, Button Mode, Button Load e Reset. Lo start serve per iniziare il conteggio, mentre il Button Mode serve per cambiare modalità e ci permette di immettere il valore da cui iniziare a contare. Il bottone per il load funziona solamente quando siamo in modalità di caricamento, esso serve per caricare in sequenza prima i secondi, poi i minuti e infine le ore. Il bottone di Reset porta il cronometro un uno stato idle, fermo nel conteggio, a partire dal valore zero nel cronometro.

Il componente più rilevante è proprio il ***seven-segment-display-manager***, il quale ci è stato fornito durante il corso. Esso prende in ingresso un vettore di 32 bit, che gestirà l’accensione dei catodi delle varie cifre del display. Collegando opportunamente il valore di conteggio del contatore precedentemente visto all’esercizio 5.1 all’ingresso del ***seven-segment-display-manager*** otterremo l’effetto di conteggio desiderato (è stata assunta una codifica esadecimale per il conteggio).

Il ***Loader*** è un componente che prende in ingresso i valori collegati a 6 switch della board e li immette nel cronometro. E’ a tutti gli effetti un mutliplexer con tre uscite di dimensioni diverse (6 bit per i secondi e i minuti, 5 bit per le ore). Il valore sarà caricato nel cronometro quando la CU manderà un segnale di set allo stesso cronometro.

Il ***Clock Filter*** prende in ingresso il clock della board e ne abbassa la frequenza, mandando quindi in uscita un segnale che utilizzeremo come ***enable*** per il cronometro. Per fermare il conteggio è stato utilizzato un multiplexer enable che prende in ingresso tale segnale di ***enable*** e un segnale costante con valore logico 0.

Per le implementazioni abbiamo seguito un approccio:

1. Strutturale per il seven-segment-display-manager.
2. Dataflow per il Loader.
3. Dataflow per il Multiplexer enable.
4. Comportamentale per la Control Unit.
5. Comportamentale per il Clock Filter.
6. Comportamentale per i Debouncer.

Il ***seven-segment-display-manager*** è composto a sua volta da un ***anodes manager*** e un ***cathodes manager***, ovvero componenti che gestiscono il valore degli anodi sulla scheda (quali cifre accendere) e dei catodi (che combinazione di segmenti accendere). Dato che gli anodi sono divisi per cifre ma i catodi invece sono condivisi, il display manager accende e spegne ogni cifra con una combinazione di catodi diversa. Questo processo avviene ad una frequenza elevata, non apprezzabile dall’occhio umano, creando questo effetto per il quale le cifre sono tutte accese con combinazioni diverse.

**Loader**

La selezione del loader è gestita dalla Control Unit, ed è un componente puramente combinatorio.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity loader is

port (

input\_switch : in std\_logic\_vector(5 downto 0);

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

output\_switch: out std\_logic\_vector(16 downto 0)

);

end entity loader;

architecture rtl of loader is

begin

output\_switch(16 downto 12)<= input\_switch(4 downto 0) when selection\_input="01"; -- sto caricando le ore

output\_switch(11 downto 6)<= input\_switch when selection\_input="10"; --sto caricando i minuti

output\_switch(5 downto 0)<= input\_switch when selection\_input="11"; --sto caricando i secondi

end architecture rtl;

**Control Unit**

Abbiamo modificato la Control Unit del 5.1. Essa adesso prende in ingresso i segnali dei bottoni, filtrati attraverso i debouncer e da segnali di abilitazione conteggio, segnale di set per il cronometro e segnali di selezione caricamento per il Loader.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity ControlUnit is

port (

clk: in std\_logic;

button\_mode: in std\_logic;

button\_reset : in std\_logic;

button\_start\_count: in std\_logic;

button\_load\_input: in std\_logic;

start\_count: out std\_logic; --diventa un segnale di selezione del multiplexer che permette di selezionare enable oppure uno 0.

offDisplay: out std\_logic;

setCron: out std\_logic; --segnale di set per caricare gli ingressi dagli switch

selLoader: out std\_logic\_vector(1 downto 0)

);

end entity ControlUnit;

architecture rtl of ControlUnit is

TYPE stati is (idle, caricaSec, caricaMin, caricaHour, setLoadedInput, startCronometro);

signal stato\_corrente: stati := idle;

begin

process (clk)

begin

if rising\_edge(clk) then

if(button\_reset='1') then

stato\_corrente<=idle;

start\_count<='0'; --mux settato a 0, non vado avanti nel conteggio

--selLoader<="00"; --input dei contatori tutti a 0

setCron<='0';

offDisplay<='0';

end if;

if(stato\_corrente=idle AND button\_mode='1') then

stato\_corrente <= caricaSec;

selLoader<="11"; --carico i secondi

elsif(stato\_corrente=caricaSec AND button\_load\_input='1') then

stato\_corrente<=caricaMin;

selLoader<="10";

elsif(stato\_corrente=caricaMin AND button\_load\_input='1') then

stato\_corrente<=caricaHour;

selLoader<="01";

elsif (stato\_corrente=caricaHour AND button\_load\_input='1') then

stato\_corrente<= setLoadedInput;

elsif(stato\_corrente=setLoadedInput AND button\_load\_input='1') then

setCron<='1';

stato\_corrente<=idle;

elsif (stato\_corrente=startCronometro AND button\_mode='1') then

start\_count<='0';

stato\_corrente<=idle;

elsif (stato\_corrente=idle AND button\_start\_count='1') then

setCron<='0';

stato\_corrente<=startCronometro;

start\_count<='1';

end if;

end if;

end process;

end architecture rtl;

## 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).

#### Sintesi su board di sviluppo

Così come abbiamo fatto per il 5.2, elenchiamo i componenti aggiuntivi che sono stati aggiunti nel 5.3. La funzionalità aggiuntiva è quella di salvare degli intertempi e, facoltativamente, mostrarli a schermo. Per fare ciò sono stati aggiunti:

1. Una Memoria (MEM), che salva gli intertempi.
2. Un Contatore, che scandisce gli indirizzi in cui scrivere nella memoria.
3. Un Multiplexer Display, che ci permette di scegliere se mostrare il contenuto della MEM o del Cronometro.

È stato aggiunto quindi un bottone di Save, il cui funzionamento dipende dalla modalità: se il contatore sta contando, serve per salvare gli intertempi; se invece stiamo mostrando il contenuto della MEM, serve per andare avanti nelle locazioni di memoria.

Immagine che contiene schermata, diagramma

Descrizione generata automaticamente

La ***MEM*** è una EPROM da 8 locazioni, ognuna da 17 bit(ore, minuti e secondi). Prende in ingresso gli indirizzi dati dal contatore e salva l’indirizzo dell’ultima locazione scritta. Questo serve per far riprendere la scrittura una volta che si è cambiata modalità di visualizzazione.

Il ***Contatore MEM*** è sempre un contatore Behavioural, che prende un segnale di enable dalla Control Unit, e viene resettato quando si entra in modalità di visualizzazione della MEM. Ciò che mostra il display cambia in base ad un multiplexer, pilotato dalla CU.

Per le implementazioni abbiamo seguito un approccio:

1. Comportamentale per la MEM,
2. [Comportamentale per il Contatore MEM](#_Contatore_comportamentale)

Di seguito, riportiamo l’implementazione della **componente MEM.**

------MEMORIA MEM----------

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

--Memoria del sistema B

entity MEM is

port (

in\_Mem : in STD\_LOGIC\_VECTOR(16 downto 0);

write\_Mem : in STD\_LOGIC;

read\_Mem: in STD\_LOGIC;

adress\_Mem : in STD\_LOGIC\_VECTOR(2 downto 0);

dataread\_out: out STD\_LOGIC\_VECTOR(16 downto 0);

lastWritten: out STD\_LOGIC\_VECTOR(2 downto 0)

);

end entity MEM;

architecture rtl of MEM is

--Definizione della memoria come tipo

type mem\_type is array (0 to 7) of STD\_LOGIC\_VECTOR(16 downto 0);

shared variable mem\_v : mem\_type;

begin

processWrite : process(write\_Mem) is

begin

if (falling\_edge(write\_Mem)) then --scrittura

mem\_v(to\_integer(unsigned(adress\_Mem))) := in\_Mem; --devo scrivere quello che esce dalla somma

lastWritten<=std\_logic\_vector(unsigned(adress\_Mem)+1);

end if;

end process;

processRead: process(read\_Mem)

begin

if(falling\_edge(read\_Mem)) then

dataread\_out<= mem\_v(to\_integer(unsigned(adress\_Mem)));

end if;

end process;

end architecture rtl;

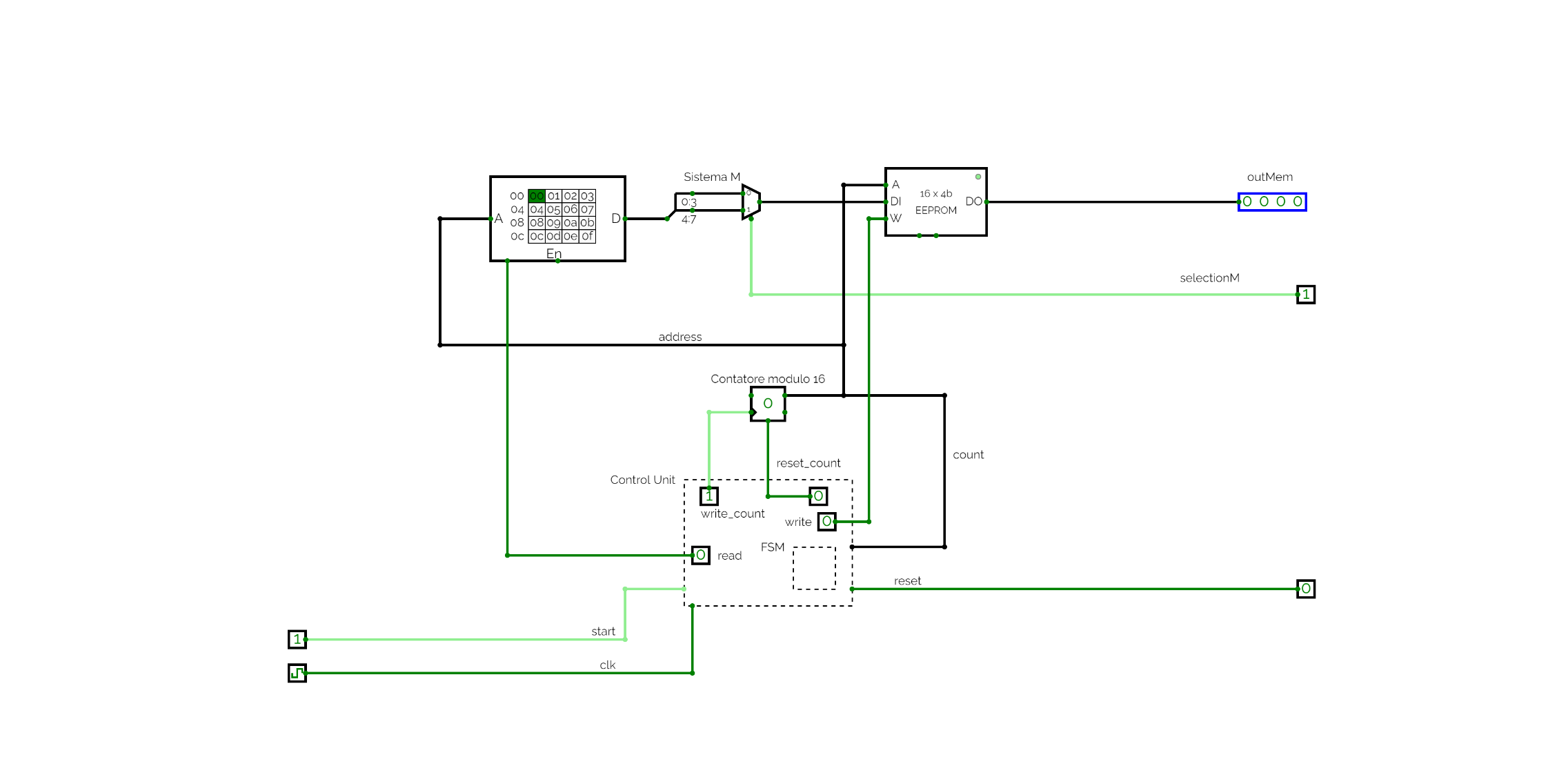
Si notino i due segnali in entrata *write\_mem* e *read\_mem*, i quali attivano i rispettivi process. Il primo permette la memorizzazione del valore salvato e dell’ultimo indirizzo di memoria in cui si è scritto, mentre il secondo permette di leggere il valore salvato dalla MEM.

## Esercizio 6.1: Sistema di lettura-elaborazione-scrittura PO\_PC

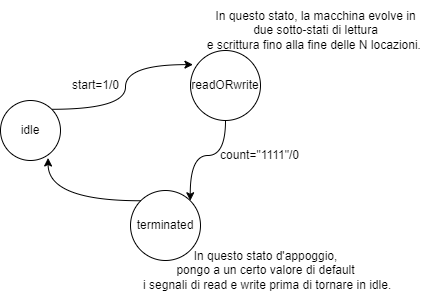
Progettare, implementare in VHDL e verificare mediante simulazione un sistema dotato di una memoria ROM di N locazioni da 8 bit ciascuna, una macchina combinatoria M in grado di trasformare (secondo una funzione a scelta dello studente) la stringa di 8 bit letta dalla ROM in una stringa di 4 bit, e una memoria MEM di N locazioni che memorizza la stringa in output da M.

Il sistema si avvia in corrispondenza di un segnale di START che viene fornito esternamente. Una volta avviato, tramite un’apposita unità di controllo che gestisce la tempificazione del sistema, viene scandita una locazione alla volta della ROM e viene scritta la corrispondente locazione di MEM. Gli indirizzi di memoria sono forniti da un contatore. Le memorie ROM e MEM hanno rispettivamente un read e un write sincrono.

#### Progetto e architettura

Di seguito è riportato uno schema dell’architettura del sistema progettato:

La **Control Unit** implementa una macchina a stati finiti che permette di sincronizzare la lettura del dato dalla **ROM** (soggetto a una elaborazione dal **sistema M**), e alla scrittuta sulla **MEM** (EEPROM in figura). I segnali di **read** e di **write** vengono gestiti dalla *control unit* a seconda dello stato in cui ci troviamo, inoltre, il segnale **write\_count** gestisce il fronte di discesa che scandisce i conteggi del *contatore modulo 16* che specifica l’indirizzo da *leggere/scrivere.* Il segnale *count* è retrazionato alla *control\_unit* per fare in modo di tener traccia del conteggio che una volta arrivato alla fine permette di far transitare la *control\_unit* in uno stato ***idle***.

****Di seguito è riportata il grafo che rappresenta la macchina a stati finiti che gestisce l’evoluzione della *control\_unit:*

Per questo esercizio, sono stati riutilizzate delle componenti già sviluppate nei punti precedenti:

* [ROM da 16 locazioni da 8 bit](#_ROM_da_16_1): la differenza sta nell’avere come input un segnale di *read* che permette di effettuare l’azione di lettura sul fronte di discesa.
* [Sistema M](#_Sistema_M)

Per quanto riguarda il contatore, la struttura è molto simile ai contatori presenti nell’esercizio precedente*,* con l’unica differenza di essere un [contatore modulo 16](#_Contatore_Modulo_16) (con quattro [*flip flop T*](#_Flip_Flop_T)) senza la possibilità di avere una funzione di *set* e l’input da caricare.

Per la **MEM** è stato adottato un approccio simile allo sviluppo della **ROM.**

#### Implementazione

Riportiamo di seguito le componenti non presenti in *Appendice*.

**Control Unit**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5. entity controlUnit is

6. port (

7. clk : in std\_logic;

8. start: in std\_logic; --segnale di start

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

10. count : in std\_logic\_vector (3 downto 0); --conteggio del contatore

11.

12.

13. reset\_count : out std\_logic;

14. read : out std\_logic; --segnale di read

15. write: out std\_logic; --segnale di write

16. write\_count: out std\_logic := '1' --segnale di aumento del conteggio

17. );

18. end entity controlUnit;

19.

20.

21.

22. architecture rtl of controlUnit is

23.

24. TYPE stati IS (idle, readORwrite, terminated);

25.

26.

27. signal stato\_corrente : stati:=idle;

28.

29. shared variable read\_var : std\_logic:='0';

30. shared variable write\_var: std\_logic:='0';

31.

32.

33. begin

34.

35. p: process (clk, reset)

36. begin

37. if reset = '1' then

38. stato\_corrente <= idle;

39. reset\_count <= '1';

40. write<= '0';

41. read<='0';

42. read\_var:='0';

43. write\_var:='0';

44.

45. elsif rising\_edge(clk) then

46.

47. reset\_count <= '0';

48.

49.

50.

51.

52.

53. if stato\_corrente = idle AND start = '1' then

54. read\_var:='1';

55. read <= '1';

56. stato\_corrente<=readORWrite;

57.

58. elsif stato\_corrente=readORwrite AND read\_var='1' AND write\_var ='0' then

59. read\_var:='0';

60. read <= '0';

61. write <= '1';

62. write\_var:='1';

63.

64. write\_count<='1'; --serve per un ipotetico "reset" di write\_count

65.

66. elsif stato\_corrente=readORWrite AND write\_var='1' AND read\_var='0' then

67.

68. write\_var:='0';

69. write <='0';

70. write\_count <='0';

71.

72. read <='1';

73. read\_var:='1';

74.

75. elsif stato\_corrente=terminated then

76.

77. stato\_corrente<= idle;

78. write\_var:='0';

79. write <='0';

80. write\_count <='1';

81.

82. read <='0';

83. read\_var:='0';

84. reset\_count<='1';

85.

86.

87. end if;

88.

89. if stato\_corrente=readORWrite AND count="1111" then

90.

91. stato\_corrente<=terminated;

92.

93. end if;

94.

95.

96.

97.

98.

99. end if;

100. end process;

101.

102.

103.

104. end architecture rtl;

105.

Come illustrato già nella sezione *Progetto e architettura,* la **macchina a stati finiti** che caratterizza la *control\_unit*  ha tre principali stati:

1. ***Idle***, stato in cui il sistema attende il segnale di **start**, per poter entrare in fare di **readORwrite**
2. ***readORwrite,*** stato in cui avviene l’evoluzione della *lettura-elaborazione-scrittura* grazie all’aiuto delle variabili di appoggio *read\_var* e *write\_var,* che grazie al loro valore, permettono di alzare/abbassare i segnali **read** e **write** in maniera opportuna per una corretta fase di lettura, elaborazione e scrittura. Si noti che **abbassando** il segnale di **write,** viene abbassato anche il segnale *write\_count* (posto sempre a 1), che permette di far avanzare il contatore e quindi gli indirizzi di *lettura/scrittura.*
3. **terminated,** stato di appoggio che permette una corretta terminazione delle fasi di *lettura/scrittura* delle ultime locazioni delle rispettive memorie e di passare allo stato di ***idle*** ponendo i segnali **read, write e quelli di appoggio** ai valori iniziali.

**MEM**

1. ------MEMORIA MEM----------

2. library IEEE;

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

4. use IEEE.numeric\_std.all;

5.

6. entity MEM is

7. port (

8. in\_Mem : in STD\_LOGIC\_VECTOR(3 downto 0);

9. write\_Mem : in STD\_LOGIC;

10. adress\_Mem : in STD\_LOGIC\_VECTOR(3 downto 0);

11.

12. outMem : out STD\_LOGIC\_VECTOR(3 downto 0)

13.

14. );

15. end entity MEM;

16.

17.

18. architecture rtl of MEM is

19.

20. --Definizione della memoria come tipo

21. type mem\_type is array (0 to 15) of STD\_LOGIC\_VECTOR(3 downto 0);

22. shared variable mem\_v : mem\_type := (others => x"0");

23.

24.

25. begin

26.

27. processMem : process(write\_Mem) is

28.

29. begin

30. if (falling\_edge(write\_Mem)) then

31. mem\_v(to\_integer(unsigned(adress\_Mem))) := in\_Mem;

32. outMem <= mem\_v(to\_integer(unsigned(adress\_Mem)));

33. end if;

34.

35. end process;

36.

37.

38.

39.

40. end architecture rtl;

41.

Anche in questo caso, la scrittura viene effettuata sul **falling\_edge** del segnale di **write.** Il segnale **outMem**, permette di vedere in uscita *l’ultimo valore* scritto nella memoria da 16 locazioni da 4 bit.

**Top-Module**

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity mPOPC is

8. port (

9. clk: in std\_logic;

10. reset: in std\_logic;

11. start: in std\_logic;

12. selectionM: in std\_logic

13. );

14. end entity mPOPC;

15.

16.

17.

18. architecture rtl of mPOPC is

19.

20.

21. component controlUnit is port(

22. clk : in std\_logic;

23. start: in std\_logic; --segnale di start

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

25. count : in std\_logic\_vector (3 downto 0); --conteggio del contatore

26.

27.

28. reset\_count : out std\_logic;

29. read : out std\_logic; --segnale di read

30. write: out std\_logic; --segnale di write

31. write\_count: out std\_logic := '1' --segnale di aumento del conteggio

32.

33. );

34. end component;

35.

36.

37. component contMod16 is port(

38. clk : in std\_logic;

39. reset : in std\_logic;

40.

41. cont: out std\_logic\_vector(3 downto 0)

42.

43. );

44. end component;

45.

46. component ROM is port(

47. output : out std\_logic\_vector(7 downto 0);

48. address : in std\_logic\_vector(3 downto 0);

49. read : in std\_logic

50.

51. );

52. end component;

53.

54. component MEM is port(

55. in\_Mem : in STD\_LOGIC\_VECTOR(3 downto 0);

56. write\_Mem : in STD\_LOGIC;

57. adress\_Mem : in STD\_LOGIC\_VECTOR(3 downto 0)

58.

59. );

60. end component;

61.

62. component componentM is port(

63.

64. input\_M : in std\_logic\_vector(7 downto 0);

65. output\_M: out std\_logic\_vector(3 downto 0);

66. selection: in std\_logic

67.

68.

69. );

70. end component;

71.

72.

73. signal count\_sig: std\_logic\_vector(3 downto 0);

74. signal reset\_count\_sig: std\_logic;

75. signal read\_sig: std\_logic;

76. signal write\_sig: std\_logic;

77. signal write\_count\_sig: std\_logic; --segnale per far avanzare il contatore

78.

79. signal romtoM : std\_logic\_vector(7 downto 0);

80. signal mtoMem : std\_logic\_vector(3 downto 0);

81.

82.

83.

84. begin

85.

86. controlUnit1: controlUnit port map(

87. clk => clk,

88. start => start,

89. reset => reset,

90. count => count\_sig,

91.

92. reset\_count => reset\_count\_sig,

93. read => read\_sig,

94. write => write\_sig,

95. write\_count => write\_count\_sig

96. );

97.

98. Count: contMod16 port map(

99. clk => write\_count\_sig,

100. reset => reset\_count\_sig,

101.

102. cont=> count\_sig

103. );

104.

105. ROM168: ROM port map(

106. output =>romtoM,

107. address => count\_sig,

108.

109. read=> read\_sig

110.

111. );

112.

113. M: componentM port map(

114. input\_M => romtoM,

115. output\_M => mtoMem,

116.

117. selection => selectionM

118. );

119.

120. MEM164: MEM port map(

121. in\_Mem => mtoMem,

122. write\_Mem => write\_sig,

123. adress\_Mem => count\_sig

124.

125. );

126.

127.

128.

129.

130.

131. end architecture rtl;

132.

Grazie a una descrizione *strutturale,* abbiamo istanziato e legato grazie a vari segnali, tutte le componenti che caratterizzano il **sistema lettura-elaborazione-scrittura.**

#### Simulazione

Di seguito riportiamo il testbench utilizzato per il test del sistema.

1. library IEEE;

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

3.

4. entity tbmPOPC is

5. end entity tbmPOPC;

6.

7. architecture tb\_architecture of tbmPOPC is

8. signal clk\_tb, reset\_tb, start\_tb, selectionM\_tb : std\_logic := '0';

9.

10.

11. component mPOPC

12. port (

13. clk: in std\_logic;

14. reset: in std\_logic;

15. start: in std\_logic;

16. selectionM: in std\_logic

17. );

18. end component;

19.

20. begin

21. DUT : mPOPC

22. port map (

23. clk => clk\_tb,

24. reset => reset\_tb,

25. start => start\_tb,

26. selectionM => selectionM\_tb

27. );

28.

29. clock\_process: process

30. begin

31. while true loop

32. clk\_tb <= '0';

33. wait for 5 ns;

34. clk\_tb <= '1';

35. wait for 5 ns;

36. end loop;

37. wait;

38. end process clock\_process;

39.

40. stimulus\_process: process

41. begin

42.

43. start\_tb <= '1';

44. selectionM\_tb<='0';

45. wait for 10 ns;

46. start\_tb <= '0';

47.

48. wait for 400 ns;

49. start\_tb <= '1';

50. selectionM\_tb<='0';

51. wait for 10 ns;

52. start\_tb <= '0';

53.

54.

55.

56.

57. wait;

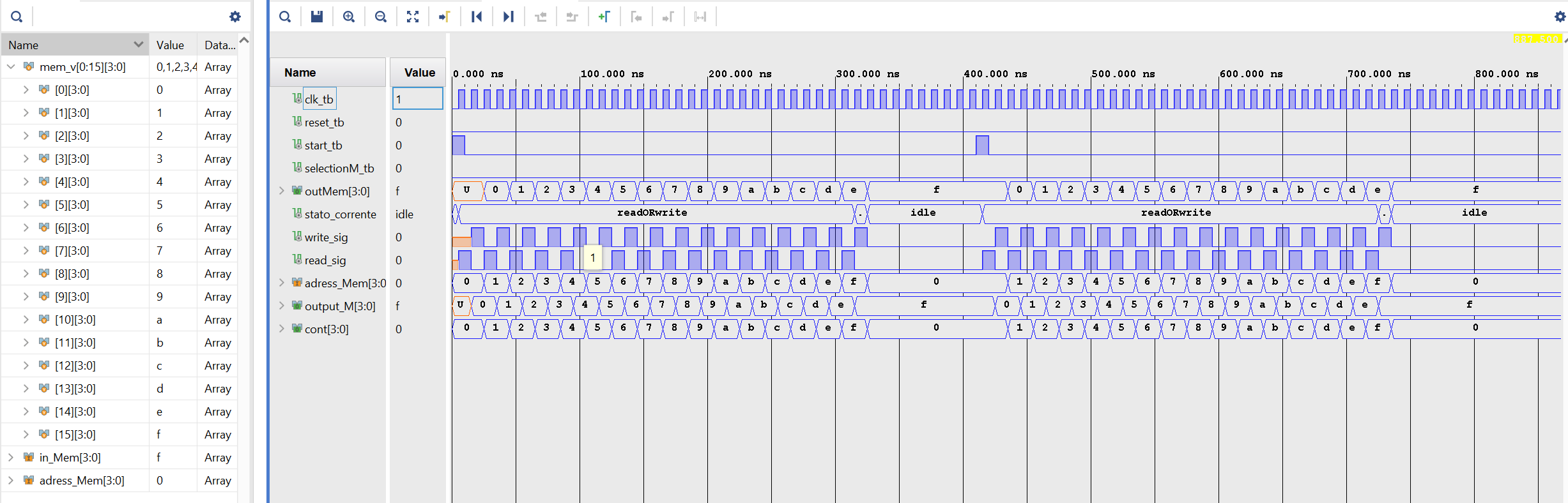
58. end process stimulus\_process;

59.

60. end architecture tb\_architecture;

61.

**Forma d’onda:**

****

## Esercizio 6.2: Implementazione su board dell’esercizio 6.1

Sintetizzare ed implementare su board il componente sviluppato al punto precedente, utilizzando due bottoni per i segnali di read e reset rispettivamente e i led per la visualizzazione delle uscite della macchina istante per istante.

#### Sintesi su board di sviluppo

Per la sinstesi dell’esercizio 6.1 abbiamo apportato alcune modifiche al design. Inanzitutto è stato rimosso il segnale di *start,* che scandiva l’avviamento della macchina permettendo di *leggere-elaborare-scrivere* automaticamente tutte le rispettive posizioni. In questo caso, il suddetto segnale è stato sostituito con un segnale di *read\_ext* ***esterno*** al sistema, legato a un pulsante che permette, quando premuto, di avviare la rispettiva *lettura-elaborazione-scrittura.* Si noti che è stato aggiunto un *Debouncer* per permettere di campionare una singola volta il segnale di *read\_ext* alla singola pressione del pulsante.

Immagine che contiene diagramma, testo, Piano, schematico

Descrizione generata automaticamente

Abbiamo modificato il codice della macchina a stati della **ControlUnit** per sincronizzare la sua evoluzione con la pressione del pulsante di *read\_ext* come segue:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity controlUnit is

port (

clk : in std\_logic;

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

count : in std\_logic\_vector (3 downto 0); --conteggio del contatore

read\_ext: in std\_logic;

reset\_count : out std\_logic;

read : out std\_logic; --segnale di read

write: out std\_logic; --segnale di write

write\_count: out std\_logic := '1' --segnale di aumento del conteggio

);

end entity controlUnit;

architecture rtl of controlUnit is

TYPE stati IS (idle, readORwrite, terminated);

signal stato\_corrente : stati:=idle;

shared variable read\_var : std\_logic:='0';

shared variable write\_var: std\_logic:='0';

begin

p: process (clk, reset)

begin

if reset = '1' then

stato\_corrente <= idle;

reset\_count <= '1';

write<= '0';

read<='0';

read\_var:='0';

write\_var:='0';

elsif rising\_edge(clk) then

reset\_count <= '0';

if stato\_corrente = idle AND read\_ext = '1' then

read\_var:='1';

read <= '1';

stato\_corrente<=readORWrite;

elsif stato\_corrente=readORwrite AND read\_var='1' AND write\_var ='0' then

read\_var:='0';

read <= '0';

write <= '1';

write\_var:='1';

write\_count<='1'; --serve per un ipotetico "reset" di write\_count

elsif stato\_corrente=readORWrite AND write\_var='1' AND read\_var='0' then

write\_var:='0';

write <='0';

write\_count <='0';

stato\_corrente<= idle;

elsif stato\_corrente=terminated then

stato\_corrente<= idle;

write\_var:='0';

write <='0';

write\_count <='1';

read <='0';

read\_var:='0';

reset\_count<='1';

end if;

if stato\_corrente=readORWrite AND count="1111" then

stato\_corrente<=terminated;

end if;

end if;

end process;

end architecture rtl;

Inoltre, il *top-module* prevede l’instanziazione del *debouncer* per il segnale di *read\_ext.*

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity mPOPC is

port (

clk: in std\_logic;

reset: in std\_logic;

selectionM: in std\_logic;

read\_ext: in std\_logic;

outLastWritten: out std\_logic\_vector(3 downto 0)

);

end entity mPOPC;

architecture rtl of mPOPC is

component ButtonDebouncer is port(

RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

BTN : in STD\_LOGIC;

CLEARED\_BTN : out STD\_LOGIC

);

end component;

component controlUnit is port(

clk : in std\_logic;

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

count : in std\_logic\_vector (3 downto 0); --conteggio del contatore

read\_ext: in std\_logic;

reset\_count : out std\_logic;

read : out std\_logic; --segnale di read

write: out std\_logic; --segnale di write

write\_count: out std\_logic := '1' --segnale di aumento del conteggio

);

end component;

component contMod16 is port(

clk : in std\_logic;

reset : in std\_logic;

cont: out std\_logic\_vector(3 downto 0)

);

end component;

component ROM is port(

output : out std\_logic\_vector(7 downto 0);

address : in std\_logic\_vector(3 downto 0);

read : in std\_logic

);

end component;

component MEM is port(

in\_Mem : in STD\_LOGIC\_VECTOR(3 downto 0);

write\_Mem : in STD\_LOGIC;

adress\_Mem : in STD\_LOGIC\_VECTOR(3 downto 0);

outMem : out std\_logic\_vector(3 downto 0)

);

end component;

component componentM is port(

input\_M : in std\_logic\_vector(7 downto 0);

output\_M: out std\_logic\_vector(3 downto 0);

selection: in std\_logic

);

end component;

signal count\_sig: std\_logic\_vector(3 downto 0);

signal reset\_count\_sig: std\_logic;

signal read\_sig: std\_logic;

signal write\_sig: std\_logic;

signal write\_count\_sig: std\_logic; --segnale per far avanzare il contatore

signal romtoM : std\_logic\_vector(7 downto 0);

signal mtoMem : std\_logic\_vector(3 downto 0);

signal cleared\_button\_sig: std\_logic;

begin

buttonDeb: ButtonDebouncer port map (

RST => reset,

CLK=> clk,

BTN=> read\_ext,

CLEARED\_BTN=> cleared\_button\_sig

);

controlUnit1: controlUnit port map(

clk => clk,

reset => reset,

count => count\_sig,

read\_ext=> cleared\_button\_sig,

reset\_count => reset\_count\_sig,

read => read\_sig,

write => write\_sig,

write\_count => write\_count\_sig

);

Count: contMod16 port map(

clk => write\_count\_sig,

reset => reset\_count\_sig,

cont=> count\_sig

);

ROM168: ROM port map(

output =>romtoM,

address => count\_sig,

read=> read\_sig

);

M: componentM port map(

input\_M => romtoM,

output\_M => mtoMem,

selection => selectionM

);

MEM164: MEM port map(

in\_Mem => mtoMem,

write\_Mem => write\_sig,

adress\_Mem => count\_sig,

outMem => outLastWritten

);

end architecture rtl;

Infine, riportiamo di seguito il file dei *constraint:*

## This file is a general .xdc for the Nexys A7-50T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

##Switches

#set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { SW[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

#set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { SW[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

#set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { SW[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

#set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { SW[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

#set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { SW[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

#set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { SW[5] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

#set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { SW[6] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

#set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { SW[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

#set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { SW[8] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

#set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { SW[9] }]; #IO\_25\_34 Sch=sw[9]

#set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { SW[10] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

#set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { SW[11] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

#set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { SW[12] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

#set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { SW[13] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

#set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports {}]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { selectionM }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { outLastWritten[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { outLastWritten[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { outLastWritten[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { outLastWritten[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

#set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { LED[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

#set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { LED[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

#set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { reset }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

#set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { LED[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

#set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { LED[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

#set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { LED[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { LED[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

#set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { LED[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

#set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { LED[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

#set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { LED[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

#set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { LED[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

#set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { led\_uscita }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

## RGB LEDs

#set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_B }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

#set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_G }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

#set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_R }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

#set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_B }]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

#set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_G }]; #IO\_0\_14 Sch=led17\_g

#set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_R }]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##7 segment display

#set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { CA }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

#set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { CB }]; #IO\_25\_14 Sch=cb

#set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { CC }]; #IO\_25\_15 Sch=cc

#set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { CD }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

#set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { CE }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

#set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { CF }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

#set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { CG }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

#set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { DP }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

#set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { AN[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

#set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { AN[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

#set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { AN[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

#set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { AN[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

#set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { AN[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

#set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { AN[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

#set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { AN[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

#set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { AN[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

#set\_property -dict { PACKAGE\_PIN C12 IOSTANDARD LVCMOS33 } [get\_ports { CPU\_RESETN }]; #IO\_L3P\_T0\_DQS\_AD1P\_15 Sch=cpu\_resetn

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { read\_ext }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports { reset }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

#set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

#set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

#set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { BTND }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

…

Il bottone **N17** è stato utilizzato per fornire il segnale di *read\_ext,* invece **M18** per il reset. L’ultimo switch **V10** è stato utilizzato come selezionatore del **Sistema M** e i Led sono stati utilizzati per visualizzare l’uscita che rappresentano l’ultima serie di bit scritta in **MEM**.

# Capitolo 3: Macchine Aritmetiche

### Esercizio 7.1: Moltiplicatore di Booth

Progettare, implementare in VHDL e simulare una macchina moltiplicatore di Booth in grado di effettuare il prodotto di 2 stringhe A e B da 8 bit ciascuna.

#### Progetto e architettura

Per realizzare un *moltiplicatore di Booth,* siamo partiti dalla implementazione fornita del moltiplicatore di **Robertson,** rappresentato dallo schema seguente:

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Descrizione generata automaticamente

Per poter trasformare questo moltiplicatore per poter lavorare con codifica di Booth abbiamo agito su alcune componenti fondamentali dell’architettura:

1. **Shift Register AQ:** formato non più da sedici bit ma da diciassette, per permettere di capire che operazione fare a secoda della coppia Q[1]Q[0].
2. **Latch F:** contiene sempre l’ultimo bit del registro di shift, per permettere l’estensione del segno prevista dall’algoritmo.
3. **Control Unit:** abbiamo cambiato il comportamento della macchina a stati per poter gestire in maniera opportuna l’operazione da svolgere nello stato ***avvia\_somma***.

#### Implementazione

Di seguito riportiamo l’implementazione della **Control Unit.**

**Control Unit**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity unita\_controllo is

port( q0, q1, clock, reset, start: in std\_logic;--clock � il clock della board, clock\_div viene dal divisore di freq

count: in std\_logic\_vector(2 downto 0);

loadM, count\_in, loadAQ, en\_shift: out std\_logic;

selM, selAQ, selF, subtract, stop\_cu: out std\_logic);

end unita\_controllo;

architecture structural of unita\_controllo is

type state is (idle, acquisisci\_op, avvia\_somma, avvia\_shift, incr\_count, avvia\_fshift, fine);

signal current\_state,next\_state: state;

begin

--abbiamo spostato la logica nello stato in sincronia con gli altri segnali.

--selM <= '1'; in ogni istante la selezione del mux � data dal bit meno significativo di A.Q (q0)

reg\_stato: process(clock)

begin

if(clock'event and clock='1') then

if(reset='1') then

current\_state <=idle;

else

current\_state <=next\_state;

end if;

end if;

end process;

comb: process(current\_state, start, count)

begin

-- Attenzione! questo process si attiva ogni volta che è presente una variazione nei segnali della sensitivity list

-- current\_state e count per loro natura variano sempre in corrispodenza del fronte di salita del clock

-- start viene dall'esterno: se non varia (sale e scende) col fronte del clock, si potrebbe avere una situazione

-- in cui il next\_state varia ma non ha modo da stabilizzarsi (perch� current\_state non � ancora variato)

-- quando il moltiplicatore sar� messo su board, START dovr� essere generato come uscita del button debouncer

count\_in <='0';

subtract <='0';

selM<='1';

selAQ <= '0';

selF <= '0';

loadAQ <='0'; --carica nello shift register

loadM <='0'; --carica il moltiplicando nel registro M

stop\_cu <='0';

en\_shift <='0'; --segnale che abilita lo shift durante le prime N-1 iterazioni

CASE current\_state is

WHEN idle =>

if(start='1') then

next\_state <= acquisisci\_op;

loadM <='1'; --abilita il caricamento del moltiplicando nel registro M

loadAQ <='1'; --abilita il caricamento del moltiplicatore e degli 8 zeri in testa

--nello shift register A.Q (perch� selAQ=0)

else

next\_state <= idle;

end if;

--fornisce i segnali di caricamento operandi

WHEN acquisisci\_op =>

next\_state <= avvia\_somma;

--acquisisce gli operandi, su cui il sommatore inizia a lavorare immediatamente

WHEN avvia\_somma =>

selAQ <= '1';

loadAQ <= '1'; --fornisce il segnale di caricamento in A del risultato della somma

if(q1='1' AND q0='0') then

subtract <='1'; --se questa � ultima iterazione avvio una sottrazione e vado in final shift

end if;

if (((q1='1' AND q0='1') OR (q1='0' AND q0='0'))) then

selM<='0'; --come se non facessi ne somma ne sottrazione, solo shift

else

selM<='1'; --somma

end if;

if(count="111") then

next\_state <= avvia\_fshift;

else

next\_state <= avvia\_shift;

end if;

--carica il risultato della somma in A e da fornisce il segnale di shift

WHEN avvia\_shift =>

en\_shift <='1';

next\_state <= incr\_count;

--esegue lo shift, abilita incremento conteggio e predispone per nuova iterazione

WHEN incr\_count =>

count\_in <= '1';

next\_state <= avvia\_somma;

--in correzione fa la sottrazione e predispone per ultimo shift

WHEN avvia\_fshift =>

en\_shift <= '1';

selF <= '1';

next\_state <= fine;

WHEN fine =>

stop\_cu <='1';

next\_state <= idle;

end CASE;

end process;

end structural;

Come si denota facilmente dal codice della macchina a stati, abbiamo modificato lo stato di ***avvia\_somma*** per poter scegliere l’operazione corretta a seconda dei bit q1 e q0 presi dal registro Q.

Se ho la coppia ‘**10’** eseguo la *sottrazione* tra il prodotto parziale e Y, con la coppia ‘**11**’ o ‘**00**’ invece utilizziamo il multiplexer per sommare un vettore di zeri e quindi eseguire uno shift a destra, infine con ’**01**’ viena eseguita la somma e poi shift.

#### Simulazione

Riportiamo di seguito i risultati di alcune delle operazione riportatte nel testbench.

**Operazione 15\*3**

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

Descrizione generata automaticamente**

**Operazione -15\*3**

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

Descrizione generata automaticamente**

**Operazione -128\*-128**

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

Descrizione generata automaticamente**

### Esercizio 7.2: Sintesi del Moltiplicatore di Booth

Sintetizzare il moltiplicatore implementato al punto 7.1 su FPGA e testarlo mediante l’utilizzo dei dispositivi di input/output (switch, bottoni, led, display) presenti sulla board di sviluppo in dotazione. La modalità di utilizzo degli stessi è a completa discrezione degli studenti.

#### Sintesi su board di sviluppo

Per la sintesi del contatore abbiamo legato gli switch come input dei due operandi, e grazie a una pressione del pulsante start, si avvia la moltiplicazione e il risultato è mostrato suoi led. Abbiamo legato anche un tasto di reset per permettere di azzerare il risultato a schermo e caricare altri operandi.

Di seguito riportiamo il file dei constraint utilizzato:

## This file is a general .xdc for the Nexys A7-50T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clock }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clock}];

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { X[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { X[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { X[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { X[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { X[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { X[5] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { X[6] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { X[7] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { Y[0] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { Y[1] }]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { Y[2] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { Y[3] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { Y[4] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { Y[5] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { Y[6] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { Y[7] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { P[0] }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { P[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { P[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { P[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { P[4] }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { P[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { P[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { P[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { P[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { P[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { P[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { P[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { P[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { P[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { P[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { P[15] }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

## RGB LEDs

#set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_B }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

#set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_G }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { stop\_cu }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

#set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_B }]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

#set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_G }]; #IO\_0\_14 Sch=led17\_g

set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { stop\_cu }]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##7 segment display

#set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { CA }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

#set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { CB }]; #IO\_25\_14 Sch=cb

#set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { CC }]; #IO\_25\_15 Sch=cc

#set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { CD }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

#set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { CE }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

#set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { CF }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

#set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { CG }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

#set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { DP }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

#set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { AN[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

#set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { AN[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

#set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { AN[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

#set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { AN[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

#set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { AN[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

#set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { AN[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

#set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { AN[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

#set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { AN[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

#set\_property -dict { PACKAGE\_PIN C12 IOSTANDARD LVCMOS33 } [get\_ports { CPU\_RESETN }]; #IO\_L3P\_T0\_DQS\_AD1P\_15 Sch=cpu\_resetn

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports {start }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

set\_property -dict { PACKAGE\_PIN M18 IOSTANDARD LVCMOS33 } [get\_ports {reset }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

#set\_property -dict { PACKAGE\_PIN P17 IOSTANDARD LVCMOS33 } [get\_ports { bottone3\_sis }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

#set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { bottone\_reset\_debouncer }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

#set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { BTND }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

…

# Capitolo 4: Comunicazione con Handshaking

### Esercizio 8.1:

Progettare, implementare in VHDL e testare mediante simulazione un sistema composto da 2 nodi, A e B, che comunicano mediante un protocollo di handshaking. Il nodo A e il nodo B possiedono entrambi una memoria interna in cui sono memorizzate N stringhe di M bit, denominate X(i) e Y(i) rispettivamente (i=0,..,N-1). Il nodo A trasmette a B ciascuna stringa X(i) utilizzando un protocollo di handshaking; B, ricevuta la stringa X(i), calcola S(i)=X(i)+Y(i) e immagazzina la somma in opportune locazioni della propria memoria interna.

Per il progetto è possibile considerare una implementazione di tipo comportamentale per effettuare la somma, mentre è necessario prevedere esplicitamente un componente contatore sia nel sistema A sia nel sistema B per scandire la trasmissione/ricezione delle stringhe e per terminare la comunicazione.

#### Progetto e architettura

Per implementare l’esercizio richiesto abbiamo scomposto le due entità in varie componenti.

Per il **Sistema A**:

1. Una ***Control Unit,***
2. Un***Contatore* modulo 4,**
3. Una **ROM**

Per il **Sistema B:**

1. Una ***Control Unit,***
2. Un ***Contatore modulo 4,***
3. Una memoria **su cui effettuiamo operazioni di lettura/scrittura,**
4. Un ***sommatore***, che permette di sommare i dati inviati da A e quelli prelevati dalla prima metà della memoria in B rispettivamente.

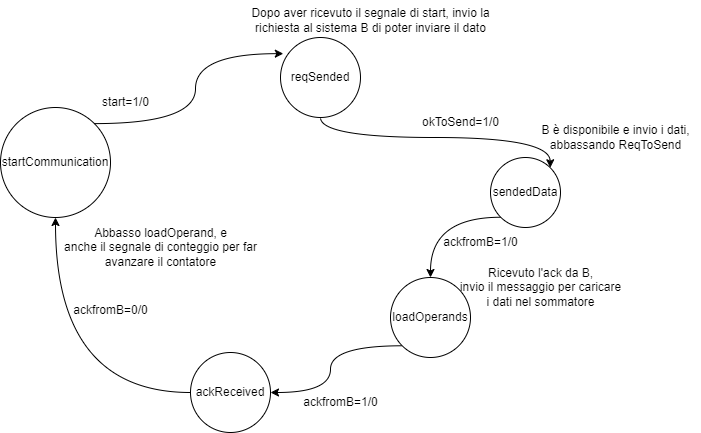
Inoltre, sono utilizzati vari segnali tra i due sistemi per permettere la *comunicazione e lo scambio dei dati tramite handshaking.*

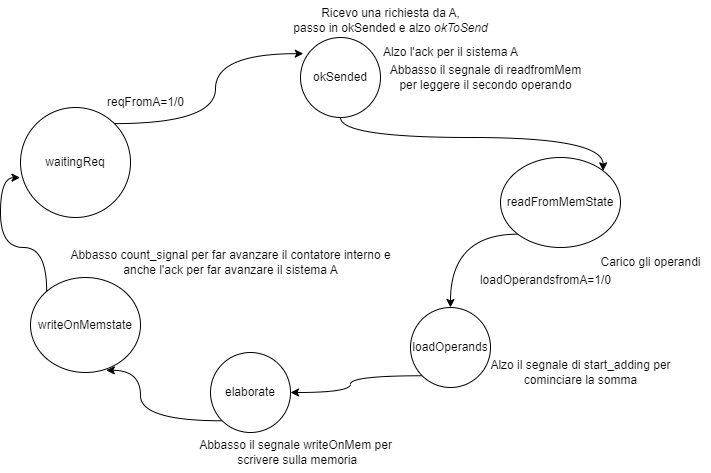
Immagine che contiene schermata, diagramma

Descrizione generata automaticamenteDi seguito riportiamo lo schema generale del sistema implementato.

Il sistema di handshaking utilizzato è *semi-semisincrono* perché ognuno dei sistemi controlla lo stato sul fronte di *salita* del clock, ma utilizziamo segnali di natura asincrona per permettere la stabilizzazione della connessione, l’invio dei dati e la loro elaborazione.

Le ***Control Unit*** riescono a gestire la comunicazione e le operazioni da svolgere tramite due macchine a stati che riportiamo di seguito.

***Control Unit A***

***Control Unit B***

#### Implementazione

Per l’implementazione sono state utilizzate alcune componenti presenti in *Appendice*:

1. [ROM](#_ROM_da_16_1),
2. [Contatore modulo 16 con approccio strutturale](#_Contatore_Modulo_16).

**MEM**

Per quanto riguarda la **MEM,** abbiamo previsto un process per la lettura e uno per la scrittura attivi *rispettivamente* sul segnale di *read* e sul segnale di *write* che riceve dalla ***Control Unit.***

------MEMORIA MEM----------

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

--Memoria del sistema B

entity MEM is

port (

in\_Mem : in STD\_LOGIC\_VECTOR(8 downto 0); --è da 9 per considerare un bit di overflow

write\_Mem : in STD\_LOGIC;

read\_Mem: in STD\_LOGIC;

adress\_Mem : in STD\_LOGIC\_VECTOR(4 downto 0);

dataread\_out: out STD\_LOGIC\_VECTOR(8 downto 0)

);

end entity MEM;

architecture rtl of MEM is

--Definizione della memoria come tipo

type mem\_type is array (0 to 31) of STD\_LOGIC\_VECTOR(8 downto 0);

shared variable mem\_v : mem\_type;

--inizializzo le prime 16 posizioni della memoria per contenere le stringhe Y

constant rilocation: std\_logic\_vector:="10000";

constant zero: std\_logic\_vector:="00000";

constant uno: std\_logic\_vector:="00001";

constant due: std\_logic\_vector:="00010";

constant tre: std\_logic\_vector:="00011";

constant quattro: std\_logic\_vector:="00100";

constant cinque: std\_logic\_vector:="00101";

constant sei: std\_logic\_vector:="00110";

constant sette: std\_logic\_vector:="00111";

constant otto: std\_logic\_vector:="01000";

constant nove: std\_logic\_vector:="01001";

constant dieci: std\_logic\_vector:="01010";

constant undici: std\_logic\_vector:="01011";

constant dodici: std\_logic\_vector:="01100";

constant tredici: std\_logic\_vector:="01101";

constant quattordici: std\_logic\_vector:="01110";

constant quindici: std\_logic\_vector:="01111";

shared variable addressRelocated: std\_logic\_vector(4 downto 0):="00000";

begin

processWrite : process(write\_Mem) is

begin

mem\_v(to\_integer(unsigned(zero))):="0"& x"10";

mem\_v(to\_integer(unsigned(uno))):="0" & x"11"; --concateno il bit in più

mem\_v(to\_integer(unsigned(due))):="0" & x"13";

mem\_v(to\_integer(unsigned(tre))):="0" & x"a0";

mem\_v(to\_integer(unsigned(quattro))):="0" & x"14";

mem\_v(to\_integer(unsigned(cinque))):="0" & x"1b";

mem\_v(to\_integer(unsigned(sei))):="0" & x"1c";

mem\_v(to\_integer(unsigned(sette))):="0" & x"10";

mem\_v(to\_integer(unsigned(otto))):="0" & x"1d";

mem\_v(to\_integer(unsigned(nove))):="0" & x"1f";

mem\_v(to\_integer(unsigned(dieci))):="0" & x"11";

mem\_v(to\_integer(unsigned(undici))):="0" & x"89";

mem\_v(to\_integer(unsigned(dodici))):="0" & x"43";

mem\_v(to\_integer(unsigned(tredici))):="0" & x"21";

mem\_v(to\_integer(unsigned(quattordici))):="0" & x"16";

mem\_v(to\_integer(unsigned(quindici))):="0" & x"f1"; --devo sommare 16 come spiazzamento di base

addressRelocated:=std\_logic\_vector(unsigned(adress\_Mem)+unsigned(rilocation));

if (falling\_edge(write\_Mem)) then --scrittura

mem\_v(to\_integer(unsigned(addressRelocated))) := in\_Mem; --devo scrivere quello che esce dalla somma

end if;

end process;

processRead: process(read\_Mem)

begin

if(falling\_edge(read\_Mem)) then

dataread\_out<= mem\_v(to\_integer(unsigned(adress\_Mem)));

end if;

end process;

end architecture rtl;

La memoria è composta da 32 posizioni da 9 bit (uno in più per prevedere un bit di overflow dato dalla somma), di cui le prime 16 posizioni sono inizializzate a certi valori che rappresentano le stringhe Y da sommare con quelle ricevute ogni volta dal sistema A. Il risultato è scritto dalla sedicesima posizione in poi.

**Adder Strings**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity AdderStrings is

port (

X: in std\_logic\_vector(8 downto 0);

Y: in std\_logic\_vector(8 downto 0);

start\_operation: in std\_logic;

data\_out: out std\_logic\_vector(8 downto 0)

);

end entity AdderStrings;

architecture rtl of AdderStrings is

begin

process(start\_operation)

begin

if(rising\_edge(start\_operation)) then

data\_out<=std\_logic\_vector(unsigned(X)+unsigned(Y));

end if;

end process;

end architecture rtl;

Questo componente ci permette di sommare le due stringhe di bit e porre il risultato in uscita, che verrà poi caricato opportunamente nella **MEM.**

**Control Unit A**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity CUA is

port (

clk: in std\_logic;

start: in std\_logic;

okToSend: in std\_logic;

ackfromB: in std\_logic;

data\_in: in std\_logic\_vector(8 downto 0);

reset: in std\_logic;

reqToSend: out std\_logic;

data\_out: out std\_logic\_vector(8 downto 0):="UUUUUUUUU";

load\_OperandtoB: out std\_logic;

--conteggio e lettura

readfromCutoRom: out std\_logic;

countA: out std\_logic --segnale di conteggio

);

end entity CUA;

architecture rtl of CUA is

TYPE stati is (startCommunication, reqSended, okReceived, sendedData, loadOperands, ackReceived, elaborated);

signal stato\_corrente: stati:=startCommunication;

begin

process (clk)

begin

if(reset='1') then

stato\_corrente<=startCommunication;

reqToSend<='0';

readfromCutoRom<='1';

countA<='1';

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

end if;

countA<='1';

readfromCutoRom<='1';

if(rising\_edge(clk)) then

if(stato\_corrente=startCommunication AND start='1') then

stato\_corrente<=reqSended;

reqToSend<='1';

readfromCutoRom<='0'; --leggo dalla ROM, quindi i data in sono pronti

elsif(stato\_corrente=reqSended AND okToSend='1') then

reqToSend<='0';

stato\_corrente<=sendedData;

data\_out<=data\_in;

elsif (stato\_corrente=sendedData AND ackfromB='1') then --aspetto che elaborazione termini

load\_OperandtoB<='1';

stato\_corrente<=loadOperands;

elsif(stato\_corrente= loadOperands AND ackfromB='1') then

stato\_corrente <= ackReceived;

elsif (stato\_corrente=ackReceived AND ackfromB='0') then

load\_OperandtoB<='0';

countA<='0'; --il conteggio funziona sul falling edge

stato\_corrente<=startCommunication;

end if;

end if;

end process;

end architecture rtl;

Grazie ai segnali ***reqToSend, okToSend, Data, LoadOperands e Ack, possiamo far evolvere la macchina a stati correttamente per ultimare l’invio e la successiva elaborazione dei dati.***

N.B. Il segnale di start viene fornito da testbench per avanzare dal primo stato in poi.

**Control Unit B**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity CUB is

port (

clk: in std\_logic;

reset: in std\_logic;

reqfromA: in std\_logic;

data\_in\_fromA: in std\_logic\_vector(8 downto 0);

data\_fromMem: in std\_logic\_vector(8 downto 0);

loadOperandfromA: in std\_logic;

--comunicazione

okToSend: out std\_logic;

acktoA: out std\_logic; --lo uso anche come segnale per avanzare il conteggio del contatore interno a B, per le posizioni omologhe

--sincronizzazione e conteggio

writeOnMem: out std\_logic;

readfromMem: out std\_logic;

start\_adding: out std\_logic;

count\_signal: out std\_logic;

X: out std\_logic\_vector(8 downto 0);

Y: out std\_logic\_vector(8 downto 0) --operandi

);

end entity CUB;

architecture rtl of CUB is

TYPE stati is (waitingReq, okSended, readFromMemState, loadOperands, elaborate, writeOnMemState);

signal stato\_corrente : stati:= waitingReq;

begin

process (clk)

begin

if(reset='1') then

stato\_corrente<=waitingReq;

writeOnMem<='1';

readfromMem<='1';

start\_adding<='0';

okToSend<='0';

acktoA<='0';

end if;

writeOnMem<='1';

readfromMem<='1';

start\_adding<='0';

count\_signal<='1';

if(rising\_edge(clk)) then

if(stato\_corrente= waitingReq AND reqfromA='1') then

stato\_corrente<= okSended;

okToSend<='1';

elsif(stato\_corrente=okSended) then

okToSend<='0';

readfromMem<='0';

acktoA<='1';

stato\_corrente<=readFromMemState;

elsif(stato\_corrente=readFromMemState AND loadOperandfromA='1') then

X<= data\_in\_fromA;

Y<= data\_fromMem;

stato\_corrente<= loadOperands;

elsif(stato\_corrente=loadOperands) then

start\_adding<='1'; --svolgo operazione

stato\_corrente<=elaborate;

elsif (stato\_corrente=elaborate) then

stato\_corrente<=writeOnMemState;

writeOnMem<='0';

elsif (stato\_corrente= writeOnMemState) then

count\_signal<='0';

stato\_corrente<=waitingReq;

acktoA<='0';

end if;

end if;

end process;

end architecture rtl;

Una possible miglioria nell’approccio utilizzato sta nell’andare a eliminare il sgenale *okToSend* e inviare solo il dato e la richiesta, inviando poi uno *strobe* da A a B per caricare gli operandi. L’entità B infine un ack ad A per fargli capire di aver cominciato l’elaborazione e al termine della suddetta, viene abbassato.

***Top-Module Sistema***

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity Sistema is

port (

clkA: in std\_logic;

clkB: in std\_logic;

start: in std\_logic;

reset: in std\_logic

);

end entity Sistema;

architecture rtl of Sistema is

component SistemaA port(

clk\_sis: in std\_logic;

start\_sis: in std\_logic;

okToSend\_sis: in std\_logic;

ackfromB\_sis: in std\_logic;

reset: in std\_logic;

reqToSend\_sis: out std\_logic;

data\_out\_sis: out std\_logic\_vector(8 downto 0);

load\_Operand\_sis: out std\_logic

);

end component;

component SistemaB port(

clk: in std\_logic;

reqfromA\_sis: in std\_logic;

data\_in\_fromA\_sis: in std\_logic\_vector(8 downto 0);

loadOperand\_fromA\_sis: in std\_logic;

reset: in std\_logic;

okToSend\_sis: out std\_logic;

acktoA\_sis: out std\_logic

);

end component;

signal reqToSend: std\_logic;

signal okToSend: std\_logic;

signal ack: std\_logic;

signal load: std\_logic;

signal data: std\_logic\_vector(8 downto 0);

begin

sisA: SistemaA port map(

clk\_sis=> clkA,

start\_sis=> start,

okToSend\_sis=> okToSend,

ackfromB\_sis=> ack,

reset=> reset,

reqToSend\_sis=> reqToSend,

data\_out\_sis=> data,

load\_Operand\_sis=> load

);

sisB: SistemaB port map(

clk=> clkB,

reqfromA\_sis=> reqToSend,

data\_in\_fromA\_sis=> data,

loadOperand\_fromA\_sis=> load,

reset=> reset,

okToSend\_sis=> okToSend,

acktoA\_sis=> ack

);

end architecture rtl;

Il *top-module* contiene l’istanziazione dei due *Sistemi* che sono composti dagli elementi descritti precedentemente.

#### Simulazione

Per la simulazione abbiamo utilizzato il seguente *test-bench.*

library IEEE;

use IEEE.std\_logic\_1164.all;

entity tbSistema is

end entity tbSistema;

architecture tb\_arch of tbSistema is

-- Constants for clock period and simulation duration

constant CLOCK\_PERIODA : time := 5 ns;

constant CLOCK\_PERIODB : time := 5 ns;

-- Signals for testbench

signal clkA\_tb : std\_logic := '0';

signal clkB\_tb: std\_logic :='1';

signal start\_tb : std\_logic := '0';

signal reset\_tb : std\_logic := '0';

begin

-- Instantiate the DUT (Design Under Test)

dut : entity work.Sistema

port map (

clkA => clkA\_tb,

clkB => clkB\_tb,

start => start\_tb,

reset => reset\_tb

);

-- Clock process

process\_a: process

begin

while true loop

clkA\_tb <= '0';

wait for CLOCK\_PERIODA / 2;

clkA\_tb <= '1';

wait for CLOCK\_PERIODA / 2;

end loop;

wait;

end process process\_a;

process\_b: process

begin

while true loop

clkB\_tb <= '0';

wait for CLOCK\_PERIODB / 2;

clkB\_tb <= '1';

wait for CLOCK\_PERIODB / 2;

end loop;

wait;

end process process\_b;

-- Stimulus process

stimulus\_process: process

begin

-- Initialize inputs

start\_tb <= '0';

reset\_tb <= '0';

-- Start the system

start\_tb <= '1';

-- End simulation

wait;

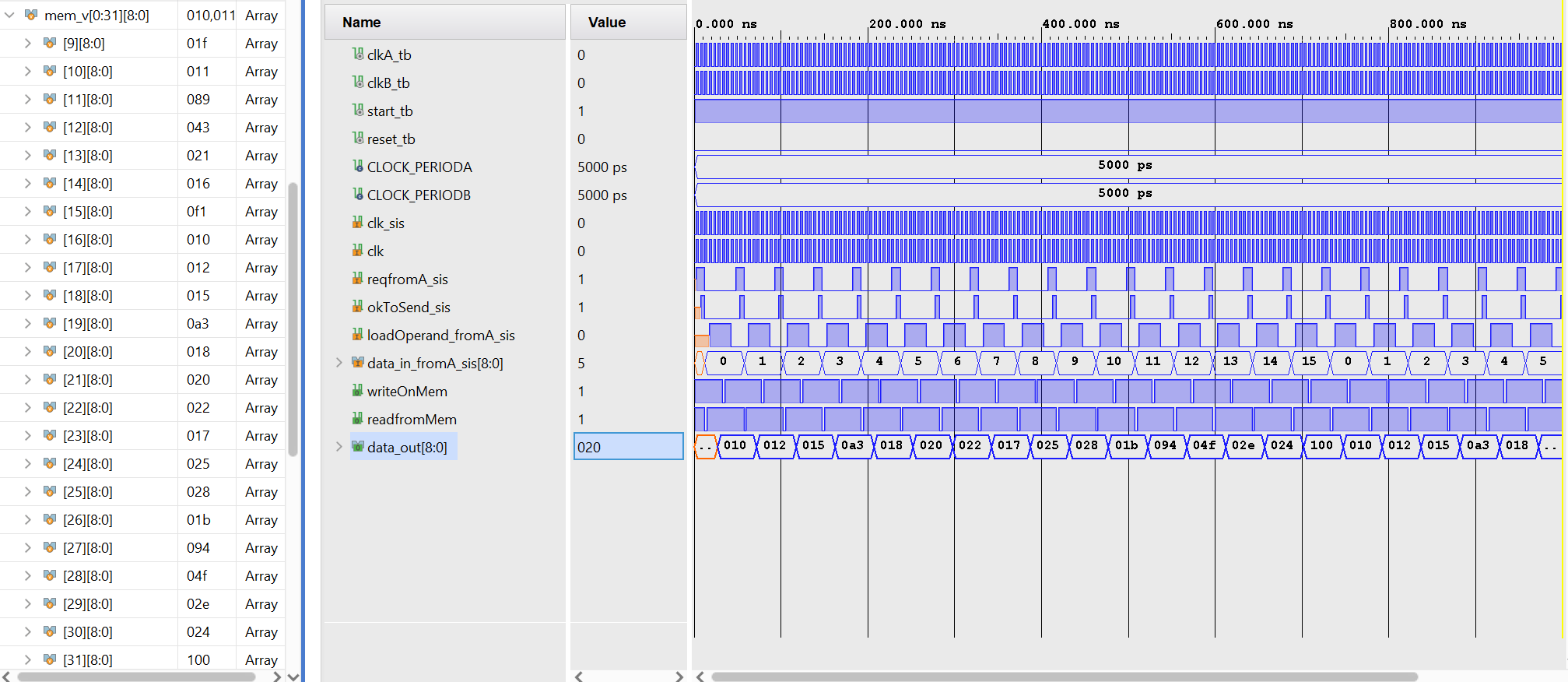
end process stimulus\_process;

end architecture tb\_arch;

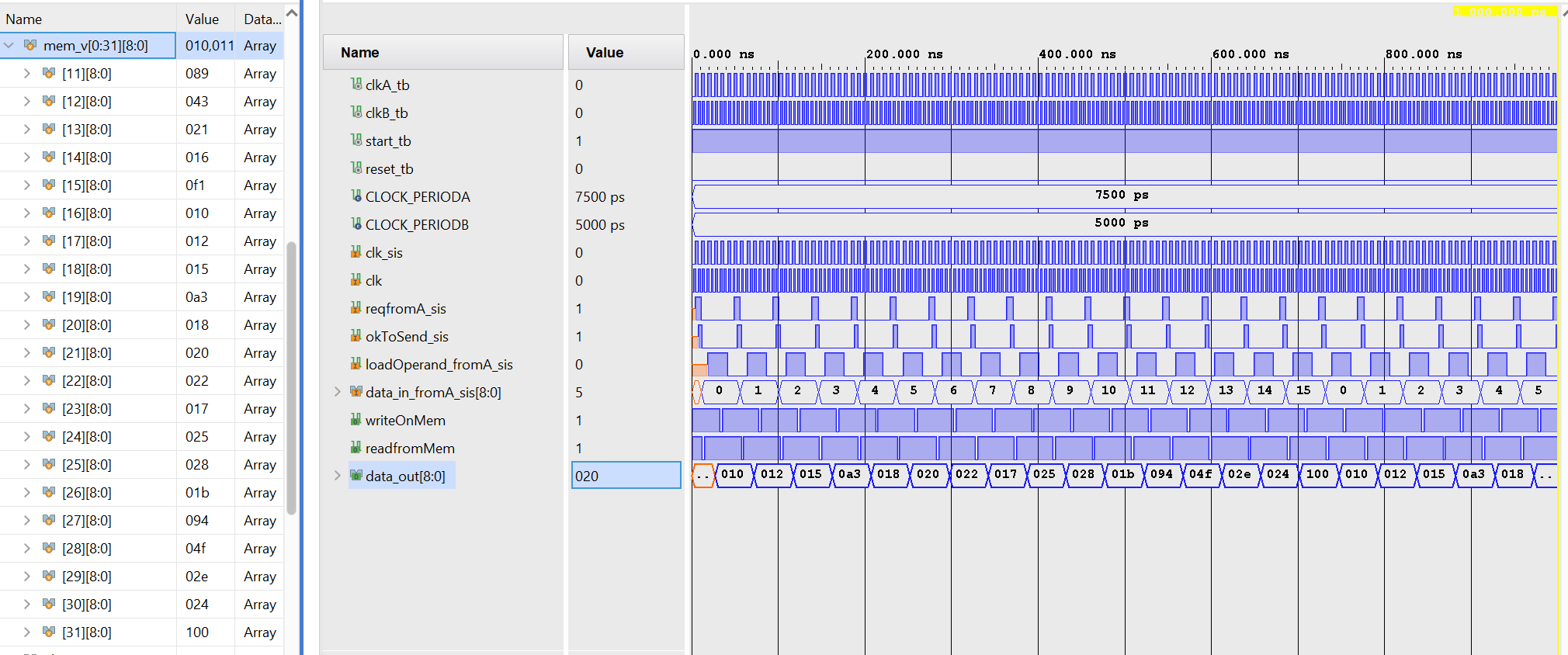
Di seguito sono riportate tre forme d’onda che rispecchiano tre casi differenti:

1. I due sistemi possiedono lo stesso clock
2. I due sistemi possiedono un clock differente (A con un perido di clock maggiore di quello di B e vicevera).

**I sistemi possiedono un clock con lo stesso periodo.**

****

**Clock del sistema A con periodo maggiore del clock del sistema B.**

****

**Clock del sistema B con periodo maggiore del clock del sistema A.**

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

Descrizione generata automaticamente**

# Capitolo 5: Processore

### Esercizio 9

A partire dall’implementazione fornita del processore operante secondo il modello IJVM,

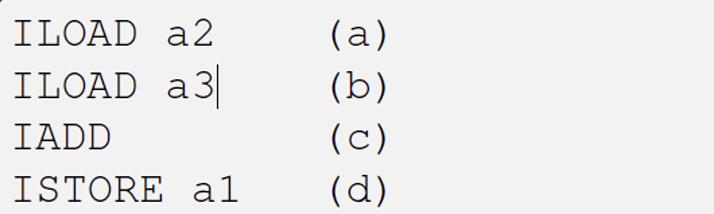
a) si proceda all’analisi dell’architettura mediante simulazione e si approfondisca lo studio del suo funzionamento per due istruzioni a scelta,

b) si modifichi un codice operativo a scelta, documentando tutte le modifiche effettuate

#### Punto A: Analisi dell’architettura, di due istruzioni a scelta e simulazione.

**Analisi dell’architettura.**

Il MIC-1 è un'architettura di processore che consiste in una unità di controllo molto semplice che esegue microcodice da una memoria di 512 parole.

****Il linguaggio microcodice MAL (Micro-Assembly Language) è realizzato per consentire la scrittura di un interprete IJVM (Integer Java Virtual Machine). L’architettura del processore è a stack, ciò significa che per fare le operazioni necessita di avere gli operandi come primo e secondo elemento dello stack. Questo tipo di architettura è molto utile per fare calcoli ripetuti.

Nella figura è possibile vedere la traduzione in Assembly IJVM di una semplice istruzione a1 = a2 + a3. È molto utile osservarla in quanto si percepisce chiaramente la differenza sostanziale tra una macchina a stack ed una a registri generali (infatti, una istruzione di ADD per un’architettura a registri generali, sarebbe **ADD D0, D1**). Gli operandi sono impliciti, essi vengono caricati precedentemente all’istruzione IADD nello stack attraverso l’istruzione ILOAD. Questo porta un vantaggio nella complessità dell’operazione IADD, la quale risulta più semplice.

Immagine che contiene testo, diagramma, Parallelo, Disegno tecnico

Descrizione generata automaticamenteDi seguto troviamo l’immagine del *datapath* del MIC-1:

La parte operativa è formata da:

* Alu,
* Shifter,
* Bus,
* Registri

**BUS**

I BUS permettono la comunicazione tra i registri e in questa architettura ne distinguiamo 2, ovvero il BUS B ed il BUS C.

Il *BUS B* permette di caricare su di esso il valore di un registro, può essere abilitato da un solo registro per volta poiché il trasferimento di dati da parte di due registri contemporaneamente implicherebbe l’inconsistenza dei dati. A valle di ciò, possiamo pensare al BUS B come un MULTIPLEXER.

Il *BUS C* permette di scrivere su un registro il contenuto presente su esso, può essere abilitato da più registri contemporaneamente, in quanto la scrittura dello stesso valore su più registri non porta a nessun tipo di problema. A valle di ciò, possiamo pensare al BUS C come un DEMULTIPLEXER.

**Tempificazione**

È di fondamentale importanza pensare bene alla tempificazione, infatti dobbiamo considerare tempi di propagazione e tempi combinatori. In particolare, è bene tenere in conto i 4 tempi che troviamo nel caso del peggior percorso, ovvero il passaggio di un dato dal BUS B al BUS C.

I 4 tempi sono: il tempo di prelievo dell’istruzione dalla **memoria di controllo**, il *tempo di selezione del registro e trasporto del contenuto sul bus B*, *il tempo di lavoro di ALU e shift register* (se necessario) ed infine il *tempo affinché i valori diventino stabili e prelevabili dal bus C*. Il clock è asimmetrico, ad ogni ciclo viene generato un breve impulso dove il fronte di discesa abilita a scrivere su B mentre il fronte di salita abilita a scrivere su un registro.

**Registri**

A differenza di una macchina a registri generali, ogni singolo registro ha uno specifico scopo e non è utilizzato in modo arbitrario. Anche registri che sono tecnicamente uguali sono considerati diversi, poiché il loro utilizzo ha finalità differenti.

Possiamo distinguere fondamentalmente 3 tipi di registri:

• Interfaccia con la memoria

o **MDR**: Memory Data Register, 32 bit, contiene i dati,

o **MAR**: Memory Address Register, 32 bit, contiene gli indirizzi ed è collegato al bus indirizzi della RAM shiftato di 2 linee in modo da moltiplicare il suo valore per 4. Questa moltiplicazione è necessaria in quanto la RAM è una memoria orientata al byte e quindi l'indirizzo di ogni parola è specificato dall'indirizzo del suo primo byte,

o **PC**: Program Counter, 32 bit,

o **MBR**: Memory Byte Register, 8 bit, possiamo considerarlo simile all’IR, contiene il risultato del PC; (MBRU: MBR ma senza complemento)

• **H**: Holding, ospita il primo operando dell’ALU

• Registri puntatore

o **LV**: Local Variable, punta alla base dell’area di memoria delle variabili locali. In realtà, formalmente punta alle variabili locali del metodo in esecuzione, e c’è sempre un metodo in esecuzione. Se cambia il metodo allora si istanzia un nuovo LV. LV è molto importante anche per consentirci di lavorare non con indirizzi interi ma in termini di offset/spiazzamento, infatti a valle di un’istruzione Assembly, come ILOAD a1, la variabile viene tradotta in termini di spiazzamento rispetto a LV, così da ottenere l’indirizzo di a1 sommando il contenuto di MBR (traduzione della variabile) e il contenuto di LV.

o **SP**: Stack Pointer, punta al valore in testa allo stack,

o **CPP**: Costant Pool Pointer, puntatore a valori costanti,

o **TOS**: Top Of Stack, punta all’indirizzo della testa dello stack,

o **OPC**: Scratch Register, permette di ospitare qualcosa di utile nella micro-istruzione.

Immagine che contiene diagramma, testo, Piano, Disegno tecnico

Descrizione generata automaticamente***Unità di Controllo***

Come già accennato, l’architettura utilizza un’unità di controllo realizzata con logica microprogrammata, infatti, tale unità di controllo è realizzata con i seguenti componenti:

• ***Generatore degli indirizzi di partenza***: l’indirizzo della micro-ROM a cui si accede per eseguire una data operazione,

• ***Control Store***: la micro-ROM stessa,

• ***MPC***: Program Counter della Micro-ROM, indica ciclo per ciclo qual è la prossima microistruzione da prelevare. Spesso non fa altro che ospitare il campo ADDR della microistruzione ma talvolta è necessario modificarlo (es. i salti).

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Nella figura vediamo il formato completo di una microistruzione, la quale ha una lunghezza totale di 36 bit (da notare la codifica dei bit di controllo del BUS B).

I primi 9 bit sono dedicati *all’indirizzo della prossima istruzione da eseguire*. La sua presenza è dovuta alla possibilità che la prossima istruzione non sia necessariamente quella immediatamente successiva a quella in esecuzione.

I successivi 3 bit sono dedicati *alla gestione di un eventuale salto*. In particolare, i bit *JMPZ e JMPN* riflettono rispettivamente l’uscita Z e l’uscita N dell’ALU. Quando uno di questi due bit è alto, viene alzato il bit più significativo dell’MPC, effettuando così un **salto di 256 nella Control Store.**

Difatti, se andassimo a controllare le istruzioni all’interno della micro-ROM noteremmo che ci sono tante istruzioni col bit più significativo basso che rappresentano i “percorsi standard”, mentre poche istruzioni con il bit più significativo alto in quanto rappresentano le sequenze alternative. JMPC invece serve a fare l'equivalente di un salto in cui andiamo a prelevare un offset dalla memoria per capire quanto ci dobbiamo spostare nella control store. Semplicemente se è alto si sostituiscono gli 8 bit meno significativi (quindi quello che sarebbe stato il campo ADDR) con quelli prelevati dalla memoria.

I successivi 8 bit sono dedicati al controllo dell’ALU e dello Shifter. In particolare, i primi due sono per lo Shifter e i successivi per l’ALU. In seguito, sono riportati esempi di controllo dell’ALU. Ne consegue che una eventuale modifica alle istruzioni può essere implementata semplicemente modificando opportunamente i bit della *microistruzione.*

Immagine che contiene testo, numero, parole crociate, Carattere

Descrizione generata automaticamente

Abbiamo visto, quindi, come viene gestita l’esecuzione delle sequenze legate ad una microistruzione, ma come faccio a sapere qual è l’indirizzo della prima microistruzione da prelevare?

È di fondamentale importanza in questa architettura che la codifica di un’istruzione a livello Assembly sia tassativamente uguale all’indirizzo della micro-ROM in cui è memorizzata la prima microistruzione che implementa l’intera sequenza (questa viene prelevata tramite fetch). A questo punto il campo ADDR ed eventuali JMP gestiranno le successive microistruzioni.

**Analisi delle istruzioni: BIPUSH e IADD.**

Le istruzioni che analizzeremo sono tra le due più utilizzate nell’architettura: **IADD** e **BIPUSH.**

Un generico programma per MIC-1:

.main

.var

a

.endvar

BIPUSH 0xB

BIPUSH 0x4

IADD

ISTORE a

HALT

.endmethod

Ognuna di queste istruzioni corrisponde ad una locazione di memoria in cui è possibile prelevare la prima microistruzione da eseguire, il cui formato è stato già approfondito.

Le due procedure si presentano in questo modo:

iadd = 0x65:

MAR = SP = SP - 1; rd

H = TOS

MDR = TOS = MDR + H; wr; goto main

bipush = 0x10:

SP = MAR = SP + 1

PC = PC + 1; fetch

MDR = TOS = MBR; wr; goto main

Un’ulteriore istruzione utilizzata è ISTORE, collocata nella locazione di memoria 0x36.

**Forma d’onda:**

Immagine che contiene linea, software, elettronica, Software multimediale

Descrizione generata automaticamente

In particolare, dall’alto verso il basso:

• Segnali di reset e clock,

• Contenuto del registro SP,

• Giallo: contenuto del registro TOS, ovvero il valore contenuto in testa allo stack; si distinguono gli operandi ed il risultato,

• Contenuto del registro MAR,

• Viola: contenuto del registro MBR, da notare il sincronismo con il blu,

• I bus B e C,

• Arancione: contenuto del registro MDR, si distinguono gli operandi e il risultato,

• Contenuto del registro PC (costantemente incrementato),

• Segnali di fetch,

• Blu: dato proveniente dalla memoria (indirizzo della prossima istruzione ed operandi),

• Rosso: gli indirizzi delle microistruzioni (da notare 0x10 per i BIPUSH, 0x65 per IADD e 0x36

per ISTORE).

**BIPUSH**

Il BIPUSH corrisponde alla locazione di memoria della control store 0x10, ovvero la sedicesima parola. Abbiamo già visto la microprocedura, che ripeteremo per facilitare la lettura.

bipush = 0x10:

SP = MAR = SP + 1

PC = PC + 1; fetch

MDR = TOS = MBR; wr; goto main

Ricercando nella control store, le 3 microistruzioni da eseguire sono:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Next\_addr** | **jam** | **ALU** | **C** | **Mem** | **B** |
| 16 => "000010001 | 000 | 00110101 | 000001001 | 000 | 0100", |
| 17 => "000010010 | 000 | 00110101 | 000000100 | 001 | 0001", |
| 18 => "000000110 | 000 | 00010100 | 001000010 | 100 | 0010" |

La prima cosa da fare è incrementare lo stack-pointer, predisponendolo a spingere l’elemento in testa allo stack. Chiaramente, per far sì che possa avvenire il push, bisogna scrivere l’indirizzo incrementato nel MAR, in modo da memorizzare il dato fisicamente in un’area di memoria stabile.

**SP = MAR = SP + 1**

Questo primo passo si effettua attraverso i campi della microistruzione pari a:

* ALU: 110101, che equivale all’incremento dell’operando B,
* C: 000001001, indicando che il bus C scrive su SP e MAR,
* MEM: 000, nessuna operazione in memoria,
* B: 0100, SP controlla il bus B,
* Next\_addr: 000010001, che equivale a 17, la locazione di memoria successiva,
* JAM: 000, nessun salto necessario.

Il secondo passo prevede l’incremento del PC così da poter leggere l’operando contenuto nell’istruzione. Tale operando viene memorizzato nell’MBR.

**PC = PC + 1; fetch**

Questo secondo passo si effettua attraverso i campi della microistruzione pari a:

* ALU: 110101, che equivale all’incremento dell’operando B,
* C: 000000100, indicando che il bus C scrive su PC,
* Mem: 001, indicando l’operazione di fetch,
* B: 0001, PC controlla il bus B,
* Next\_addr: 000010010, che equivale a 18, la locazione di memoria successiva,
* JAM: 000, nessun salto necessario.

Il terzo ed ultimo passo prevede la copia del contenuto di MBR in TOS ed infine MDR. La parola chiave *wr* specifica la scrittura in memoria del dato. Infine, il *goto main* riporta alla microprocedura *main* dove avviene l’incremento di PC, un’operazione di fetch per prelevare un byte dalla memoria istruzioni ed un *goto*(MBR) per saltare alla microistruzione memorizzata nel buffer.

**MDR = TOS = MBR; wr; goto main**

Questo ultimo passo si effettua attraverso i campi della microistruzione pari a:

* ALU: 010100, che equivale al passaggio dell’operando B invariato,
* C: 001000010, il bus C scrive su MDR e TOS,
* Mem: 100, indicando la scrittura di dati,
* B: 0010, MBR controlla il bus B,
* Next\_addr: 000000110, un indirizzo che porta ad un idle in cui viene incrementato il pc,
* JAM: 000, nessun salto necessario.

Termina così il flusso di esecuzione dell’istruzione BIPUSH.

**IADD**

IADD corrisponde alla locazione di memoria della control store 0x65, ovvero la centouno-esima parola. Abbiamo già visto la microprocedura, che ripeteremo per facilitare la lettura.

iadd = 0x65:

MAR = SP = SP - 1; rd

H = TOS

MDR = TOS = MDR + H; wr; goto main

Ricercando nella control store, le tre microistruzioni da eseguire sono:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Next\_addr** | | | **jam** | **ALU** | | **C** | | | **Mem** | | **B** |
| 101 => "001100110 | | | 000 | 00110110 | | 000001001 | | | 010 | | 0100", |
| 102 => "001100111 | | | 000 | 00010100 | | 100000000 | | | 000 | | 0111", |
| 103 => "000000110 | | | 000 | 00111100 | | 001000010 | | | 100 | | 0000" |
|  | |  |  | | |  | |  |  | |

La prima cosa da fare, a differenza del BIPUSH, è decrementare lo stack pointer, per effettuare una read dalla memoria. Il decremento e il successivo passaggio dell’indirizzo puntato al MAR permette l’acquisizione dell’operando dalla memoria.

NB: il segnale di read *rd* diventa alto solo al fronte di clock successivo, poiché è solo allora che l’indirizzo a cui eseguire la lettura diventa disponibile nel MAR.

**MAR = SP = SP - 1; rd**

Questo primo passo si effettua attraverso i campi della microistruzione pari a:

* ALU: 110110, che equivale al decremento dell’operando B,
* C: 000001001, indicando che il bus C scrive su SP e MAR,
* MEM: 010, lettura dei dati dalla memoria,
* B: 0100, SP controlla il bus B,
* Next\_addr: 001100110, che equivale a 102, la locazione di memoria successiva,
* JAM: 000, nessun salto necessario.

La seconda cosa da fare è far passare l’operando B invariato e scriverlo sul registro H (prendendolo dal registro TOS).

**H = TOS**

Questo secondo passo si effettua attraverso i campi della microistruzione pari a:

* ALU: 010100, che equivale al passaggio dell’operando B invariato;
* C: 100000000, indicando che il bus C scrive su H;
* Mem: 000, nessuna operazione in memoria;
* B: 0111, TOS controlla il bus B;
* Next\_addr: 001100111, che equivale a 103, la locazione di memoria successiva;
* JAM: 000, nessun salto necessario.

La terza ed ultima cosa da fare è la somma vera e propria, ovvero sommare il contenuto di MDR ed H e memorizzarlo in MDR stesso e nel TOS.

**MDR = TOS = MDR + H; wr; goto main**

Questo ultimo passo si effettua attraverso i campi della microistruzione pari a:

* ALU: 111100, che equivale alla somma dei due operandi,
* C: 001000010, il bus C scrive su MDR e TOS,
* Mem: 100, indicando la scrittura di dati,
* B: 0000, MDR controlla il bus B,
* Next\_addr: 000000110, un indirizzo che porta ad un idle in cui viene incrementato il PC,
* JAM: 000, nessun salto necessario.

Termina così il flusso di esecuzione dell’istruzione IADD.

NB: sia nel BIPUSH che nell’IADD, il dato viene memorizzato sia nel TOS che nell’MDR. Difatti, è nostro desiderio memorizzarlo in MDR per poterlo scrivere in memoria ma anche in testa allo stack, per definizione stessa della macchina.

#### Punto B: Modifica di un codice operativo.

L’istruzione che abbiamo deciso di modificare è IAND, trasformandola in una IOR:

iand = 0x7E:

MAR = SP = SP - 1; rd

H = TOS

MDR = TOS = MDR AND H; wr; goto main

Com’è possibile osservare, si trova nella locazione di memoria 0x7E (o 126 in decimale) ed è implementata in modo praticamente identico a IADD, eccezion fatta per l’ultima operazione.

Il programma da eseguire è:

.main

.var a

.endvar

BIPUSH 0xB

BIPUSH 0x5

IAND

ISTORE a

HALT

.endmethod

**Forma d’onda data dall’esecuzione:**

Immagine che contiene software, linea, Software multimediale, Policromia

Descrizione generata automaticamente  
  
La AND logica tra 0xB e 0x5 fornisce risultato **0x1.**

Le control word associate alle tre microistruzioni sono:

126 => "001111111000001101100000010010100100",

127 => "010000000000000101001000000000000111",

128 => "000000110000000011000010000101000000"

Chiaramente, le prime due control word sono uguali a quelle già analizzate in IADD, ovvero decremento di SP e scrittura sul registro H. Ciò che è diverso è la parte dedicata all’ALU:

001100

Per ottenere l’effetto di una IOR dobbiamo modificare proprio questi bit, facendoli diventare

011100

Come visto dalla tabella delle operazioni nel paragrafo dedicato all’*Unità di controllo.*

Una volta apportata questa modifica il risultato è il seguente:

Immagine che contiene schermata, elettronica, software, Software multimediale

Descrizione generata automaticamente

Il risultato ottenuto è una OR bit a bit tra il valore 0xB e 0x5, quindi **0xF**.

# Capitolo 6: Interfaccia seriale

### Esercizio 10

Partendo dall’implementazione fornita dalla Digilent di un dispositivo UART-RS232 (componente RS232RefComp.vhd), progettare, implementare e simulare in VHDL un sistema composto da 2 unità A e B che condividono lo stesso segnale di clock e comunicano tra loro mediante interfaccia seriale. Il sistema A contiene una ROM di 8 locazioni da 1 byte ciascuno, un contatore CONT\_A per scandire le locazioni della ROM e una UART\_A, mentre il sistema B contiene una memoria MEM di 8 locazioni da 1 byte ciascuno, un contatore CONT\_B per scandire le locazioni della MEM e una UART\_B. Quando un segnale WR viene asserito nell’unità A, viene prelevato un byte dalla ROM e inviato all’unità B, che dovrà riceverlo e salvarlo in MEM.

#### Progetto e architettura

Per realizzare questo sistema abbiamo scelto di utilizzare la modalità di trasmissione simplex con protocollo di comunicazione sincrono, la quale prevede un’unica linea monodirezionale in cui soltanto un’entità trasmette e l’altra riceve ed entrambe devono condividere il clock (questo protocollo di comunicazione è consigliabile quando il segnale di clock deve coprire piccole distanze).

Il baud rate utilizzato nel codice è 9600, ovvero vengono trasmessi 9600 bit ogni secondo tra le entità (ovviamente il sender e il receiver devono accordarsi per avere lo stesso baud rate).

Per realizzare i collegamenti tra le entità, abbiamo utilizzato la UART (Universal Asynchronous Receiver Transmitter), ossia un dispositivo hardware che supporta la comunicazione seriale asincrona, ed è stata riadattata affinché le due entità condividessero lo stesso segnale di clock.

L’interfaccia UART consiste di due segnali TX (sul quale viaggiano i dati da trasmettere) e RX (sul quale viaggiano i dati da ricevere) e i dispositivi devono accordarsi sulla struttura del frame e sul baud rate, come detto sopra (in particolare, affinché si possa garantire un corretto funzionamento, i baud rate dei due dispositivi comunicanti non devono differire di più del 10%).

#### Implementazione

Di seguto riportiamo l’implementazione di alcune delle componenti utilizzate.

**CONTROL UNIT – ENTITY “A”**

Riportiamo qui la Control Unit dell’entità A, responsabile dell’invio dei dati verso l’entità B.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity CU\_A is

port (

clk: in std\_logic;

start : in std\_logic;

Write\_UART: out std\_logic;

conteggio\_attuale: in std\_logic\_vector(2 downto 0); --mi permette di controllare il valore cui è arrivato il contatore

conteggio: out std\_logic; --conteggio contatore ROM

TBE\_UART: in std\_logic

);

end entity CU\_A;

architecture rtl of CU\_A is

type stati is (IDLE, INVIA, TRASMETTI, FINE\_TRASMISSIONE);

signal stato\_corrente, stato\_prossimo : stati := IDLE;

begin

CU\_A: process(clk)

begin

if(rising\_edge(clk)) then

stato\_corrente<=stato\_prossimo;

end if;

end process;

CU\_A2: process(start, TBE\_UART, stato\_corrente)

begin

Write\_UART <= '0';

conteggio <= '0';

case stato\_corrente is

when IDLE =>

Write\_UART <= '0';

if(start = '1') then

stato\_prossimo<=INVIA;

else

stato\_prossimo<=IDLE;

end if;

when INVIA =>

Write\_UART<='1';

stato\_prossimo<=TRASMETTI;

when TRASMETTI =>

Write\_UART<='0';

if(TBE\_UART='1') then

stato\_prossimo<=FINE\_TRASMISSIONE;

else

stato\_prossimo<=TRASMETTI;

end if;

when FINE\_TRASMISSIONE =>

if(conteggio\_attuale="111") then

stato\_prossimo<=IDLE;

else

conteggio<='1';

stato\_prossimo<=INVIA;

end if;

end case;

end process;

end architecture rtl;

**CONTROL UNIT – ENTITY “B”**

Riportiamo qui la Control Unit dell’entità B, responsabile dell’acquisizione dei dati in arrivo dall’entità A.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity CU\_B is

port (

clk: in std\_logic;

conteggio\_attuale: in std\_logic\_vector(2 downto 0);

conteggio: out std\_logic;

read\_UART: out std\_logic;

RDA\_UART: in std\_logic;

Write\_on\_Mem: out std\_logic;

reset\_Uart: out std\_logic

);

end entity CU\_B;

architecture rtl of CU\_B is

type stati is (RICEVI, LEGGI, SCRIVI);

signal stato\_corrente, stato\_prossimo: stati := RICEVI;

begin

CU\_B: process(clk)

begin

if(rising\_edge(clk)) then

stato\_corrente<=stato\_prossimo;

end if;

end process;

CU\_B2: process(RDA\_UART, stato\_corrente)

begin

Write\_on\_Mem <='1';

read\_UART <= '0';

conteggio <='0';

reset\_Uart<='0';

case stato\_corrente is

when RICEVI =>

Write\_on\_Mem <='1';

read\_UART <= '0';

conteggio <='0';

reset\_Uart<='0';

if(RDA\_UART = '1') then

stato\_prossimo<=LEGGI;

else

stato\_prossimo <= RICEVI;

end if;

when LEGGI =>

read\_UART<='1';

reset\_Uart<='0';

stato\_prossimo <= SCRIVI;

when SCRIVI =>

read\_UART<='1';

Write\_on\_Mem <= '0';

reset\_Uart<='1';

if(conteggio\_attuale="111") then

stato\_prossimo <= RICEVI;

else

conteggio <= '1';

stato\_prossimo <= RICEVI;

end if;

end case;

end process;

end architecture rtl;

**SISTEMA COMPLESSIVO**

Di seguito abbiamo riportato il sistema generale, esplicitando le due entità che realizzano l’architettura.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity Sistema is

port (

clk: in std\_logic;

start\_sis: in std\_logic;

reset\_sis : in std\_logic;

dato\_uscita: out std\_logic\_vector(7 downto 0)

);

end entity Sistema;

architecture rtl of Sistema is

component EntityA is port(

clk: in std\_logic;

start : in std\_logic;

reset: in std\_logic;

TXD\_UART: out std\_logic

);

end component;

component EntityB is port(

clk: in std\_logic;

reset: in std\_logic;

uscita\_mem: out std\_logic\_vector(7 downto 0);

RXD\_UART\_B: in std\_logic

);

end component;

signal TXDtoRXD : std\_logic;

begin

A: EntityA port map(

clk => clk,

start => start\_sis,

reset => reset\_sis,

TXD\_UART => TXDtoRXD

);

B : EntityB port map(

clk => clk,

reset => reset\_sis,

uscita\_mem => dato\_uscita,

RXD\_UART\_B => TXDtoRXD

);

end architecture rtl;

#### Simulazione

Di seguito il testbench del sistema.

-- Testbench created online at:

-- https://www.doulos.com/knowhow/perl/vhdl-testbench-creation-using-perl/

-- Copyright Doulos Ltd

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

entity Sistema\_tb is

end;

architecture bench of Sistema\_tb is

component Sistema

port (

clk: in std\_logic;

start\_sis: in std\_logic;

reset\_sis : in std\_logic;

dato\_uscita: out std\_logic\_vector(7 downto 0)

);

end component;

signal clk: std\_logic;

signal start: std\_logic;

signal reset: std\_logic;

signal dato\_uscita: std\_logic\_vector(7 downto 0) ;

begin

uut: Sistema port map ( clk => clk,

start\_sis => start,

reset\_sis => reset,

dato\_uscita => dato\_uscita );

stimulus: process

begin

-- Put initialisation code here

start <= '1';

wait for 10 ns;

start<='0';

-- Put test bench stimulus code here

wait;

end process;

clock: process

begin

while true loop

wait for 0.6 ps;

clk <='0';

wait for 0.6 ps;

clk <='1';

end loop;

end process;

end;

**Forma d’onda**

Immagine che contiene testo, linea, Diagramma, diagramma

Descrizione generata automaticamente

# Capitolo 7: Switch multistadio

### Esercizio 11.1

Progettare ed implementare in VHDL uno switch multistadio secondo il modello omega network. Lo switch deve consentire lo scambio di messaggi di 2 bit ciascuno da un nodo sorgente a un nodo destinazione in una rete con 4 nodi, implementando uno schema a priorità fissa fra i nodi (es. nodo 1 più prioritario, con priorità decrescenti fino al nodo 4).

#### Progetto e architettura

Per realizzare uno switch multistadio che permette di trasferire "pacchetti" di due bit abbiamo bisogno di un componente fondamentale. Tale componente è formato da:

1. un Multiplexer con: due ingressi da due bit, una uscita da due bit, un bit di selezione

2. un Demultiplexer con: un ingresso da due bit, due uscite da due bit, un bit di selezione

L’interconnessione di questi componenti forma una parte operativa.

Per implementare uno schema a priorità fissa fra i nodi si è sfruttata l’architettura intrinseca del modello omega network. Lo schema che vogliamo ricreare associa ad ogni nodo una priorità crescente, partendo dal nodo a priorità più bassa l’ordine è il seguente: nodo 0, nodo 1, nodo 2, nodo 3. Si nota che se gli switch, ogni volta che arrivano più richieste contemporanee di invio pacchetto, scelgono sempre una porta “privilegiata” stiamo implicitamente rispettando la gerarchia delle priorità.

Per la parte di controllo abbiamo implementato due componenti:

1. Una Contro Unit: due ingressi di richiesta invio pacchetto, un uscita di selezione

2. Un Adress Selecter: un ingresso di selezione, due ingressi da due bit(sorgente e destinazione per ciascuna porta), due uscite ciascuna di un singolo bit(sorgente e destinazione scelti)

La Control Unit manda un segnale di selezione all’Adress Selecter, il quale in base al valore inviato sceglie di caricare gli indirizzi sorgente e destinazione della porta 1 o della porta 2. Se alla CU arrivano più richieste in contemporanea sceglie indiscriminatamente di far caricare gli indirizzi della porta 1 (i nodi che all’i-esimo livello della rete hanno un bit di indirizzo sorgente pari a 1)

Di seguito si riporta lo schema progettuale dello switch.

Immagine che contiene diagramma, testo, linea, Piano

Descrizione generata automaticamente

#### Implementazione

Di seguito sono riportate le implementazioni dei componenti più significativi.

**Control Unit**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity controlUnit is

port (

clk : in STD\_LOGIC;

--Segnali di prenotazione invio pacchetto da parte delle destinazioni

warning\_1 : in STD\_LOGIC;

warning\_2 : in STD\_LOGIC;

--Comando di uscita della control unit che comandano lo switch

command\_to\_selecter : out STD\_LOGIC

);

end entity controlUnit;

architecture rtl of controlUnit is

begin

processChoice : process (clk) is

begin

if(rising\_edge(clk)) then

--Se arrivano due warning in contemporanea, prediligo la SECONDA porta di ingresso

if(warning\_1='1' and warning\_2 = '1') then

command\_to\_selecter <= '1';

--Altrimenti verifico chi mi ha inviato il pacchetto e prendo le sue selezioni

elsif(warning\_1 = '1'and warning\_2 = '0') then

command\_to\_selecter <= '0';

elsif(warning\_2 = '1'and warning\_1 = '0') then

command\_to\_selecter <= '1';

end if;

end if;

end process;

end architecture rtl;

La Control Unit è stata implementata con un process sensibile al clock di sistema. Sceglie che selezione inviare al Selecter in base alle richieste di invio pacchetto.

**Address Selecter**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity adressSelecter is

port (

s\_selecter\_source\_porta\_1 : in STD\_LOGIC;

s\_selecter\_des\_porta\_1 : in STD\_LOGIC;

s\_selecter\_source\_porta\_2 : in STD\_LOGIC;

s\_selecter\_des\_porta\_2: in STD\_LOGIC;

selection : in STD\_LOGIC;

s\_source\_out : out STD\_LOGIC;

s\_des\_out : out STD\_LOGIC

);

end entity adressSelecter;

architecture rtl of adressSelecter is

begin

processSelecter : process(selection)

begin

if(selection = '0') then

s\_source\_out <= s\_selecter\_source\_porta\_1 ;

s\_des\_out <= s\_selecter\_des\_porta\_1;

elsif(selection = '1') then

s\_source\_out <= s\_selecter\_source\_porta\_2;

s\_des\_out <= s\_selecter\_des\_porta\_2;

end if;

end process;

end architecture rtl;

L’Adress Selecter è un elemento puramente combinatorio che seleziona e manda in uscita i bit di indirizzo sorgente e destinazione di una delle due porte di ingresso.

#### Simulazione

Di seguito riportiamo il testbench utilizzato per il test dello switch multistadio.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity switchMultiStadio\_tb is

end entity switchMultiStadio\_tb;

architecture tb of switchMultiStadio\_tb is

-- Constants

constant CLK\_PERIOD : time := 10 ns; -- Clock period

-- Signals

signal clk\_tb : std\_logic := '0'; -- Testbench clock signal

signal in\_sw\_porta\_1\_tb : std\_logic\_vector(1 downto 0) := "00"; -- Example input for in\_sw\_porta\_1

signal in\_sw\_porta\_2\_tb : std\_logic\_vector(1 downto 0) := "00"; -- Example input for in\_sw\_porta\_2

signal adr\_sw\_source\_porta\_1\_tb : std\_logic := '0'; -- Example input for adr\_sw\_source\_porta\_1

signal adr\_sw\_des\_porta\_1\_tb : std\_logic := '0'; -- Example input for adr\_sw\_des\_porta\_1

signal adr\_sw\_source\_porta\_2\_tb : std\_logic := '0'; -- Example input for adr\_sw\_source\_porta\_2

signal adr\_sw\_des\_porta\_2\_tb : std\_logic := '0'; -- Example input for adr\_sw\_des\_porta\_2

signal warning\_sw\_1\_tb : std\_logic := '0'; -- Example input for warning\_sw\_1

signal warning\_sw\_2\_tb : std\_logic := '0'; -- Example input for warning\_sw\_2

-- Component instantiation

component switchMultiStadio

port (

clk : in std\_logic;

in\_sw\_porta\_1 : in std\_logic\_vector(1 downto 0);

in\_sw\_porta\_2 : in std\_logic\_vector(1 downto 0);

out\_sw\_porta\_1 : out std\_logic\_vector(1 downto 0);

out\_sw\_porta\_2 : out std\_logic\_vector(1 downto 0);

adr\_sw\_source\_porta\_1 : in std\_logic;

adr\_sw\_des\_porta\_1 : in std\_logic;

adr\_sw\_source\_porta\_2 : in std\_logic;

adr\_sw\_des\_porta\_2 : in std\_logic;

warning\_sw\_1 : in std\_logic;

warning\_sw\_2 : in std\_logic

);

end component;

begin

-- DUT instantiation

dut: switchMultiStadio

port map (

clk => clk\_tb,

in\_sw\_porta\_1 => in\_sw\_porta\_1\_tb,

in\_sw\_porta\_2 => in\_sw\_porta\_2\_tb,

out\_sw\_porta\_1 => open, -- Unused output

out\_sw\_porta\_2 => open, -- Unused output

adr\_sw\_source\_porta\_1 => adr\_sw\_source\_porta\_1\_tb,

adr\_sw\_des\_porta\_1 => adr\_sw\_des\_porta\_1\_tb,

adr\_sw\_source\_porta\_2 => adr\_sw\_source\_porta\_2\_tb,

adr\_sw\_des\_porta\_2 => adr\_sw\_des\_porta\_2\_tb,

warning\_sw\_1 => warning\_sw\_1\_tb,

warning\_sw\_2 => warning\_sw\_2\_tb

);

-- Clock process

clk\_process: process

begin

while now < 1000 ns loop -- Simulate for 1000 ns

clk\_tb <= not clk\_tb; -- Toggle the clock

wait for CLK\_PERIOD / 2;

end loop;

wait;

end process clk\_process;

-- Stimulus process

stimulus: process

begin

-- Insert your test stimuli here

-- Example:

wait for 50ns;

in\_sw\_porta\_1\_tb <= "01"; -- Example input for in\_sw\_porta\_1

wait for 20 ns;

in\_sw\_porta\_2\_tb <= "10"; -- Example input for in\_sw\_porta\_2

wait for 20 ns;

adr\_sw\_source\_porta\_1\_tb <= '0'; -- Example input for adr\_sw\_source\_porta\_1

wait for 20 ns;

adr\_sw\_des\_porta\_1\_tb <= '0'; -- Example input for adr\_sw\_des\_porta\_1

wait for 20 ns;

adr\_sw\_source\_porta\_2\_tb <= '1'; -- Example input for adr\_sw\_source\_porta\_2

wait for 20 ns;

adr\_sw\_des\_porta\_2\_tb <= '1'; -- Example input for adr\_sw\_des\_porta\_2

wait for 20 ns;

warning\_sw\_2\_tb <= '1'; -- Example input for warning\_sw\_2, maggiore priorità

warning\_sw\_1\_tb <= '1'; -- Example input for warning\_sw\_1

wait for 100 ns;

warning\_sw\_2\_tb <= '0'; -- Example input for warning\_sw\_2, maggiore priorità

warning\_sw\_1\_tb <= '0'; -- Example input for warning\_sw\_1

wait for 100 ns;

in\_sw\_porta\_1\_tb <= "01"; -- Example input for in\_sw\_porta\_1

wait for 20 ns;

in\_sw\_porta\_2\_tb <= "10"; -- Example input for in\_sw\_porta\_2

wait for 20 ns;

adr\_sw\_source\_porta\_1\_tb <= '0'; -- Example input for adr\_sw\_source\_porta\_1

wait for 20 ns;

adr\_sw\_des\_porta\_1\_tb <= '0'; -- Example input for adr\_sw\_des\_porta\_1

wait for 20 ns;

adr\_sw\_source\_porta\_2\_tb <= '1'; -- Example input for adr\_sw\_source\_porta\_2

wait for 20 ns;

adr\_sw\_des\_porta\_2\_tb <= '1'; -- Example input for adr\_sw\_des\_porta\_2

wait for 20 ns;

warning\_sw\_2\_tb <= '0'; -- Example input for warning\_sw\_1

wait for 20 ns;

warning\_sw\_1\_tb <= '1'; -- Example input for warning\_sw\_2

wait for 100 ns;

warning\_sw\_2\_tb <= '0'; -- Example input for warning\_sw\_2, maggiore priorità

warning\_sw\_1\_tb <= '0'; -- Example input for warning\_sw\_1

wait for 100 ns;

in\_sw\_porta\_1\_tb <= "01"; -- Example input for in\_sw\_porta\_1

wait for 20 ns;

in\_sw\_porta\_2\_tb <= "10"; -- Example input for in\_sw\_porta\_2

wait for 20 ns;

adr\_sw\_source\_porta\_1\_tb <= '0'; -- Example input for adr\_sw\_source\_porta\_1

wait for 20 ns;

adr\_sw\_des\_porta\_1\_tb <= '0'; -- Example input for adr\_sw\_des\_porta\_1

wait for 20 ns;

adr\_sw\_source\_porta\_2\_tb <= '1'; -- Example input for adr\_sw\_source\_porta\_2

wait for 20 ns;

adr\_sw\_des\_porta\_2\_tb <= '1'; -- Example input for adr\_sw\_des\_porta\_2

wait for 20 ns;

warning\_sw\_2\_tb <= '1'; -- Example input for warning\_sw\_1

wait for 20 ns;

warning\_sw\_1\_tb <= '0'; -- Example input for warning\_sw\_2

wait;

end process stimulus;

end architecture tb;

**Forma d’ondaImmagine che contiene testo, schermata, numero, diagramma

Descrizione generata automaticamente**

# 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 T

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity fftClassic is

8. port (

9. clk : in std\_logic;

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

11.

12.

13. Y : out std\_logic

14.

15. );

16. end entity fftClassic;

17.

18.

19. architecture rtl of fftClassic is

20.

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

22.

23. begin

24. ff: process (clk, reset)

25. begin

26. if (reset='1') then

27. T<='0';

28. elsif falling\_edge(clk) then

29. T<= not T;

30. end if;

31. end process;

32.

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

34.

35. end architecture rtl;

## 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.

## Sistema 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(7 downto 0);

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

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(3 downto 0) when selection = '0' else --i primi valori

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

22. "----";

23.

24. end architecture dataflow;

25.

## Contatore Modulo 16

1. library IEEE;

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

3. use IEEE.numeric\_std.all;

4.

5.

6.

7. entity contMod16 is

8. port (

9. clk : in std\_logic;

10. reset : in std\_logic;

11.

12. cont: out std\_logic\_vector(3 downto 0)

13. );

14. end entity contMod16;

15.

16.

17. architecture rtl of contMod16 is

18.

19. component fftClassic

20. port(

21. clk : in std\_logic;

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

23.

24. Y : out std\_logic

25. );

26. end component;

27.

28.

29. signal wirings : std\_logic\_vector (3 downto 0);

30. signal reset\_count : std\_logic;

31.

32. begin

33.

34.

35. ff0: fftClassic port map(

36. clk => clk,

37. reset => reset\_count,

38.

39. Y=> wirings(0)

40. );

41.

42.

43. gen : for i in 1 to 3 generate

44.

45. ffn: fftClassic port map(

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

47. reset => reset\_count,

48.

49. Y=> wirings(i)

50. );

51. end generate;

52.

53.

54. cont <= wirings;

55.

56.

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

58. begin

59. reset\_count<='0';

60.

61. if(reset='1') then

62. reset\_count<='1';

63.

64. end if;

65.

66. end process;

67.

68. end architecture rtl;

69.

## Contatore comportamentale

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity contatoreB is

generic(

k: integer:=4;

max\_count: integer:=16

);

port (

clk: in std\_logic;

reset: in std\_logic;

set: in std\_logic;

input\_to\_load: in std\_logic\_vector(k downto 0);

enable: in std\_logic;

counter\_out: out std\_logic\_vector(k downto 0);

divider: out std\_logic:='0'

);

end entity contatoreB;

architecture rtl of contatoreB is

signal c: std\_logic\_vector(k downto 0) := (others=> '0');

begin

counter\_out <= c;

process (clk)

begin

if(reset ='1') then

c<=(others => '0');

divider<='1';

elsif set='1' then

c<=input\_to\_load;

divider<='1';

elsif falling\_edge(enable) then

if(to\_integer(unsigned(c))>=max\_count) then

c<= (others => '0');

divider<='0';

else

divider<='1';

c<= std\_logic\_vector(unsigned(c)+1);

end if;

end if;

end process;

end architecture rtl;

### 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>