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. 2024-25

Studenti:

Alessandro Campanella M63001697

Cesare Pulcrano M63001243

Sommario

[**Capitolo 1: Reti combinatorie elementari** 1](#_Toc197861193)

[Esercizio 1: Multiplexer 16:1 1](#_Toc197861194)

[Esercizio 1.1 1](#_Toc197861195)

[Progetto e architettura 1](#_Toc197861196)

[Implementazione 2](#_Toc197861197)

[Simulazione 3](#_Toc197861198)

[Esercizio 1.2 4](#_Toc197861199)

[Progetto e architettura 4](#_Toc197861200)

[Implementazione 5](#_Toc197861201)

[Simulazione 6](#_Toc197861202)

[Esercizio 1.3 7](#_Toc197861203)

[Sintesi su board di sviluppo 8](#_Toc197861204)

[Esercizio 2: Sistema ROM + M 10](#_Toc197861205)

[Esercizio 2.1 10](#_Toc197861206)

[Progetto e architettura 10](#_Toc197861207)

[Implementazione 10](#_Toc197861208)

[Simulazione 12](#_Toc197861209)

[Esercizio 2.2 13](#_Toc197861210)

[Sintesi su board di sviluppo 13](#_Toc197861211)

[**Capitolo 2: Reti sequenziali elementari** 15](#_Toc197861212)

[Esercizio 3: Riconoscitore di sequenze 15](#_Toc197861213)

[Esercizio 3.1 15](#_Toc197861214)

[Progetto e architettura 15](#_Toc197861215)

[Implementazione 16](#_Toc197861216)

[Simulazione 16](#_Toc197861217)

[Esercizio 3.2 18](#_Toc197861218)

[Sintesi su board di sviluppo 20](#_Toc197861219)

[Esercizio 4: Shift Register 20](#_Toc197861220)

[Esercizio 4.1 20](#_Toc197861221)

[Progetto e architettura 20](#_Toc197861222)

[Approccio Behavioral 20](#_Toc197861223)

[Approccio Structural 20](#_Toc197861224)

[Implementazione 20](#_Toc197861225)

[Approccio Behavioral 24](#_Toc197861226)

[Approccio Structural 23](#_Toc197861227)

[Simulazione 25](#_Toc197861228)

[Esercizio 5: Cronometro 25](#_Toc197861229)

[Esercizio 5.1 25](#_Toc197861230)

[Progetto e architettura 26](#_Toc197861231)

[Implementazione 26](#_Toc197861232)

[Simulazione 27](#_Toc197861233)

[Esercizio 5.2 29](#_Toc197861234)

[Sintesi su board di sviluppo 31](#_Toc197861235)

[Esercizio 5.3 (solo 9 CFU) 31](#_Toc197861236)

[Sintesi su board di sviluppo 31](#_Toc197861237)

[Esercizio 6: Sistema di lettura-elaborazione-scrittura PO\_PC 31](#_Toc197861238)

[Esercizio 6.1 31](#_Toc197861239)

[Progetto e architettura 32](#_Toc197861240)

[Implementazione 32](#_Toc197861241)

[Simulazione 32](#_Toc197861242)

[Esercizio 6.2 32](#_Toc197861243)

[Sintesi su board di sviluppo 32](#_Toc197861244)

[**Capitolo 3: Macchine aritmetiche** 32](#_Toc197861245)

[Esercizio 7: Moltiplicatore di Booth 32](#_Toc197861246)

[Esercizio 7.1 32](#_Toc197861247)

[Progetto e architettura 32](#_Toc197861248)

[Implementazione 33](#_Toc197861249)

[Simulazione 33](#_Toc197861250)

[Esercizio 7.2 33](#_Toc197861251)

[Sintesi su board di sviluppo 33](#_Toc197861252)

[Esercizio 7BIS: Divisore Non-Restoring (solo 9 CFU) 33](#_Toc197861253)

[Esercizio 7BIS.1 33](#_Toc197861254)

[Progetto e architettura 33](#_Toc197861255)

[Implementazione 33](#_Toc197861256)

[Simulazione 33](#_Toc197861257)

[Esercizio 7BIS.2 33](#_Toc197861258)

[Sintesi su board di sviluppo 33](#_Toc197861259)

[**Capitolo 4: Comunicazione con handshaking** 34](#_Toc197861260)

[Esercizio 8: Comunicazione con handshaking 34](#_Toc197861261)

[Esercizio 8.1 34](#_Toc197861262)

[Progetto e architettura 34](#_Toc197861263)

[Implementazione 34](#_Toc197861264)

[Simulazione 34](#_Toc197861265)

[**Capitolo 5: Processore** 35](#_Toc197861266)

[Esercizio 9: Processore IJVM 35](#_Toc197861267)

[Progetto e architettura 35](#_Toc197861268)

[Implementazione 35](#_Toc197861269)

[Simulazione 35](#_Toc197861270)

[**Capitolo 6: Interfaccia seriale** 36](#_Toc197861271)

[Esercizio 10: Interfaccia UART 36](#_Toc197861272)

[Progetto e architettura 36](#_Toc197861273)

[Implementazione 36](#_Toc197861274)

[Simulazione 36](#_Toc197861275)

[Esercizio 10BIS: Interfaccia UART (solo 9 CFU) 36](#_Toc197861276)

[Sintesi su board di sviluppo 36](#_Toc197861277)

[**Capitolo 7: Switch multistadio** 37](#_Toc197861278)

[Esercizio 11: Switch multistadio 37](#_Toc197861279)

[Esercizio 11.1 37](#_Toc197861280)

[Progetto e architettura 37](#_Toc197861281)

[Implementazione 37](#_Toc197861282)

[Simulazione (?) 37](#_Toc197861283)

[Esercizio 11.2 (solo 9 CFU) 37](#_Toc197861284)

[Progetto e architettura 37](#_Toc197861285)

[Implementazione 37](#_Toc197861286)

[Simulazione (?) 37](#_Toc197861287)

[Esercizio 11.3 (solo 9 CFU) 37](#_Toc197861288)

[Progetto e architettura 37](#_Toc197861289)

[Implementazione 38](#_Toc197861290)

[Simulazione (?) 38](#_Toc197861291)

[**Capitolo 8: Esercizio prova di esame dicembre 2024** 39](#_Toc197861292)

[Esercizio 12: Prova di esame del 19 dicembre 2024 39](#_Toc197861293)

[Progetto e architettura 39](#_Toc197861294)

[Implementazione 39](#_Toc197861295)

[Simulazione (?) 39](#_Toc197861296)

[**Appendice** 40](#_Toc197861297)

[Multiplexer 4:1 40](#_Toc197861298)

[Progetto e architettura 40](#_Toc197861299)

[Implementazione 40](#_Toc197861300)

[Button Debouncer 40](#_Toc197861301)

[Progetto e architettura 40](#_Toc197861302)

[Implementazione 41](#_Toc197861303)

[Divisore di frequenza 42](#_Toc197861304)

[Progetto e architettura 42](#_Toc197861305)

[Implementazione 42](#_Toc197861306)

# 

# **Capitolo 1: Reti combinatorie elementari**

## Esercizio 1: Multiplexer 16:1

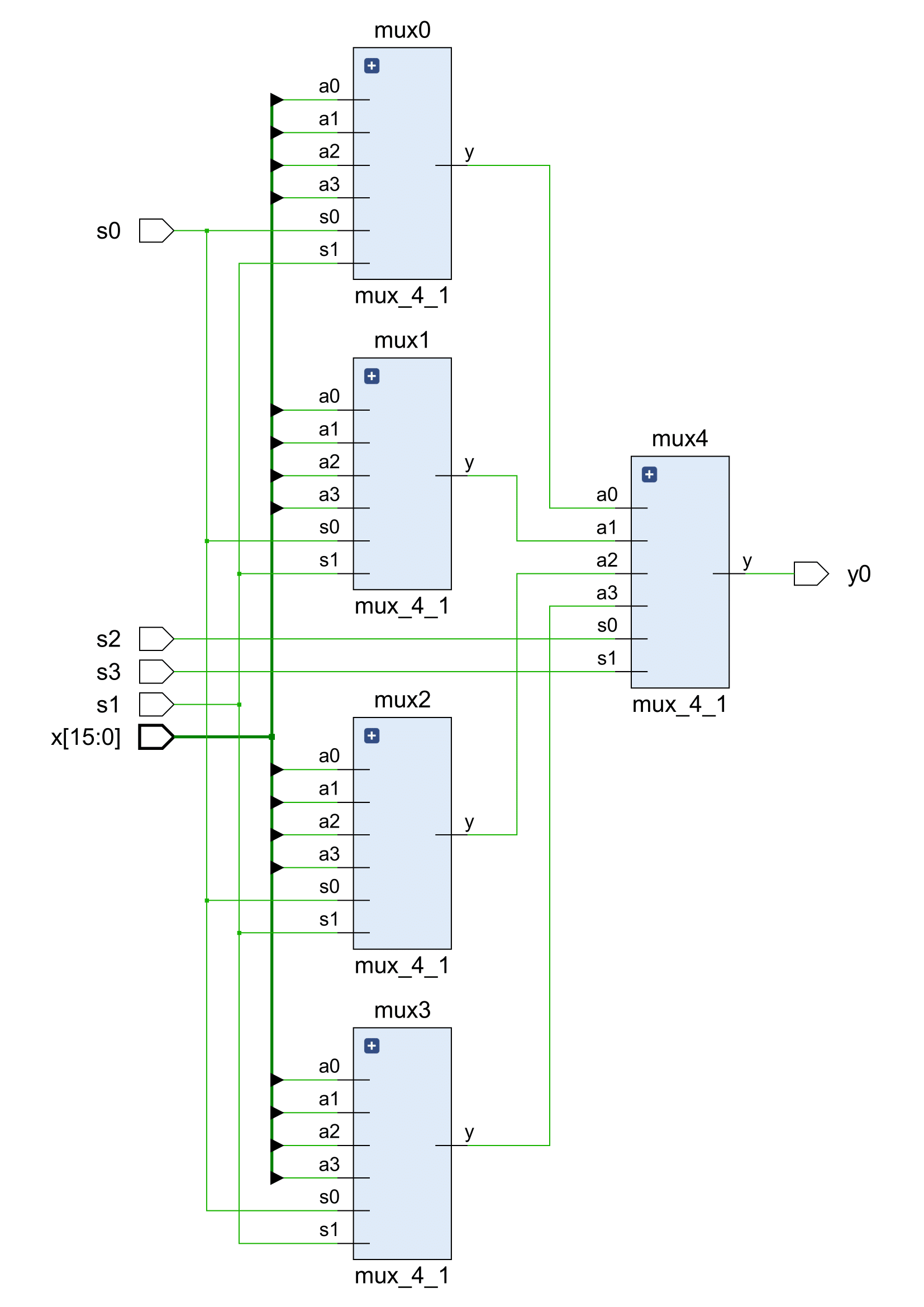
### Esercizio 1.1

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

Per realizzare un multiplexer indirizzabile 16:1, sono stati combinati cinque multiplexer 4:1 ([vedi appendice](#_Multiplexer_4:1)) seguendo un approccio **Structural**.

Un MUX 16:1 convoglia una delle 16 linee di ingresso verso un’unica uscita. Seguendo un’architettura strutturale *ad albero*, si è reso necessario utilizzare quattro MUX 4:1 al primo livello della struttura, in modo da realizzare i 16 input, e un MUX 4:1 al secondo livello per raccogliere le quattro uscite del livello precedente e convogliare solo una di esse verso l’uscita finale.

****

#### Implementazione

Gli ingressi sono stati formattati come uno std\_logic\_vector di 16 locazioni, le selezioni e l’output invece sono dei semplici input std\_logic.

entity mux\_16\_1 is

Port ( x : in STD\_LOGIC\_VECTOR (15 downto 0);

s0 : in STD\_LOGIC;

s1 : in STD\_LOGIC;

s2 : in STD\_LOGIC;

s3 : in STD\_LOGIC;

y0 : out STD\_LOGIC

);

end mux\_16\_1;

architecture Structural of mux\_16\_1 is

signal u0 : STD\_LOGIC := '0';

signal u1 : STD\_LOGIC := '0';

signal u2 : STD\_LOGIC := '0';

signal u3 : STD\_LOGIC := '0';

component mux\_4\_1

port( a0 : in STD\_LOGIC;

a1 : in STD\_LOGIC;

a2 : in STD\_LOGIC;

a3 : in STD\_LOGIC;

s0 : in STD\_LOGIC; -- selezione

s1 : in STD\_LOGIC; -- selezione

y : out STD\_LOGIC

);

end component;

begin

mux0 : mux\_4\_1

Port map( a0 => x(0),

a1 => x(1),

a2 => x(2),

a3 => x(3),

s0 => s0,

s1 => s1,

y => u0

);

mux1 : mux\_4\_1

Port map( a0 => x(4),

a1 => x(5),

a2 => x(6),

a3 => x(7),

s0 => s0,

s1 => s1,

y => u1

);

mux2 : mux\_4\_1

Port map( a0 => x(8),

a1 => x(9),

a2 => x(10),

a3 => x(11),

s0 => s0,

s1 => s1,

y => u2

);

mux3 : mux\_4\_1

Port map( a0 => x(12),

a1 => x(13),

a2 => x(14),

a3 => x(15),

s0 => s0,

s1 => s1,

y => u3

);

mux4 : mux\_4\_1

Port map( a0 => u0,

a1 => u1,

a2 => u2,

a3 => u3,

s0 => s2,

s1 => s3,

y => y0

);

end Structural;

#### Simulazione

Per simulare il componente, è bastato realizzare un vettore di input e uno di controllo attraverso dei signal. Aggiornando i segnali nella testbench, si è potuto osservare, in un diagramma temporale di simulazione, che il MUX 16:1 strutturale porta in output il dato richiesto dalle linee di selezione.

entity mux\_16\_1\_tb is

end mux\_16\_1\_tb;

architecture behavioral of mux\_16\_1\_tb is

component mux\_16\_1

port( x : in std\_logic\_vector (15 downto 0);

s0 : in STD\_LOGIC;

s1 : in STD\_LOGIC;

s2 : in STD\_LOGIC;

s3 : in STD\_LOGIC;

y0 : out STD\_LOGIC

);

end component;

signal input : std\_logic\_vector (15 downto 0 ) := (others => 'U');

signal control : std\_logic\_vector (3 downto 0) := (others => 'U');

signal output : std\_logic :='U';

begin

uut: entity work.mux\_16\_1(Structural)

port map( x => input,

s0 => control(0),

s1 => control(1),

s2 => control(2),

s3 => control(3),

y0 => output

);

stim\_proc: process

begin

wait for 100 ns;

input <= "1000000010000000";

wait for 50 ns;

control <= "0000";

wait for 50 ns;

control <="1111";

wait for 50 ns;

control <= "0111";

wait for 50 ns;

assert output = '0'

report "error"

severity failure;

wait;

end process;

end;

I risultati della simulazione sono riportati di seguito: l’output diventa alto solo nell’istante in cui si richiede al componente di convogliare il dato alla linea di input “1111” (15), sulla quale è presente un segnale alto [200.000 ns]. Analogo comportamento avviene per il dato alla linea “0111” (8) [250.000 ns].

Nel momento in cui, invece, si richiede il dato alla linea “0000” (0), l’uscita è bassa, in quanto il dato trasportato in output è proprio il valore logico “0” [150.000 ns].

Immagine che contiene schermata, testo, Software multimediale, software

Descrizione generata automaticamente

### Esercizio 1.2

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**.

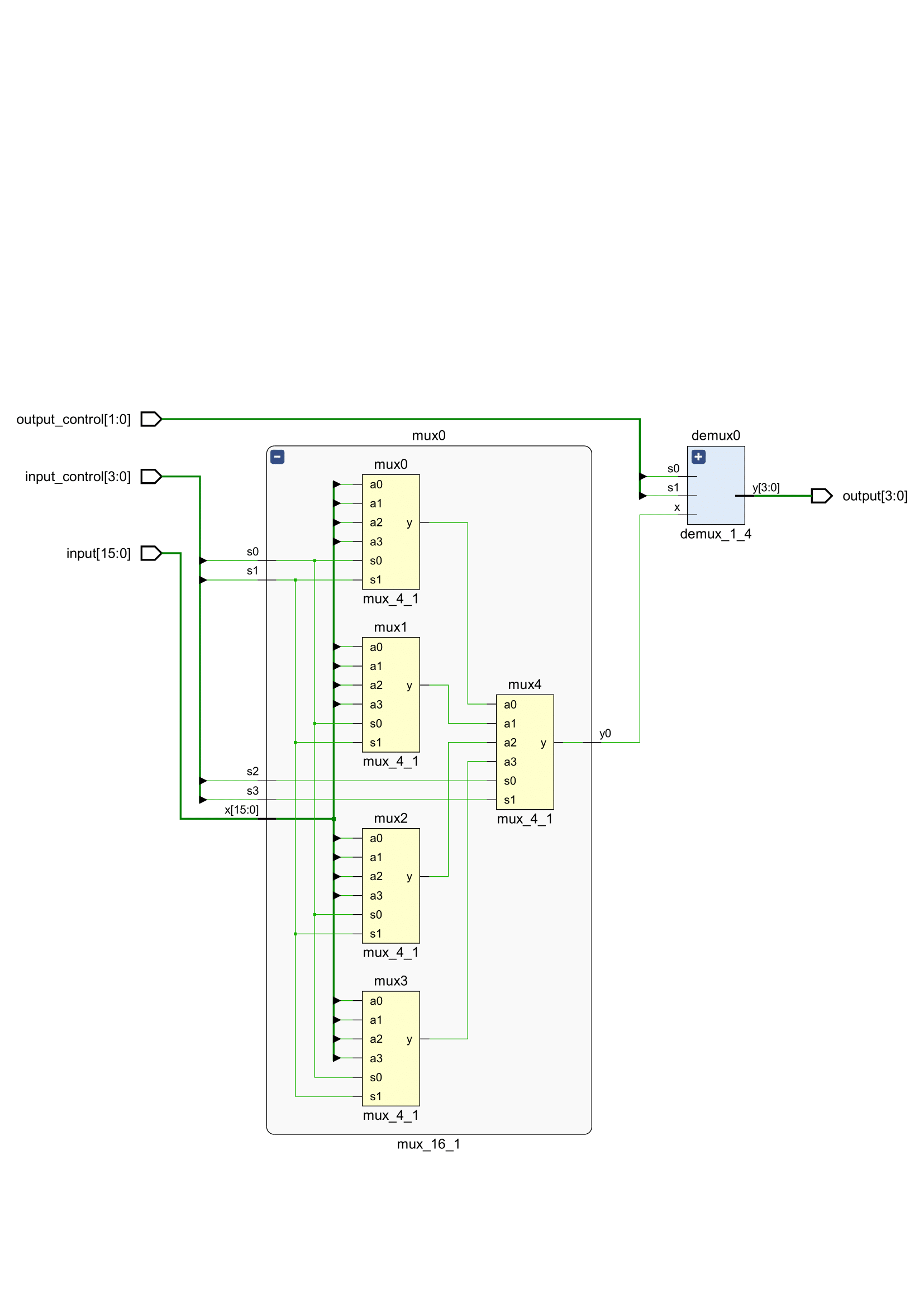
#### Progetto e architettura

Per trasportare un dato, scelto tra 16 sorgenti lungo una delle 4 destinazioni, è necessario realizzare un’architettura **Structural** che unisca un MUX 16:1, realizzato [al punto precedente](#_Esercizio_1.1), e un DEMUX 1:4, un nuovo componente.

Un **demultiplexer indirizzabile** **1:4** è un componente puramente combinatorio che, a differenza del MUX, convoglia l’unico dato disponibile in input lungo una delle quattro uscite, selezionata opportunamente mediante un ingresso di selezione.

La rete di interconnessione, dunque, è formata da un MUX 16:1 (16 input e 4 linee di selezione) e un DEMUX 1:4 (4 output e 2 linee di selezione). L’uscita del MUX è collegata all’ingresso del DEMUX.

Alla pagina successiva è mostrato il design della rete complessiva:

****

#### Implementazione

Il demultiplexer indirizzabile è stato realizzato seguendo un approccio **Behavioral**:

entity demux\_1\_4 is

port( x : in std\_logic;

s0 : in std\_logic;

s1 : in std\_logic;

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

);

end demux\_1\_4;

architecture Behavioral of demux\_1\_4 is

begin

process(x,s1,s0)

begin

y<="0000";

if(s1='0' and s0 = '0') then

y(0)<=x;

elsif(s1='0' and s0 = '1') then

y(1)<=x;

elsif(s1='1' and s0 = '0') then

y(2)<=x;

elsif(s1='1' and s0 = '1') then

y(3)<=x;

end if;

end process;

end Behavioral;

Nel componente principale, invece (approccio **Structural**), tutti gli ingressi, le selezioni e le uscite sono state realizzate come std\_logic\_vector. È stato realizzato, inoltre, un signal demux\_in per collegare fisicamente l’uscita del multiplexer 16:1 all’ingresso del demultiplexer 1:4.

entity net\_16\_4 is

Port ( input : in std\_logic\_vector(15 downto 0);

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

output\_control : in std\_logic\_vector (1 downto 0);

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

);

end net\_16\_4;

architecture Structural of net\_16\_4 is

signal demux\_in : std\_logic :='0';

component mux\_16\_1

port( x : in std\_logic\_vector (15 downto 0);

s0 : in std\_logic;

s1 : in std\_logic;

s2 : in std\_logic;

s3 : in std\_logic;

y0 : out std\_logic

);

end component;

component demux\_1\_4

port( x : in std\_logic;

s0 : in std\_logic;

s1 : in std\_logic;

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

);

end component;

begin

mux0: mux\_16\_1

Port map( x=>input,

s0=>input\_control(0),

s1=>input\_control(1),

s2=>input\_control(2),

s3=>input\_control(3),

y0=>demux\_in

);

demux0: demux\_1\_4

Port map( x=>demux\_in,

s0=>output\_control(0),

s1=>output\_control(1),

y=>output

);

end Structural;

#### Simulazione

Per simulare il componente, si è proceduto in maniera analoga all’[esercizio precedente](#_Simulazione): sono stati creati dei segnali per simulare ingressi, selezioni e uscite.

entity net\_16\_4\_tb is

end net\_16\_4\_tb;

architecture Behavioral of net\_16\_4\_tb is

component net\_16\_4

Port ( input : in std\_logic\_vector (15 downto 0);

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

output\_control : in std\_logic\_vector (1 downto 0);

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

);

end component;

signal input\_test : std\_logic\_vector (15 downto 0) :=(others=>'0');

signal input\_control\_test : std\_logic\_vector (3 downto 0) :=(others=>'0');

signal output\_control\_test : std\_logic\_vector (1 downto 0) :=(others=>'0');

signal output\_test : std\_logic\_vector (3 downto 0) :=(others=>'0');

begin

uut: entity work.net\_16\_4(Structural)

Port map( input=>input\_test,

input\_control=>input\_control\_test,

output\_control=>output\_control\_test,

output=>output\_test

);

stim\_proc: process

begin

input\_test<="1000000000000000";

input\_control\_test<="1111";

output\_control\_test<="01";

wait for 10 ns;

input\_control\_test<="0111";

output\_control\_test<="10";

wait for 10 ns;

input\_test<="1001001000010000";

input\_control\_test<="0100";

output\_control\_test<="11";

wait for 10 ns;

input\_control\_test<="1110";

output\_control\_test<="00";

wait for 10 ns;

assert output\_test="0000";

report "errore"

severity failure;

end process;

end Behavioral;

Il risultato della simulazione appare come segue:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

### Esercizio 1.3

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 realizzare il componente sulla board FPGA fornita, è necessario fornire, in aggiunta ai segnali visti [al punto precedente](#_Esercizio_1.2), il **segnale di clock** della board. Infatti, a differenza di switch e LED che possono essere codificati anche in una logica puramente combinatoria, i bottoni hanno bisogno di un fronte del clock (salita o discesa) al fine di riconoscere il livello logico (basso/alto) del bottone.

La gestione del riconoscimento del livello del bottone è affidata a un Button Debouncer, discusso nell’[appendice](#_Button_Debouncer). I bottoni sono stati gestiti in maniera sincrona.

Utilizzando i primi 8 switch, si procede come segue:

* al clic di BTNL (bottone sinistro) i valori degli switch verranno salvati come i primi 8 LSB della stringa in input alla rete di interconnessione;
* al clic di BTNR (bottone destro) i valori degli switch verranno salvati come gli ultimi 8 MSB della stringa in input alla rete di interconnessione;

Per fare ciò, è stato pensato un componente **Behavioral** switch\_8\_capture, mostrato di seguito:

entity switch\_8\_capture is

Port ( CLOCK : in std\_logic; -- Clock signal

BTN : in std\_logic; -- Button input

SWITCH\_INPUT : in std\_logic\_vector(7 downto 0); -- 8 input

OUTPUT : out std\_logic\_vector(7 downto 0) -- Input + selezioni

);

end switch\_8\_capture;

architecture Behavioral of switch\_8\_capture is

begin

process(CLOCK, BTN, SWITCH\_INPUT)

begin

if (rising\_edge(CLOCK)) then

if(BTN = '1') then -- Click del bottone

OUTPUT(7 downto 0) <= SWITCH\_INPUT; -- Input

end if;

end if;

end process;

end Behavioral;

Il componente completo, invece, è realizzato secondo un approccio **Structural**, come segue:

entity net\_16\_4\_fpga is

Port ( CLOCK : in std\_logic; -- Clock signal

BTNL : in std\_logic; -- Button input 1

BTNR : in std\_logic; -- Button input 2

SWITCH\_INPUT : in std\_logic\_vector(7 downto 0); -- 16 input

SWITCH\_SEL\_IN : in std\_logic\_vector(3 downto 0); -- 4 selezioni input

SWITCH\_SEL\_OUT : in std\_logic\_vector(1 downto 0); -- 2 selezioni output

LED : out std\_logic\_vector(3 downto 0) -- 4 LED

);

end net\_16\_4\_fpga;

architecture Structural of net\_16\_4\_fpga is

signal switch\_to\_net : std\_logic\_vector(15 downto 0);

component switch\_8\_capture

Port ( CLOCK : in std\_logic; -- Clock signal

BTN : in std\_logic; -- Button input

SWITCH\_INPUT : in std\_logic\_vector(7 downto 0); -- 8 input

OUTPUT : out std\_logic\_vector(7 downto 0) -- Input

);

end component;

component net\_16\_4 is

Port ( input : in std\_logic\_vector(15 downto 0);

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

output\_control : in std\_logic\_vector (1 downto 0);

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

);

end component;

begin

switch\_0 : switch\_8\_capture

Port map( CLOCK => CLOCK,

BTN => BTNL, -- Button input 1

SWITCH\_INPUT => SWITCH\_INPUT, -- 8 input

OUTPUT => switch\_to\_net(7 downto 0) -- Input

);

switch\_1 : switch\_8\_capture

Port map( CLOCK => CLOCK,

BTN => BTNR, -- Button input 2

SWITCH\_INPUT => SWITCH\_INPUT, -- 8 input

OUTPUT => switch\_to\_net(15 downto 8) -- Input

);

net : net\_16\_4

Port map( input => switch\_to\_net,

input\_control => SWITCH\_SEL\_IN,

output\_control => SWITCH\_SEL\_OUT,

output => LED

);

end Structural;

Come si può osservare, per realizzare gli ingressi di selezione sono stati utilizzati gli altri switch disponibili sulla board.

I *constraints* del progetto sono i seguenti:

## 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 { SWITCH\_INPUT[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

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

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

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

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

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

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

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

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

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

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

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

set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { SWITCH\_SEL\_OUT[0] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { SWITCH\_SEL\_OUT[1] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

## LEDs

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

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

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { LED[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]

## Buttons

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

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

## Esercizio 2: Sistema ROM + M

### Esercizio 2.1

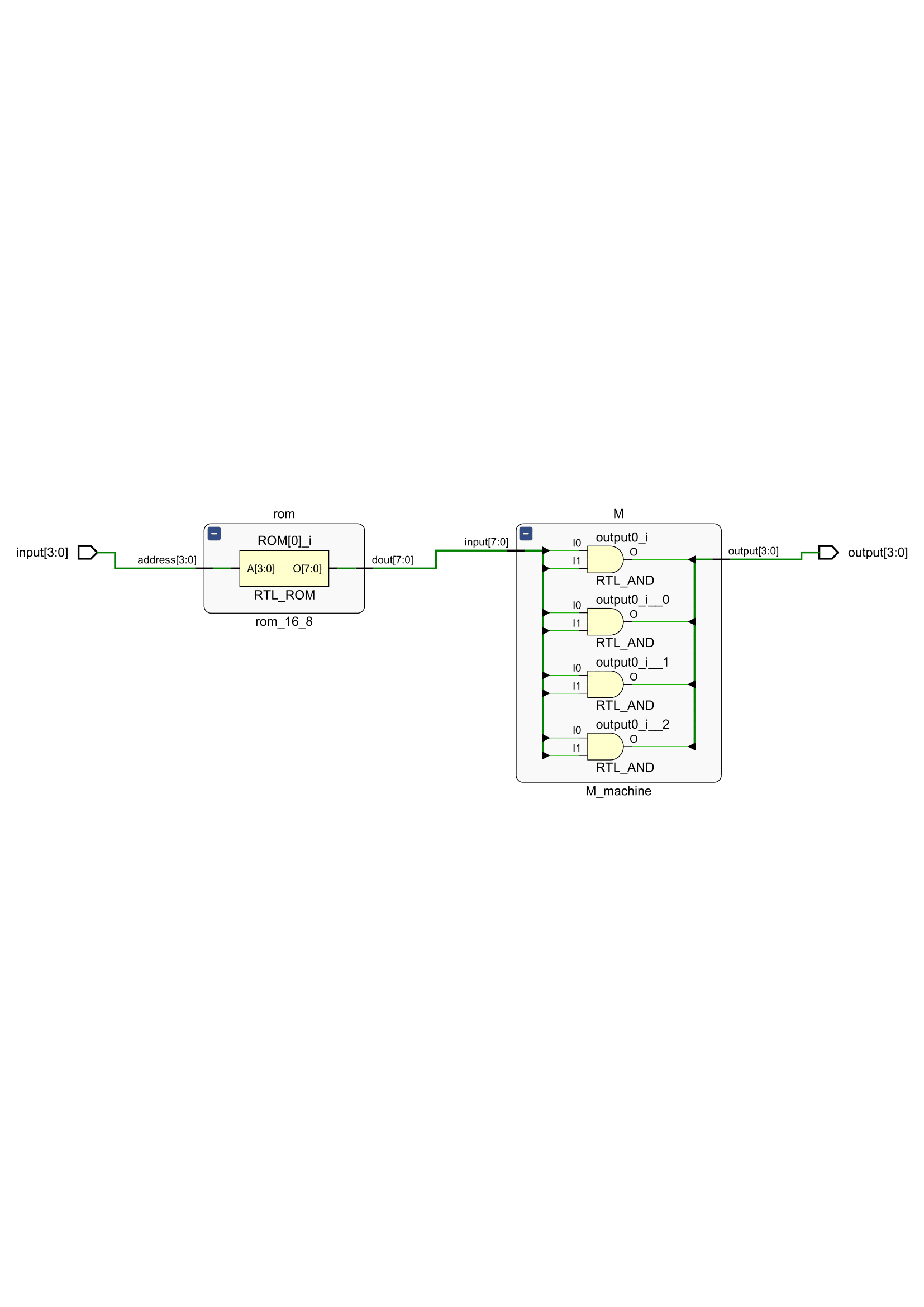
Progettare, implementare in VHDL e testare mediante simulazione un **sistema S** composto da una **ROM** puramente combinatoria di 16 locazioni da 8 bit ciascuna e da una macchina combinatoria **M** che opera come segue: fornito al sistema un indirizzo A di 4 bit, il sistema restituisce il valore contenuto 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

Il primo passo consiste nel realizzare la memoria ROM da cui il sistema preleverà i dati. Per realizzare 16 locazioni da 8 bit ciascuna, è stato necessario realizzare una matrice di tale dimensione. Ogni riga della matrice corrisponderà a una cella di memoria da 8 bit. La ROM trasporta il dato in uscita, noto l’indirizzo di memoria; per realizzare ciò, l’approccio seguito è stato di tipo **Dataflow**.

Come funzione per la macchina combinatoria M, è stata scelta la *AND bitwise*, realizzata seguendo un approccio **Dataflow**.

Il sistema S complessivo è stato realizzato secondo un approccio **Structural**.



#### Implementazione

La memoria ha un solo input address, che consente di accedere a uno specifico indirizzo di memoria, e un unico output dout, corrispondente al dato in memoria all’indirizzo scelto.

La ROM è un array di 16 locazioni; ogni riga di tale vettore, invece, è uno std\_logic\_vector di 8 locazioni. È stato necessario definire un nuovo tipo MEMORY\_16\_8 per istanziare la matrice. Il contenuto della ROM è constant e arbitrario.

Il dato in uscita viene aggiornato, seguendo una logica puramente combinatoria, con il valore in memoria all’indirizzo indicato. L’indirizzo è convertito opportunamente da binario a intero grazie al metodo to\_integer.

entity rom\_16\_8 is

port(

address : in std\_logic\_vector(3 downto 0); -- 2^4 locazioni

dout : out std\_logic\_vector(7 downto 0) -- Locazioni

);

end entity rom\_16\_8;

architecture Dataflow of rom\_16\_8 is

type MEMORY\_16\_8 is array (0 to 15) of std\_logic\_vector(7 downto 0); -- Matrice 16x8

constant ROM : MEMORY\_16\_8 := ( -- ROM content

x"1A",

x"2B",

x"3C",

x"4D",

x"5E",

x"6F",

x"71",

x"82",

x"93",

x"A4",

x"B5",

x"C6",

x"D7",

x"E8",

x"F9",

x"3A"

);

begin

process(address)

begin

dout <= ROM(to\_integer(unsigned(address))); -- Casting da unsigned a intero

end process;

end architecture Dataflow;

La macchina M, dato un vettore in input di 8 locazioni, produce in output un nuovo vettore di 4 locazioni, dove .

entity M\_machine is

port (

input : in std\_logic\_vector(7 downto 0); -- Valore contenuto nella ROM

output : out std\_logic\_vector(3 downto 0) -- Output trasformato

);

end M\_machine;

-- AND bitwise

architecture Dataflow of M\_machine is

begin

output(0) <= input(0) AND input(4);

output(1) <= input(1) AND input(5);

output(2) <= input(2) AND input(6);

output(3) <= input(3) AND input(7);

end Dataflow;

Il sistema S complessivo prende l’indirizzo desiderato come unico input, e restituisce un opportuno output trasformando il dato in ROM mediante la macchina M.

entity S\_system is

port (

input : in std\_logic\_vector(3 downto 0); -- Locazione ROM

output : out std\_logic\_vector(3 downto 0) -- AND bitwise

);

end S\_system;

architecture Structural of S\_system is

signal rom\_to\_M : std\_logic\_vector(7 downto 0) := (others => 'U'); -- Link tra ROM e M

component rom\_16\_8 is

port(

address : in std\_logic\_vector(3 downto 0); -- 2^4 locazioni

dout : out std\_logic\_vector(7 downto 0) -- Locazioni

);

end component;

component M\_machine is

port (

input : in std\_logic\_vector(7 downto 0); -- Valore contenuto nella ROM

output : out std\_logic\_vector(3 downto 0) -- Output trasformato

);

end component;

begin

rom : rom\_16\_8

Port map(

address => input,

dout => rom\_to\_M

);

M : M\_machine

Port map(

input => rom\_to\_M,

output => output

);

end Structural;

#### Simulazione

Per simulare il sistema, un ciclo for-loop ha scandito tutte le locazioni della memoria mostrando i rispettivi output per 50 ns.

entity S\_system\_tb is

end S\_system\_tb;

architecture Behavioral of S\_system\_tb is

component S\_system is

port (

input : in std\_logic\_vector(3 downto 0); -- Locazione ROM

output : out std\_logic\_vector(3 downto 0) -- AND bitwise

);

end component;

signal input\_test : std\_logic\_vector(3 downto 0) := (others => 'U');

signal output\_test : std\_logic\_vector(3 downto 0) := (others => 'U');

begin

uut : entity work.S\_system(Structural) -- unity under test

Port map(

input => input\_test,

output => output\_test

);

stim\_proc : process

begin

wait for 10 ns;

for i in 0 to 15 loop

input\_test <= std\_logic\_vector(to\_unsigned(i, 4)); -- Conversione da decimale a binario

wait for 50 ns;

end loop;

assert output\_test = "0000"

report "error"

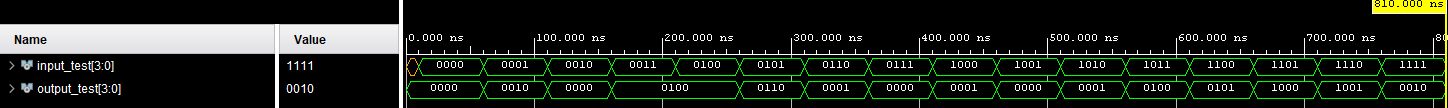
severity failure;

wait;

end process;

end ;

Il risultato è il seguente:



È stata eseguita una verifica manuale, locazione per locazione, per verificare il corretto funzionamento della AND bitwise del contenuto di ogni cella; gli output sono risultati corretti.

### Esercizio 2.2

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 soddisfare la richiesta, è stato sufficiente mappare i bit necessari per codificare 16 indirizzi (4 switch) sugli switch a disposizione, e le 4 uscite sui LED a disposizione.

entity S\_system\_fpga is

port (

SW : in std\_logic\_vector(3 downto 0); -- Locazione ROM

LED : out std\_logic\_vector(3 downto 0) -- AND bitwise

);

end S\_system\_fpga;

architecture Structural of S\_system\_fpga is

component S\_system is

port (

input : in std\_logic\_vector(3 downto 0); -- Locazione ROM

output : out std\_logic\_vector(3 downto 0) -- AND bitwise

);

end component;

begin

S : S\_system

Port map(

input => SW,

output => LED

);

end Structural;

I *constraint*s per il progetto in analisi sono i seguenti:

## 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]

## LEDs

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

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

set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { LED[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]

# **Capitolo 2: Reti sequenziali elementari**

## Esercizio 3: Riconoscitore di sequenze

### Esercizio 3.1

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

Un riconoscitore di sequenza è una macchina sequenziale in grado di rilevare un segnale composto da più bit. L’obiettivo, in questo caso, è il riconoscimento della sequenza “101” seguendo due differenti modalità: **sequenze non sovrapposte** e **sequenze parzialmente sovrapposte**.

Il riconoscitore di sequenze va implementato come una FSM (Finite State Machine, o automa a stati finiti).

Si è deciso di realizzare un unico automa in grado di *switchare* tra le due modalità richieste in base al valore di . L’automa che segue gli stati rappresenta la modalità non sovrapposta, mentre gli stati indicano il funzionamento della modalità parzialmente sovrapposta.

**Immagine che contiene nero, oscurità

Descrizione generata automaticamente**

#### Implementazione

Realizzare una FSM in VHDL (modalità **Behavioral**) ha previsto la realizzazione di due processi:

* un **process combinatorio**, per aggiornare lo stato prossimo sulla base degli input e del modo;
* un **process sequenziale**, per realizzare la retroazione del sistema, aggiornando cioè lo stato corrente con il valore dello stato prossimo.

Si è creato un nuovo tipo stato per raccogliere in maniera coerente tutti gli stati dell’automa. Il processo combinatorio si occupa, con un case-when, di valutare l’azione da compiere per ogni stato presente nell’automa a stati finiti.

Il processo sequenziale opera sul fronte di salita del segnale di abilitazione richiesto dalla traccia.

entity riconoscitore\_101 is

Port ( input : in std\_logic;

a : in std\_logic; -- segnale di tempificazione (clock)

reset : in std\_logic;

m : in std\_logic; -- 0 modalità non sovrapposta, 1 modalità parz. sovrapposta

output : out std\_logic

);

end riconoscitore\_101;

architecture Behavioral of riconoscitore\_101 is

--da S0 a S4 stati automa 1 (m=0, non sovrapposto)

--da S5 a S7 stati automa 1 (m=1,parzialmente sovrapposto)

type stato is (S0,S1,S2,S3,S4,S5,S6,S7);

signal stato\_corrente : stato := S0;

signal stato\_prossimo : stato;

begin

state\_process: process(stato\_corrente,input,m)

begin

case stato\_corrente is

when S0 =>

if(m='0') then

if(input='0') then

stato\_prossimo<=S1;

output<='0';

else

stato\_prossimo<=S2;

output<='0';

end if;

else

stato\_prossimo<=S5;

output<='0';

end if;

when S1=>

if(m='0') then

stato\_prossimo<=S3;

output<='0';

else

stato\_prossimo<=S5;

output<='0';

end if;

when S2 =>

if(m='0') then

if(input='0') then

stato\_prossimo<=S4;

output<='0';

else

stato\_prossimo<=S3;

output<='0';

end if;

else

stato\_prossimo<=S5;

output<='0';

end if;

when S3 =>

if(m='0') then

stato\_prossimo<=S0;

output<='0';

else

stato\_prossimo<=S5;

output<='0';

end if;

when S4 =>

if(m='0') then

if(input='0') then

stato\_prossimo<=S0;

output<='0';

else

stato\_prossimo<=S0;

output<='1';

end if;

else

stato\_prossimo<=S5;

output<='0';

end if;

when S5 =>

if(m='1') then

if(input='0') then

stato\_prossimo<=S5;

output<='0';

else

stato\_prossimo<=S6;

output<='0';

end if;

else

stato\_prossimo<=S0;

output<='0';

end if;

when S6 =>

if(m='1') then

if(input='0') then

stato\_prossimo<=S7;

output<='0';

else

stato\_prossimo<=S6;

output<='0';

end if;

else

stato\_prossimo<=S0;

output<='0';

end if;

when S7 =>

if(m='1') then

if(input='0') then

stato\_prossimo<=S5;

output<='0';

else

stato\_prossimo<=S5;

output<='1';

end if;

else

stato\_prossimo<=S0;

output<='0';

end if;

end case;

end process;

memory\_process : process (a,reset)

begin

if(reset = '1') then

stato\_corrente<=S0;

elsif(a'event and a='1') then

stato\_corrente <= stato\_prossimo;

end if;

end process;

end Behavioral;

#### Simulazione

Per simulare il riconoscitore, è stato generato un segnale di clock, alzando e abbassando un opportuno signal per la metà del periodo completo del clock della board. È stato poi modificato il valore dell’input del riconoscitore, in modo da osservare se l’uscita si alzasse al riconoscimento della sequenza 101. È stato verificato il funzionamento per entrambi i modi.

entity riconoscitore\_101\_tb is

end riconoscitore\_101\_tb;

architecture Behavioral of riconoscitore\_101\_tb is

signal input\_tb: std\_logic:='U';

signal a\_tb: std\_logic:='U';

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

signal m\_tb : std\_logic:='U';

signal output\_tb: std\_logic:='U';

constant a\_period : time := 10ns;

component riconoscitore\_101

Port ( input : in std\_logic;

a : in std\_logic; -- segnale di tempificazione (clock)

reset : in std\_logic;

m : in std\_logic; -- 0 modalità non sovrapposta, 1 modalità parz. sovrapposta

output : out std\_logic

);

end component;

begin

uut: riconoscitore\_101

port map(

input=>input\_tb,

a=>a\_tb,

reset=>reset\_tb,

m=>m\_tb,

output=>output\_tb

);

clock\_process : process

begin

a\_tb<='0';

wait for a\_period/2;

a\_tb<='1';

wait for a\_period/2;

end process;

stim\_proc: process

begin

m\_tb<='0';

input\_tb<='0';

wait for 10 ns;

input\_tb<='0';

wait for 10 ns;

input\_tb<='0';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

input\_tb<='0';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

m\_tb<='1';

input\_tb<='1';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

input\_tb<='0';

wait for 10 ns;

input\_tb<='1';

wait for 10 ns;

assert output\_tb ='0'

report "error"

severity failure;

wait;

end process;

end Behavioral;

Il risultato è il seguente:

Immagine che contiene schermata, testo, linea

Descrizione generata automaticamente

### Esercizio 3.2

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

<se richiesto: descrizione dell’architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell’I/O); file di constraint utilizzato per il progetto>

## Esercizio 4: Shift Register

### Esercizio 4.1

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

**[finisci dopo averlo finito nel moltiplicatore]**

##### Approccio Behavioral

Considerando il contenuto dello shift register come un vettore, basterà valutare il livello logico di **Y** (shift di 1 o 2 bit) e di **s** (shift a destra o a sinistra) per aggiornare un opportuno vettore di appoggio con le posizioni aggiornate.

In base alla direzione dello shift e al numero di bit da shiftare, il vettore di appoggio salverà degli *slices* ridotti del vettore di partenza e aggiungerà i nuovi bit nelle locazioni rimanenti.

##### Approccio Structural

L’approccio strutturale prevede di istanziare tanti flip-flop quanti ne vengono richiesti dal parametro Generic, in quanto tale componente salva un singolo bit.

In input a ogni flip-flop , bisogna selezionare opportunamente un bit tra:

* il valore del FF precedente (shift di 1 bit a destra);
* il valore del FF successivo (shift di 1 bit a sinistra);
* il valore del FF precedente (shift di 2 bit a destra);
* il valore del FF successivo (shift di 2 bit a sinistra).

Dunque, per ogni flip-flop si è istanziato anche un [multiplexer 4:1](#_Multiplexer_4:1) per convogliare i dati. Il MUX è controllato opportunamente proprio dai segnali Y e s.

Vanno fatte inoltre delle considerazioni in base alla posizione del flip-flop per il salvataggio dell’input a seguito dello shift:

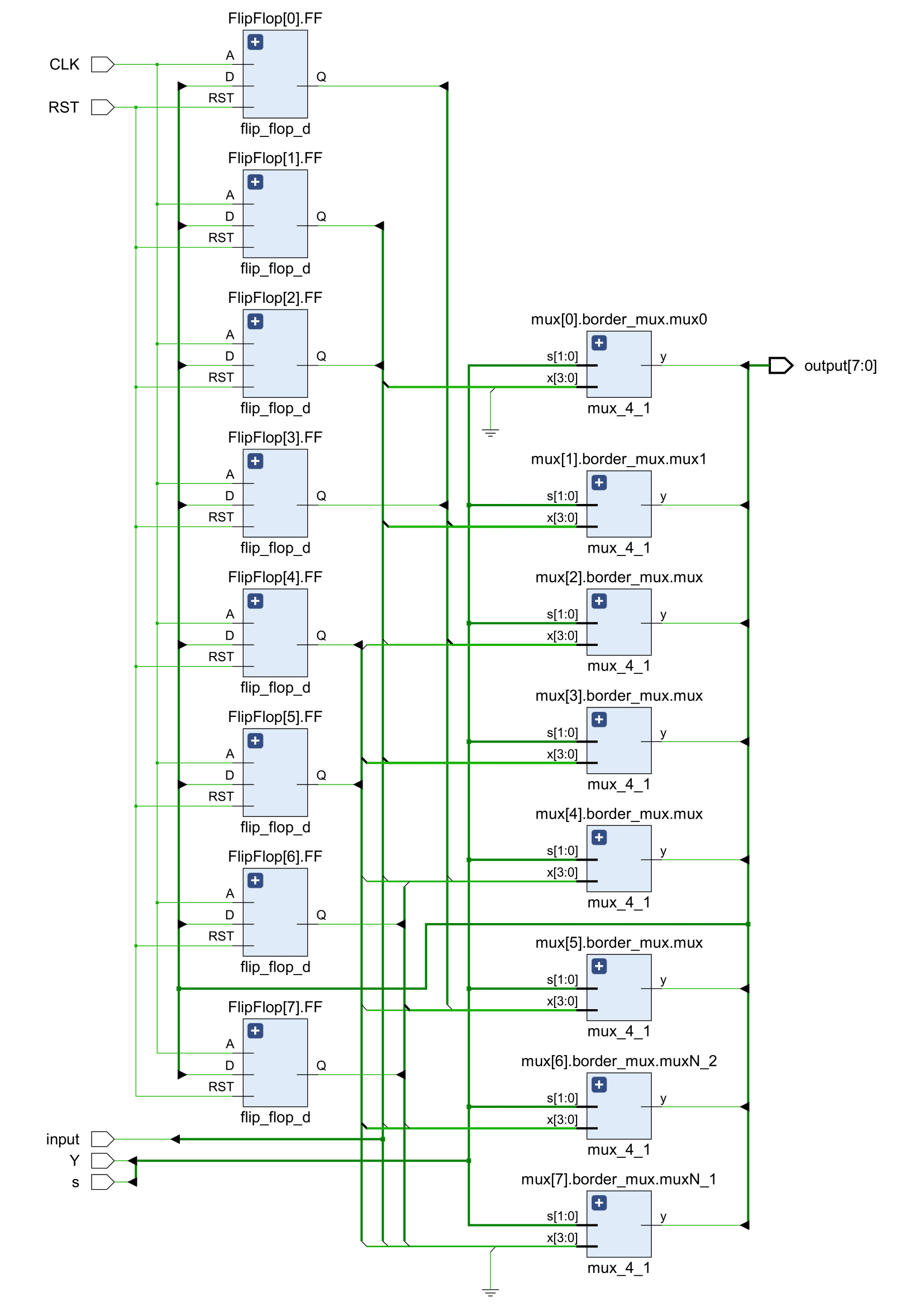
* per lo shift a destra di una posizione, il primo FF dovrà salvare il valore di input;
* per lo shift a sinistra di una posizione, l’ultimo FF dovrà salvare il valore di input.

Nel caso di shift a due posizioni, il secondo valore da shiftare sarà un bit ‘0’. In tal caso:

* per lo shift a destra di due posizioni, il primo FF dovrà salvare ‘0’ e il secondo FF dovrà salvare il valore di input;
* per lo shift a sinistra di due posizioni, l’ultimo FF dovrà salvare ‘0’ e il penultimo FF dovrà salvare il valore di input.

Si è reso necessario dunque effettuare un controllo nella generazione dei multiplexer: difatti, il primo multiplexer convoglierà sempre i dati dei flip-flop successivi, l’ultimo i dati dei flip-flop precedenti, mentre i flip-flop intermedi convoglieranno sia i dati provenienti dai flip-flop successivi che da quelli precedenti.

Un esempio di design per è il seguente:



#### Implementazione

##### Approccio Behavioral

Per realizzare l’approccio descritto in precedenza, il componente usufruisce di un segnale di appoggio output\_temp, che viene aggiornato con gli *slices* giusti in base al valore logico di s e Y.

Al di fuori del process incaricato di gestire a logica dello shift, viene aggiornata l’uscita dell’entity.

entity shift\_register\_behavioral is

Generic(N : integer := 8);

Port ( input : in std\_logic;

RST : in std\_logic;

LOAD : in std\_logic;

CLK : in std\_logic; -- Abilitazione

s : in std\_logic; -- 0 = shift a destra, 1 = shift a sinistra

Y : in std\_logic; -- 0 = shift di 1 bit, 1 = shift di 2 bit

output : out std\_logic\_vector (N-1 downto 0)

);

end shift\_register\_behavioral;

architecture Behavioral of shift\_register\_behavioral is

signal output\_temp : std\_logic\_vector (N-1 downto 0) := (others => '0');

begin

mem: process(CLK)

begin

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

if(RST = '1') then

output\_temp <= (others => '0');

elsif(LOAD = '1') then

case s is

when '0' => -- Shift a destra

if(Y = '0') then -- Shift di 1 bit

output\_temp(N-2 downto 0) <= output\_temp(N-1 downto 1);

output\_temp(N-1) <= input;

elsif(Y = '1') then -- Shift di 2 bit

output\_temp(N-3 downto 0) <= output\_temp(N-1 downto 2);

output\_temp(N-2) <= input;

output\_temp(N-1) <= '0';

end if;

when '1' => -- Shift a sinistra

if(Y = '0') then -- Shift di 1 bit

output\_temp(N-1 downto 1) <= output\_temp(N-2 downto 0);

output\_temp(0) <= input;

elsif(Y = '1') then -- Shift di 2 bit

output\_temp(N-1 downto 2) <= output\_temp(N-3 downto 0);

output\_temp(1) <= input;

output\_temp(0) <= '0';

end if;

when others =>

output\_temp <= (others => '0');

end case;

end if;

end if;

end process;

output <= output\_temp;

end Behavioral;

##### Approccio Structural

Per l’implementazione strutturale, si è ricorso al costrutto for-generate, che consente di generare dispositivi. Per quanto riguarda i flip-flop, è bastato generarne tanti quanti sono richiesti dal parametro . Per i MUX, sono stati inseriti controlli aggiuntivi sul numero di flip-flop da istanziare per gestire opportunamente le retroazioni, come spiegato al capitolo precedente.

entity shift\_register is

Generic(N : integer := 8);

Port ( input : in std\_logic;

RST : in std\_logic;

LOAD : in std\_logic;

CLK : in std\_logic; -- Abilitazione

s : in std\_logic; -- 0 = shift a destra, 1 = shift a sinistra

Y : in std\_logic; -- 0 = shift di 1 bit, 1 = shift di 2 bit

output : out std\_logic\_vector (N-1 downto 0)

);

end shift\_register;

architecture Structural of shift\_register is

signal retro : std\_logic\_vector(N-1 downto 0);-- := (others => 'U');

signal mux\_ff: std\_logic\_vector (N-1 downto 0);

-- component mux\_2\_1

-- Port ( x : in std\_logic\_vector(1 downto 0);

-- s : in std\_logic;

-- y : out std\_logic

-- );

-- end component;

component mux\_4\_1

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

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

y : out std\_logic

);

end component;

component flip\_flop\_d

Port ( D : in std\_logic;

A : in std\_logic;

RST : in std\_logic;

Q : out std\_logic

);

end component;

begin

mux:for i in N-1 downto 0 generate

border\_mux:if(i=0 ) generate

mux0 : mux\_4\_1

Port map( x(0) => input,

x(1) => retro(i+1),

x(2) => '0',

x(3) => retro(i+2),

s(0) => s,

s(1) => Y,

y => mux\_ff(i)

);

elsif(i=1 AND N>3) generate

mux1 : mux\_4\_1

Port map( x(0) => retro(i-1),

x(1) => retro(i+1),

x(2) => input,

x(3) => retro(i+2),

s(0) => s,

s(1) => Y,

y => mux\_ff(i)

);

elsif(i=N-2 AND N>3) generate

muxN\_2 : mux\_4\_1

Port map( x(0) => retro(i-1),

x(1) => retro(i+1),

x(2) => retro(i-2),

x(3) => input,

s(0) => s,

s(1) => Y,

y => mux\_ff(i)

);

elsif(i=N-1) generate

muxN\_1 : mux\_4\_1

Port map( x(0) => retro(i-1),

x(1) => input,

x(2) => retro(i-2),-- AND LOAD,

x(3) => '0',

s(0) => s,

s(1) => Y,

y => mux\_ff(i)

);

else generate

mux : mux\_4\_1

Port map( x(0) => retro(i-1),

x(1) => retro(i+1),

x(2) => retro(i-2),

x(3) => retro(i+2),

s(0) => s,

s(1) => Y,

y => mux\_ff(i)

);

end generate border\_mux;

end generate mux;

FlipFlop:for i in N-1 downto 0 generate

FF:flip\_flop\_d

Port Map( D=>mux\_ff(i),

A=>CLK,

RST=>RST,

Q=>retro(i)

);

end generate FlipFlop;

output<=mux\_ff;

end Structural;

#### Simulazione

**[finisci dopo averlo finito nel moltiplicatore]**

## Esercizio 5: Cronometro

### Esercizio 5.1

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

Per realizzare un cronometro, servono dei contatori per ogni tempo richiesto. In particolare, è necessario:

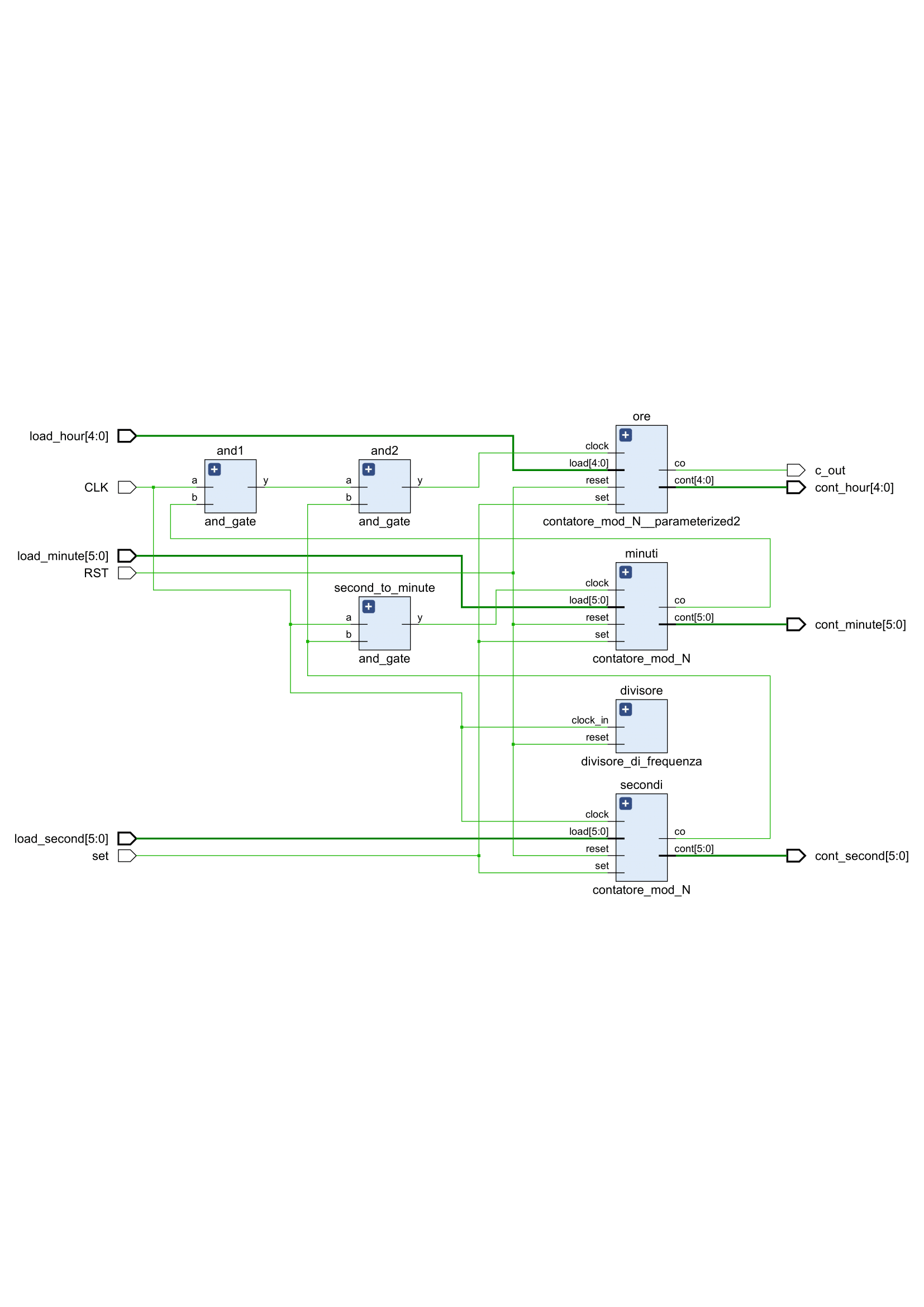
* un **contatore mod 60** per i secondi e i minuti;
* un **contatore mod 24** per le ore.

Per avviare il conteggio sulla base dei tempi richiesta dal singolo contatore, è necessario partire dal clock della board per ottenere un segnale più lento. Si è rivelato fondamentale, dunque, ricorrere a un divisore di frequenza ([vedi appendice](#_Divisore_di_frequenza)) per ottenere un nuovo segnale alla frequenza di 1 Hz. Il segnale in output del divisore di frequenza, dunque, avvierà il conteggio del contatore dei secondi.

Non appena terminerà il conteggio, il contatore dei secondi comunicherà il risultato (con un semplice segnale alto) al contatore dei minuti. Dunque, il contatore dei minuti riceverà come stimolo per avviare il proprio conteggio il termine del conteggio dei secondi. Analogamente, quando i minuti termineranno il proprio conteggio, verrà abilitato il conteggio delle ore.

Il contatore dei minuti avrà come segnale di clock una AND tra il clock della board e il segnale di avvenuto conteggio da parte del contatore dei secondi; il contatore delle ore, analogamente, riceverà una AND tra il clock, l’avvenuto conteggio dei minuti e l’avvenuto conteggio dei secondi.

I contatori prevedono un *load* dei valori parallelo.



#### Implementazione

La rete complessiva è stata realizzata seguendo un approccio **Structural**, mentre i singoli componenti sono realizzate secondo un approccio **Behavioral**.

Il contatore utilizza un signal di appoggio per aggiornare il conteggio. Il fronte del clock su cui il contatore è sensibile è quello di discesa, per evitare problemi di tempificazione; difatti, se si utilizzasse il fronte di salita, tutti i flip-flop commuterebbero nello stesso istante sin dall’avvio del sistema, gestendo erroneamente il conteggio. Per lo stesso motivo, il segnale co di conteggio massimo raggiunto diventa alto un colpo di clock prima che venga raggiunto il valore massimo (ovvero quando il conteggio temp + 1 è pari a ). In tal modo, al successivo ciclo di clock, il conteggio sarà arrivato al valore desiderato, e sul fronte di discesa il valore logico alto di co sarà letto.

entity contatore\_mod\_N is

Generic (N: integer := 60);

Port ( clock : in std\_logic;

reset : in std\_logic;

set : in std\_logic;

load : in std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0);

cont : out std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0);

co : out std\_logic

);

end contatore\_mod\_N;

architecture Behavioral of contatore\_mod\_N is

signal temp: std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0):=(others=>'0');

signal co\_temp: std\_logic:='0';

begin

count:process(clock,reset,set)

begin

co\_temp<='0';

if(reset='1') then

temp<=(others=>'0');

elsif(set='1') then

if(load>=std\_logic\_vector(to\_unsigned(N-1,integer(ceil(log2(real(N)))))) ) then

temp<=std\_logic\_vector(to\_unsigned(N-1,integer(ceil(log2(real(N))))) );

co\_temp<='1';

else

temp<=load;

end if;

elsif(falling\_edge(clock))then

if(std\_logic\_vector(unsigned(temp) + 1) = std\_logic\_vector(to\_unsigned(N-1,integer(ceil(log2(real(N)))))) ) then

temp <= std\_logic\_vector(unsigned(temp) + 1);

co\_temp<='1';

elsif(temp = std\_logic\_vector(to\_unsigned(N-1,integer(ceil(log2(real(N)))))) ) then

temp<=(others=>'0');

--co\_temp<='0';

else

temp <= std\_logic\_vector(unsigned(temp) + 1);

end if;

end if;

end process;

cont<=temp;

co<=co\_temp;

end Behavioral;

Il contatore appena analizzato è stato utilizzato tre volte per realizzare il contatore dei secondi, dei minuti e delle ore.

L’entity complessiva risulta essere:

entity cronometro is

Port ( CLK: in std\_logic;

RST: in std\_logic;

set: in std\_logic;

load\_second:in std\_logic\_vector(5 downto 0);

load\_minute: in std\_logic\_vector(5 downto 0);

load\_hour: in std\_logic\_vector(4 downto 0);

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

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

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

c\_out : out std\_logic

);

end cronometro;

architecture Structural of cronometro is

component divisore\_di\_frequenza is

generic(

CLKIN\_freq : integer := 100000000; -- clock board 100MHz

CLKOUT\_freq : integer := 1 -- frequenza desiderata 1Hz

);

Port (

clock\_in : in STD\_LOGIC;

reset : in STD\_LOGIC;

clock\_out : out STD\_LOGIC

);

end component;

component contatore\_mod\_N is

Generic (N: integer :=60);

Port ( clock : in std\_logic;

reset : in std\_logic;

set : in std\_logic;

load : in std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0);

cont : out std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0);

co : out std\_logic

);

end component;

component and\_gate is

Port ( a: in std\_logic; b: in std\_logic; y: out std\_logic);

end component;

signal clk\_wave: std\_logic;

signal second\_to\_and: std\_logic;

signal minute\_to\_and: std\_logic;

signal and\_to\_and: std\_logic;

signal minute\_en: std\_logic;

signal hour\_en: std\_logic;

begin

divisore: divisore\_di\_frequenza

generic map(

CLKIN\_freq=>100000000, -- clock board 100MHz

CLKOUT\_freq=>1 -- frequenza desiderata 1Hz

)

Port map(

clock\_in=>CLK,

reset=>RST,

clock\_out=>clk\_wave

);

secondi: contatore\_mod\_N

Generic Map(N=>60)

Port Map( clock=>CLK,

reset=>RST,

set=>set,

load=>load\_second,

cont=>cont\_second,

co=>second\_to\_and

);

second\_to\_minute: and\_gate

Port Map( a=>CLK,

b=>second\_to\_and,

y=>minute\_en

);

minuti: contatore\_mod\_N

Generic Map(N=>60)

Port Map( clock=>minute\_en,

reset=>RST,

set=>set,

load=>load\_minute,

cont=>cont\_minute,

co=>minute\_to\_and

);

and1: and\_gate

Port Map( a=>CLK,

b=>minute\_to\_and,

y=>and\_to\_and

);

and2: and\_gate

Port Map( a=>and\_to\_and,

b=>second\_to\_and,

y=>hour\_en

);

ore: contatore\_mod\_N

Generic Map(N=>24)

Port Map( clock=>hour\_en,

reset=>RST,

set=>set,

load=>load\_hour,

cont=>cont\_hour,

co=>c\_out

);

end Structural;

#### Simulazione

Per verificare il corretto funzionamento del cronometro, è stato fornito un segnale di clock analogo a quello della board, e successivamente sono stati testati sia il precaricamento di valori di secondi, minuti e ore, che il reset.

**[togli “precaricamento dei secondi” se li levate da FPGA]**

entity cronometro\_tb is

end cronometro\_tb;

architecture Behavioral of cronometro\_tb is

component cronometro is

Port ( CLK: in std\_logic;

RST: in std\_logic;

set: in std\_logic;

load\_second:in std\_logic\_vector(5 downto 0);

load\_minute: in std\_logic\_vector(5 downto 0);

load\_hour: in std\_logic\_vector(4 downto 0);

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

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

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

c\_out : out std\_logic

);

end component;

signal clock: std\_logic;

signal sec: std\_logic\_vector(5 downto 0):=(others=>'0');

signal min: std\_logic\_vector(5 downto 0):=(others=>'0');

signal hour: std\_logic\_vector(4 downto 0):=(others=>'0');

signal r: std\_logic;

signal sec\_in: std\_logic\_vector(5 downto 0):=(others=>'0');

signal min\_in: std\_logic\_vector(5 downto 0):=(others=>'0');

signal hour\_in: std\_logic\_vector(4 downto 0):=(others=>'0');

signal set\_l : std\_logic;

signal cout: std\_logic;

constant CLK\_period : time := 10 ns;

begin

uut:cronometro

port map( CLK=>clock,

RST=>r,

set=>set\_l,

load\_second=>sec\_in,

load\_minute=>min\_in,

load\_hour=>hour\_in,

cont\_second=>sec,

cont\_minute=>min,

cont\_hour=>hour,

c\_out =>cout

);

CLK\_process :process

begin

clock <= '0';

wait for CLK\_period/2;

clock <= '1';

wait for CLK\_period/2;

end process;

stim\_proc: process

begin

set\_l<='1';

sec\_in<="011101";

min\_in<="111010";

hour\_in<="10111";

wait for 90 ns;

set\_l<='0';

r<='1';

wait for 100 ns;

r<='0';

wait for 200000 ms;

end process;

end Behavioral;

Il risultato ottenuto (visualizzazione dei valori in *Unsigned Decimal*) è il seguente:

Immagine che contiene schermata, linea, software

Descrizione generata automaticamente

Guardando nel dettaglio i segnali, è possibile osservare il corretto azzeramento dei valori dinanzi al segnale di reset alto; non appena lo stesso si abbassa, sul fronte di discesa del clock riparte il conteggio dei secondi.

Immagine che contiene schermata

Descrizione generata automaticamente

### Esercizio 5.2

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

Poiché la board ha a disposizione un display 7-segment a 8 cifre, si è optato per una visualizzazione dell’orario decimale, come un vero orologio.

### Esercizio 5.3 (solo 9 CFU)

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

<se richiesto: descrizione dell’architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell’I/O); file di constraint utilizzato per il progetto>

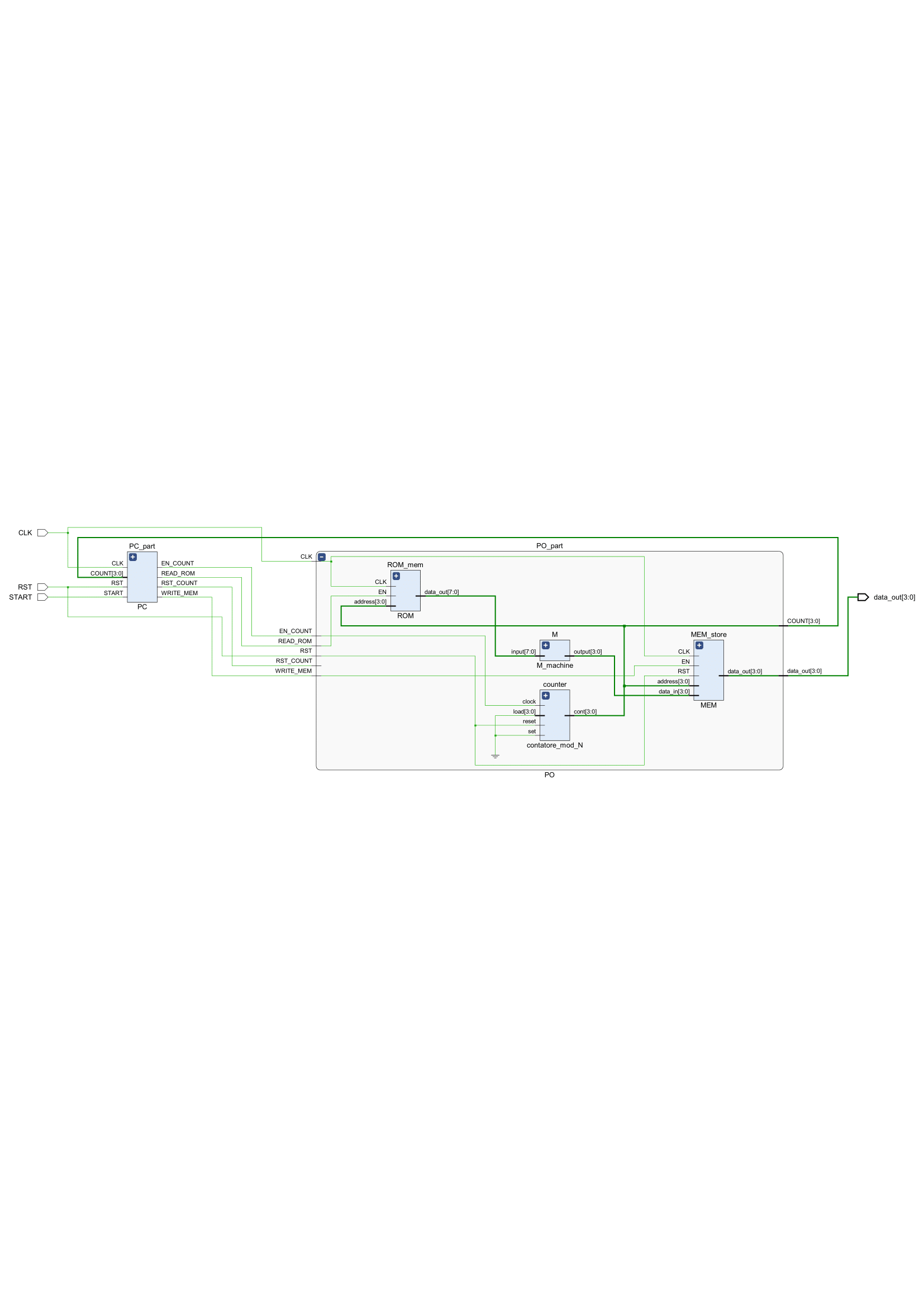
## Esercizio 6: Sistema di lettura-elaborazione-scrittura PO\_PC

### Esercizio 6.1

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



#### Implementazione

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

#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 6.2

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

<se richiesto: descrizione dell’architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell’I/O); file di constraint utilizzato per il progetto>

# **Capitolo 3: Macchine aritmetiche**

## Esercizio 7: Moltiplicatore di Booth

### Esercizio 7.1

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

<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>

#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 7.2

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

<se richiesto: descrizione dell’architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell’I/O); file di constraint utilizzato per il progetto>

## Esercizio 7BIS: Divisore Non-Restoring (solo 9 CFU)

### Esercizio 7BIS.1

Progettare, implementare in VHDL e simulare una macchina divisore (modalità non-restoring) in grado di effettuare la divisione intera fra due stringhe A e B di 4 bit ciascuna.

#### 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>

#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 7BIS.2

Sintetizzare il divisore implementato al punto 7BIS.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

<se richiesto: descrizione dell’architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell’I/O); file di constraint utilizzato per il progetto>

# **Capitolo 4: Comunicazione con handshaking**

## Esercizio 8: 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

<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>

#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

# **Capitolo 5: Processore**

## Esercizio 9: Processore IJVM

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

1. si proceda all’analisi dell’architettura mediante simulazione e si approfondisca lo studio del suo funzionamento per due istruzioni a scelta,
2. si modifichi un codice operativo a scelta, documentando tutte le modifiche effettuate.

#### 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>

#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

# **Capitolo 6: Interfaccia seriale**

## Esercizio 10: Interfaccia UART

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

<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>

#### Simulazione

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

## Esercizio 10BIS: Interfaccia UART (solo 9 CFU)

Dopo aver simulato il comportamento del sistema, lo si implementi su board usando un bottone per il segnale di WR in A, un bottone per il segnale di RD in B, e i display per la visualizzazione del dato correntemente trasmesso e memorizzato in MEM. Si testi l’errore di overrun.

#### Sintesi su board di sviluppo

<se richiesto: descrizione dell’architettura complessiva necessaria per la sintesi su board di sviluppo (nel caso ci siano eventuali componenti aggiuntivi per la gestione dell’I/O); file di constraint utilizzato per il progetto>

# **Capitolo 7: Switch multistadio**

## Esercizio 11: 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

<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>

#### Simulazione (?)

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 11.2 (solo 9 CFU)

Rimuovendo l’ipotesi di lavorare secondo uno schema a priorità fissa fra i nodi e considerando una rete di 8 nodi, lo switch deve gestire eventuali conflitti generati da collisioni con un meccanismo a scelta dello studente.

#### 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>

#### Simulazione (?)

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

### Esercizio 11.3 (solo 9 CFU)

Si implementi un protocollo di handshaking semplice regolato da una coppia di segnali (pronto a inviare/pronto a ricevere) per l’invio di ciascun messaggio fra due nodi.

#### 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>

#### Simulazione (?)

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

# **Capitolo 8: Esercizio prova di esame dicembre 2024**

## Esercizio 12: Prova di esame del 19 dicembre 2024

Un sistema è composto da 2 nodi, A e B. A include una ROM (progettata come macchina sequenziale con READ sincrono) di 8 locazioni da 4 bit, mentre B include un sommatore parallelo in grado di effettuare la somma di 2 stringhe di 4 bit ciascuna e un registro R di 4 bit. Il sistema opera come segue: all’arrivo di un segnale di start, A inizia a prelevare gli elementi ROM[i] dalla propria memoria e li invia, uno alla volta, a B mediante handshaking. B somma progressivamente le stringhe ricevute utilizzando il sommatore e alla fine inserisce il risultato nel registro R.

1. Si disegni l’architettura complessiva del sistema tramite un diagramma a blocchi, identificando parte operativa e parte di controllo di ciascun nodo. Ogni nodo deve essere progettato seguendo un approccio strutturale, individuando tutti i componenti, le loro interfacce e le loro interconnessioni.
2. Si progettino le unità di controllo di A e B evidenziando gli stati, gli ingressi e le uscite negli automi risultanti. È obbligatorio specificare la tempificazione che si intende dare alle macchine (fronte attivo del clock, tempificazione dei segnali di READ/WRITE su registri e memorie).
3. Si progetti il sommatore secondo un’architettura di tipo carry look ahead.
4. Si fornisca l’implementazione in VHDL dell’intero sistema e si proceda alla simulazione nel caso in cui il clock del sistema A e del sistema B siano diversi (A più lento e A più veloce).

#### 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>

#### Simulazione (?)

<descrizione dei testbench di maggiore rilevanza utilizzati per testare il sistema e i suoi componenti e discussione dei principali risultati in simulazione>

# **Appendice**

## Multiplexer 4:1

### Progetto e architettura

Il **Multiplexer 4:1** è un componente puramente combinatorio che convoglia, attraverso opportune linee di selezione, una tra le quattro linee d’ingresso verso un’unica uscita.

In particolare, il multiplexer realizzato è un **multiplexer indirizzabile**, ovvero un componente nel quale le linee di selezione sono opportunamente decodificate da un decoder. Le linee di selezione realizzate in tale architettura, dunque, non sono pari al numero delle linee di ingresso (come avviene nella versione lineare del multiplexer), bensì , ovvero il numero di bit necessari a codificare i numeri che vanno da a .

Per selezionare un’opportuna linea, dunque, è stata realizzata un’architettura di tipo **Dataflow** basata sugli ingressi e sulle selezioni.

### Implementazione

Nel Port sono stati definiti quattro ingressi , due selezioni e un’uscita . L’architecture definisce il flusso dati secondo la funzione logica del multiplexer indirizzabile.

entity mux\_4\_1 is

Port ( a0 : in STD\_LOGIC;

a1 : in STD\_LOGIC;

a2 : in STD\_LOGIC;

a3 : in STD\_LOGIC;

s0 : in STD\_LOGIC;

s1 : in STD\_LOGIC;

y : out STD\_LOGIC

);

end mux\_4\_1;

architecture Dataflow of mux\_4\_1 is

begin

y <= ((a0 AND NOT(s1) AND NOT(s0)) OR (a1 AND NOT(s1) AND s0) OR (a2 AND s1 AND NOT(s0)) OR (a3 AND s1 AND s0));

end Dataflow;

## Button Debouncer

### Progetto e architettura

Questo componente serve per evitare che il segnale logico del bottone venga rilevato in maniera non corretta a causa di possibili oscillazioni e rumori. L’obiettivo è assicurarsi che il clic abbia una durata simile a quella di un plausibile clic umano; i rumori hanno durate nettamente inferiori a un vero clic. Se non inserissimo tale componente, si rischierebbe di valutare un’oscillazione temporanea e indesiderata come valore desiderato per il funzionamento di un componente che presenti un bottone. Inoltre, anche se tenessimo

Il Button Debouncer prende in input il segnale proveniente dal bottone, e genera un segnale "ripulito" della durata di un colpo di clock per segnalare l'avvenuta pressione del bottone.

Per farlo, si implementa un automa a 4 stati:

* **NOT\_PRESSED**: il bottone non è stato cliccato (stato iniziale);
* **CHK\_PRESSED**: possibile clic del bottone (BTN=’1’ rilevato);
* **PRESSED**: clic riconosciuto (se BTN è ancora ’1’ dopo un certo tempo);
* **CHK\_NOT\_PRESSED**: stato intermedio in cui si attende un certo tempo per superare eventuali oscillazioni. Se dopo tale tempo il segnale logico del bottone è ancora alto, si va in PRESSED; altrimenti, si torna in NOT\_PRESSED.

In tal modo, se si mantiene il bottone premuto, non vengono generati più impulsi in uscita.

### Implementazione

entity ButtonDebouncer is

generic (

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

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

);

Port ( RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

BTN : in STD\_LOGIC;

CLEARED\_BTN : out STD\_LOGIC);

end ButtonDebouncer;

architecture Behavioral of ButtonDebouncer is

type stato is (NOT\_PRESSED, CHK\_PRESSED, PRESSED, CHK\_NOT\_PRESSED);

signal BTN\_state : stato := NOT\_PRESSED;

constant max\_count : integer := btn\_noise\_time/CLK\_period; -- 10000000/10= conto 1000000 colpi di clock

begin

deb: process (CLK)

variable count: integer := 0;

begin

if rising\_edge(CLK) then

if( RST = '1') then

BTN\_state <= NOT\_PRESSED;

CLEARED\_BTN <= '0';

else

case BTN\_state is

when NOT\_PRESSED =>

if( BTN = '1' ) then

BTN\_state <= CHK\_PRESSED;

else

BTN\_state <= NOT\_PRESSED;

end if;

when CHK\_PRESSED =>

if(count = max\_count -1) then

if(BTN = '1') then --se arrivo a count max ed è ancora alto vuol dire che non era un bounce, devo alzare CLEARED\_BTN

count:=0;

CLEARED\_BTN <= '1';

BTN\_state <= PRESSED;

else

count:=0;

BTN\_state <= NOT\_PRESSED;

end if;

else

count:= count+1;

BTN\_state <= CHK\_PRESSED;

end if;

when PRESSED =>

CLEARED\_BTN<= '0'; -- questo lo metto per fare in modo che il segnale sia alto per un solo impulso di clock

if(BTN = '0') then

BTN\_state <= CHK\_NOT\_PRESSED;

else

BTN\_state <= PRESSED;

end if;

when CHK\_NOT\_PRESSED =>

if(count = max\_count -1) then

if(BTN = '0') then -- se arrivo a count max ed è ancora basso vuol dire che non era un bounce e il bottone è stato rilasciato

count:=0;

BTN\_state <= NOT\_PRESSED;

else

count:=0;

BTN\_state <= PRESSED;

end if;

else

count:= count+1;

BTN\_state <= CHK\_NOT\_PRESSED;

end if;

when others =>

BTN\_state <= NOT\_PRESSED;

end case;

end if;

end if;

end process;

end Behavioral;

## Divisore di frequenza

### Progetto e architettura

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

### Implementazione

entity divisore\_di\_frequenza is

generic(

CLKIN\_freq : integer := 100000000; -- clock board 100MHz

CLKOUT\_freq : integer := 1000 -- frequenza desiderata 1Hz

);

Port (

clock\_in : in STD\_LOGIC;

reset : in STD\_LOGIC;

clock\_out : out STD\_LOGIC

);

end divisore\_di\_frequenza;

architecture Behavioral of divisore\_di\_frequenza is

signal clockfx : std\_logic := '0';

constant count\_max\_value : integer := CLKIN\_freq/(CLKOUT\_freq)-1;

begin

clock\_out <= clockfx;

count\_for\_division: process(clock\_in)

variable counter : integer range 0 to count\_max\_value := 0;

begin

if rising\_edge(clock\_in) then

if( reset = '1') then

counter := 0;

clockfx <= '0';

else

if counter = count\_max\_value then

clockfx <= '1';

counter := 0;

else

clockfx <= '0';

counter := counter + 1;

end if;

end if;

end if;

end process;

end Behavioral;