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 2](#_Toc1970147757)

[Esercizio 1: Multiplexer 16:1 3](#_Toc2066943553)

[Esercizio 1.1 3](#_Toc1358590550)

[Progetto e architettura 3](#_Toc1634045876)

[Implementazione 3](#_Toc726221675)

[Simulazione 5](#_Toc270496921)

[Esercizio 1.2 6](#_Toc1950965468)

[Progetto e architettura 6](#_Toc49786403)

[Implementazione 7](#_Toc303111792)

[Simulazione 8](#_Toc924042819)

[Esercizio 1.3 9](#_Toc796614847)

[Sintesi su board di sviluppo 9](#_Toc1588496696)

[Esercizio 2: Sistema ROM + M 12](#_Toc1255951489)

[Esercizio 2.1 12](#_Toc1521035852)

[Progetto e architettura 12](#_Toc2135724334)

[Implementazione 12](#_Toc2068793371)

[Simulazione 14](#_Toc1857410487)

[Esercizio 2.2 15](#_Toc887035135)

[Sintesi su board di sviluppo 15](#_Toc871731789)

[Capitolo 2: Reti sequenziali elementari 16](#_Toc1021812482)

[Esercizio 3: Riconoscitore di sequenze 17](#_Toc747288828)

[Esercizio 3.1 17](#_Toc1911128996)

[Progetto e architettura 17](#_Toc503278614)

[Implementazione 17](#_Toc198325707)

[Simulazione 20](#_Toc129636023)

[Esercizio 3.2 21](#_Toc613084457)

[Sintesi su board di sviluppo 21](#_Toc1640150823)

[Esercizio 4: Shift Register 24](#_Toc164380026)

[Esercizio 4.1 24](#_Toc1662313975)

[Progetto e architettura 24](#_Toc481660439)

[Approccio Behavioral 24](#_Toc348451362)

[Approccio Structural 24](#_Toc2134069514)

[Implementazione 25](#_Toc543184072)

[Approccio Behavioral 26](#_Toc294696337)

[Approccio Structural 26](#_Toc898108354)

[Simulazione 28](#_Toc712650553)

[Approccio Behavioral 29](#_Toc1087252553)

[Approccio Structural 30](#_Toc382689709)

[Timing Analysis 32](#_Toc1976975460)

[Setup Timing 33](#_Toc1206048949)

[Hold Timing 33](#_Toc458658669)

[Pulse Width Timing 33](#_Toc171924827)

[Analisi finale 33](#_Toc340649341)

[Esercizio 5: Cronometro 33](#_Toc2088901106)

[Esercizio 5.1 33](#_Toc1036872710)

[Progetto e architettura 33](#_Toc1436859805)

[Implementazione 34](#_Toc1384149031)

[Simulazione 36](#_Toc1323411763)

[Esercizio 5.2 38](#_Toc747856937)

[Sintesi su board di sviluppo 38](#_Toc541529818)

[Esercizio 5.3 (solo 9 CFU) 44](#_Toc2127772441)

[Progetto e architettura 44](#_Toc1181979740)

[Implementazione 44](#_Toc658559951)

[Esercizio 6: Sistema di lettura-elaborazione-scrittura PO\_PC 49](#_Toc1712928584)

[Esercizio 6.1 49](#_Toc1394319006)

[Progetto e architettura 49](#_Toc287857108)

[Implementazione 49](#_Toc377980794)

[Simulazione 54](#_Toc930094480)

[Esercizio 6.2 56](#_Toc875117490)

[Sintesi su board di sviluppo 56](#_Toc946628956)

[Capitolo 3: Macchine aritmetiche 58](#_Toc1346591232)

[Esercizio 7: Moltiplicatore di Booth 58](#_Toc2013870900)

[Esercizio 7.1 58](#_Toc1349165584)

[Progetto e architettura 58](#_Toc1023660332)

[Implementazione 60](#_Toc1875226549)

[Simulazione 68](#_Toc1834235966)

[Esercizio 7.2 69](#_Toc1003650015)

[Sintesi su board di sviluppo 69](#_Toc1423151625)

[Esercizio 7BIS: Divisore Non-Restoring (solo 9 CFU) 71](#_Toc972460034)

[Esercizio 7BIS.1 71](#_Toc1609486019)

[Progetto e architettura 72](#_Toc46695081)

[Implementazione 72](#_Toc212631991)

[Simulazione 72](#_Toc105276186)

[Esercizio 7BIS.2 72](#_Toc2008996895)

[Sintesi su board di sviluppo 72](#_Toc150197108)

[Capitolo 4: Comunicazione con handshaking 72](#_Toc1806419498)

[Esercizio 8: Comunicazione con handshaking 73](#_Toc940404493)

[Esercizio 8.1 73](#_Toc1506464481)

[Progetto e architettura 73](#_Toc1911903020)

[Sistema A 73](#_Toc1965342756)

[Sistema B 74](#_Toc903963974)

[Sistema complessivo 75](#_Toc94989415)

[Implementazione 75](#_Toc549109924)

[Sistema A 75](#_Toc561418959)

[Sistema B 77](#_Toc297790645)

[Sistema complessivo 79](#_Toc1117443340)

[Simulazione 80](#_Toc225953188)

[Capitolo 5: Processore 81](#_Toc666817526)

[Esercizio 9: Processore IJVM 82](#_Toc743573547)

[MIC-1 82](#_Toc1231676668)

[Datapath 82](#_Toc615380163)

[Control Unit 83](#_Toc1477226645)

[Analisi delle istruzioni 84](#_Toc1150723489)

[IADD 84](#_Toc1296408101)

[BIPUSH 85](#_Toc768272188)

[Nuova istruzione 86](#_Toc1571634607)

[THREEAND 86](#_Toc574030805)

[Capitolo 6: Interfaccia seriale 86](#_Toc114243248)

[Esercizio 10: Interfaccia UART 87](#_Toc797032965)

[Progetto e architettura 87](#_Toc1109268080)

[Sistema A 88](#_Toc1293007923)

[Sistema B 88](#_Toc1314825057)

[Sistema complessivo 89](#_Toc2081731509)

[Implementazione 90](#_Toc678160596)

[Sistema A 90](#_Toc1881600119)

[Sistema B 93](#_Toc152741959)

[Sistema complessivo 95](#_Toc702574819)

[Simulazione 96](#_Toc697925482)

[Esercizio 10BIS: Interfaccia UART (solo 9 CFU) 97](#_Toc1679100161)

[Sintesi su board di sviluppo 98](#_Toc1991486910)

[Capitolo 7: Switch multistadio 98](#_Toc454875642)

[Esercizio 11: Switch multistadio 99](#_Toc1018364679)

[Esercizio 11.1 99](#_Toc626324730)

[Progetto e architettura 99](#_Toc1745499792)

[Implementazione 100](#_Toc1352616927)

[Simulazione 105](#_Toc1073921431)

[Esercizio 11.2 (solo 9 CFU) 107](#_Toc1776554675)

[Progetto e architettura 107](#_Toc233836868)

[Implementazione 108](#_Toc217882078)

[Simulazione 113](#_Toc984852565)

[Esercizio 11.3 (solo 9 CFU) 115](#_Toc688510463)

[Progetto e architettura 115](#_Toc492009325)

[Implementazione 115](#_Toc29583798)

[Simulazione 117](#_Toc845113099)

[Capitolo 8: Esercizio prova di esame dicembre 2024 119](#_Toc727311702)

[Esercizio 12: Prova di esame del 19 dicembre 2024 120](#_Toc1934405759)

[Progetto e architettura 120](#_Toc1981726549)

[Sistema A 120](#_Toc474748332)

[Sistema B 121](#_Toc499777585)

[Sistema complessivo 122](#_Toc404304690)

[Implementazione 122](#_Toc688222481)

[Sistema A 122](#_Toc1043827613)

[Sistema B 125](#_Toc1226489021)

[Sistema complessivo 130](#_Toc2139139132)

[Simulazione 131](#_Toc1991742392)

[Appendice 132](#_Toc1338510627)

[Multiplexer 4:1 133](#_Toc310692545)

[Progetto e architettura 133](#_Toc1572881116)

[Implementazione 133](#_Toc1726309537)

[Button Debouncer 133](#_Toc476842549)

[Progetto e architettura 133](#_Toc1723838418)

[Implementazione 134](#_Toc2062533937)

[Divisore di frequenza 135](#_Toc1088173726)

[Progetto e architettura 135](#_Toc104693201)

[Implementazione 135](#_Toc1056749814)

[Contatore modulo N 136](#_Toc821880318)

[Progetto e architettura 136](#_Toc1042051522)

[Implementazione 136](#_Toc919868626)

# 

# **Capitolo 1: Reti combinatorie elementari**

## Esercizio 1: Multiplexer 16:1

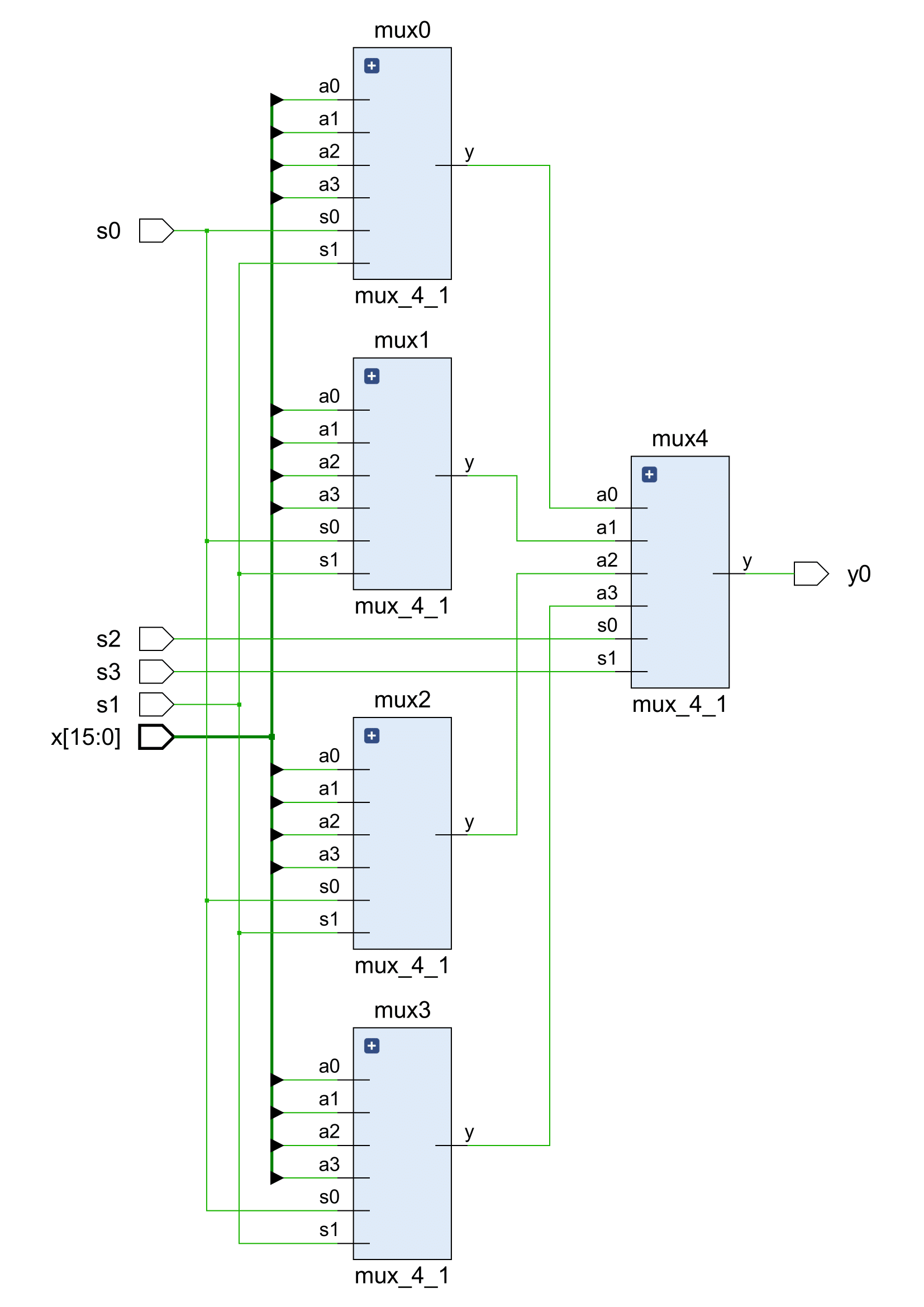
### Esercizio 1.1

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**.

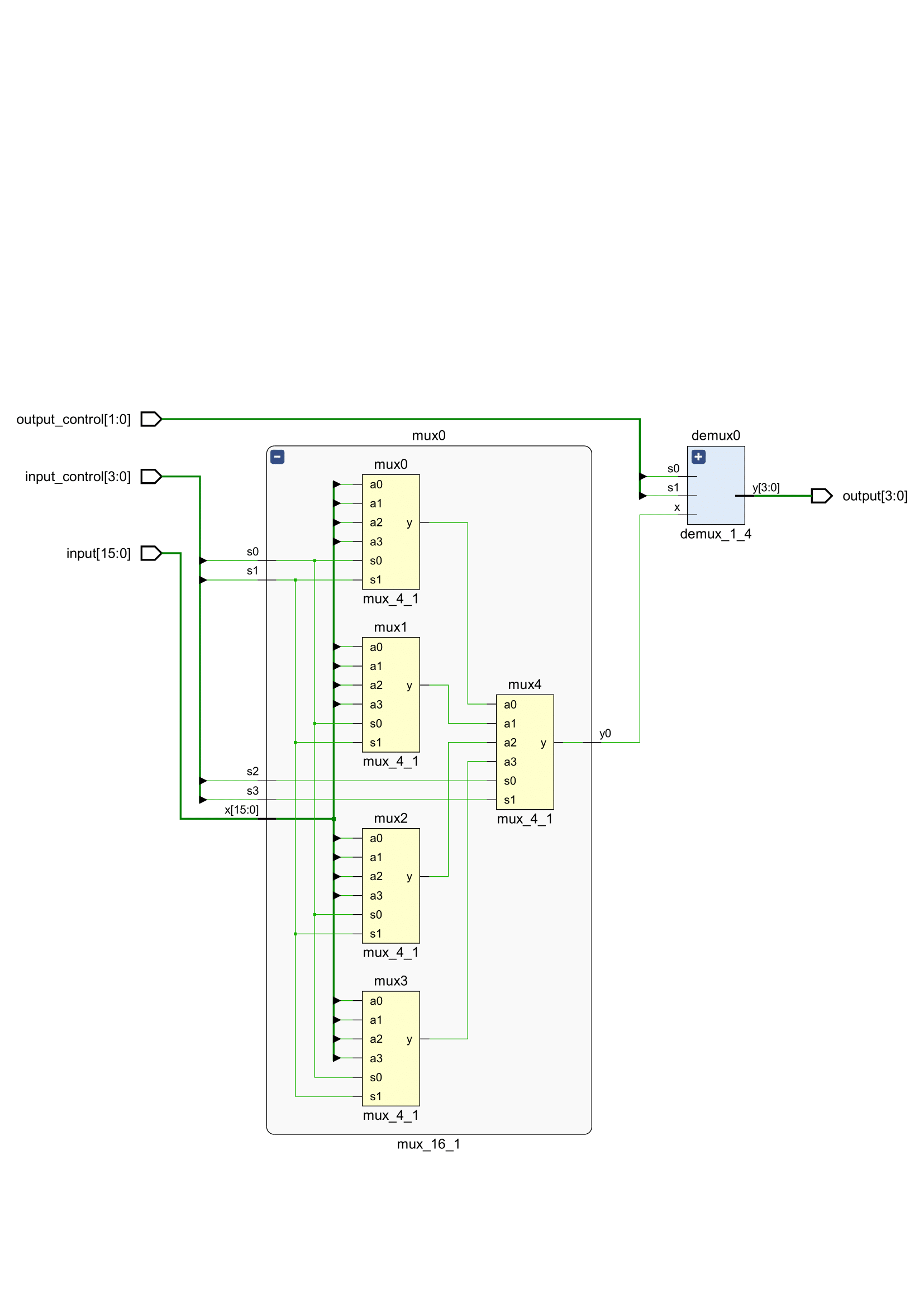
#### Progetto e architettura

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

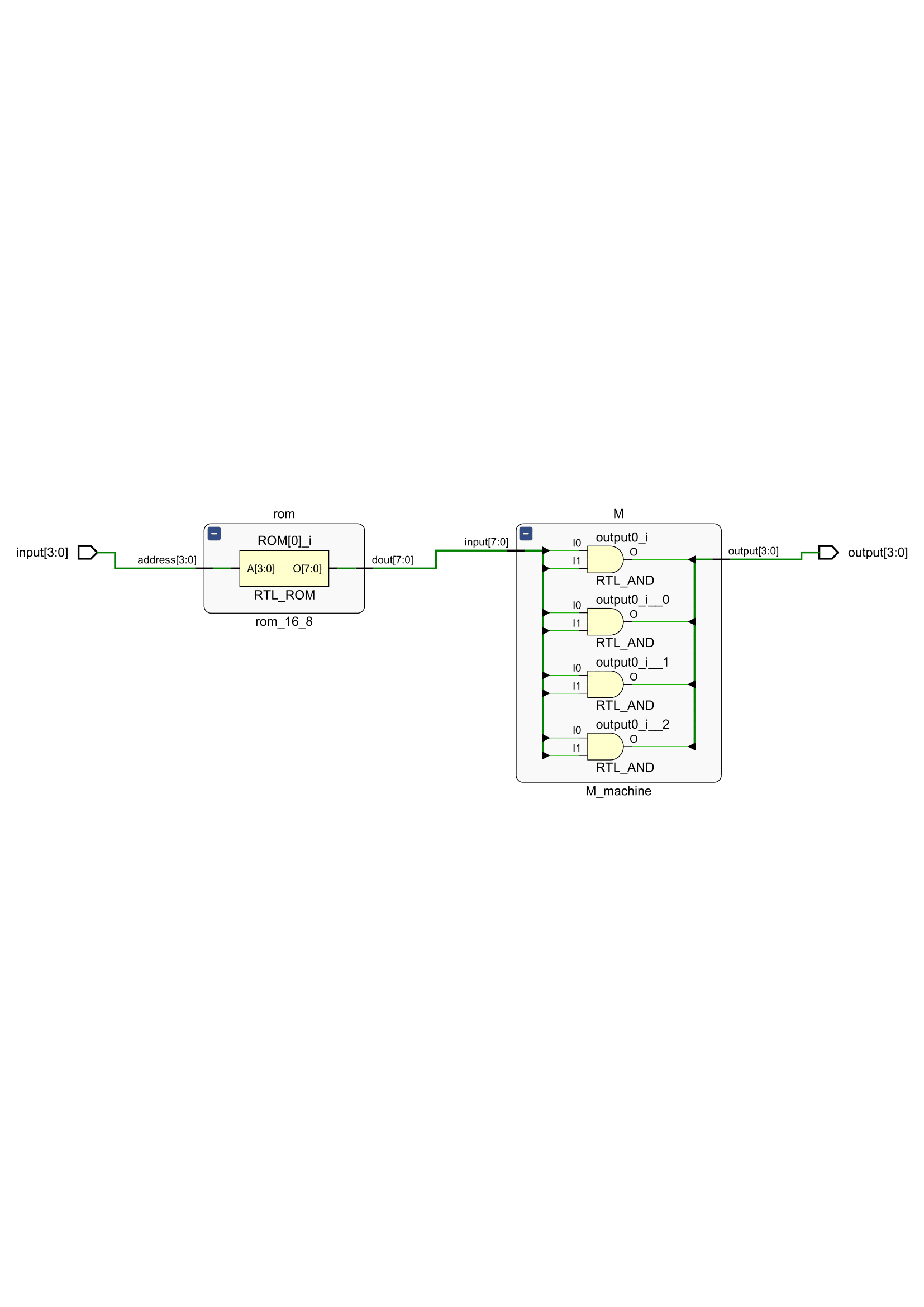
Progettare, implementare in VHDL e testare mediante simulazione un **sistema S** composto da una **ROM** puramente combinatoria di 16 locazioni da 8 bit ciascuna e da una macchina combinatoria **M** che opera come segue: fornito al sistema un indirizzo A di 4 bit, il sistema restituisce il valore contenuto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

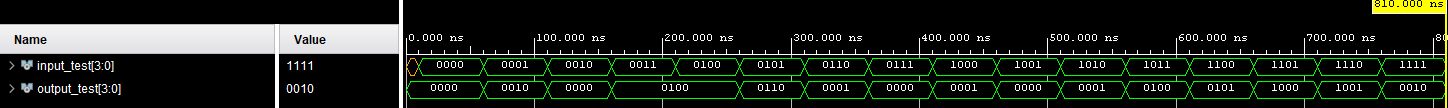
severity failure;

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]

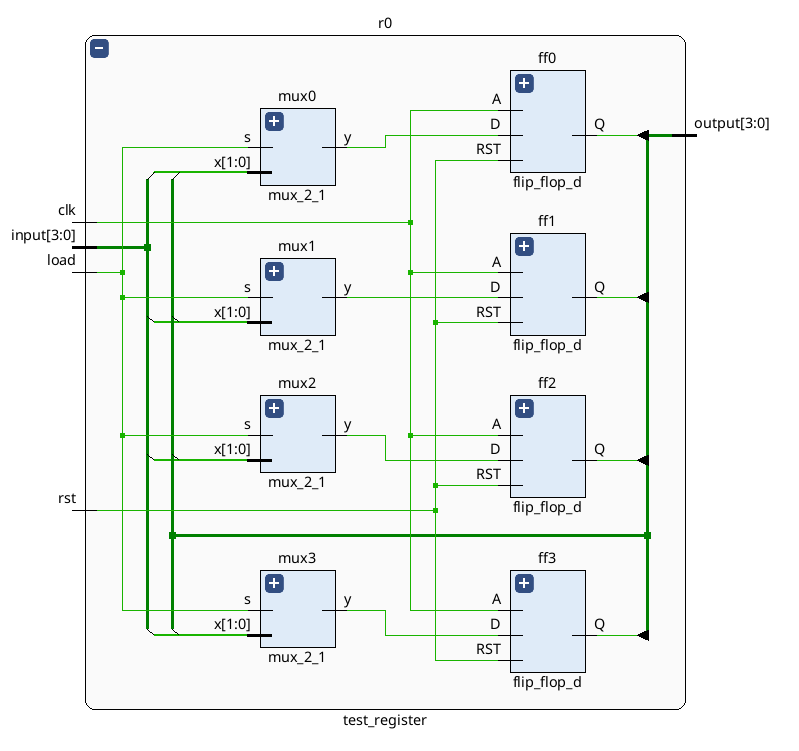
set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { LED[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

#### Timing Analysis

È stata effettuata un’analisi temporale sulla versione strutturale dello shift register. Eseguire una **Timing Analysis** su un dispositivo significa osservare i tempi di propagazione dei segnali nel circuito. Il timing deve rispettare specifici vincoli temporali imposti dal clock, per cui l’analisi richiede la presenza (o l’aggiunta, se non previsti) di elementi **clockati** (flip-flop e registri). Grazie alla tempificazione data dal clock, è possibile osservare il percorso dei dati e dunque la sincronizzazione complessiva del sistema.

Quindi è stato sviluppato un registro dotato di 4 flip flop e mux per inizializzare i dati presenti nei flip flop.

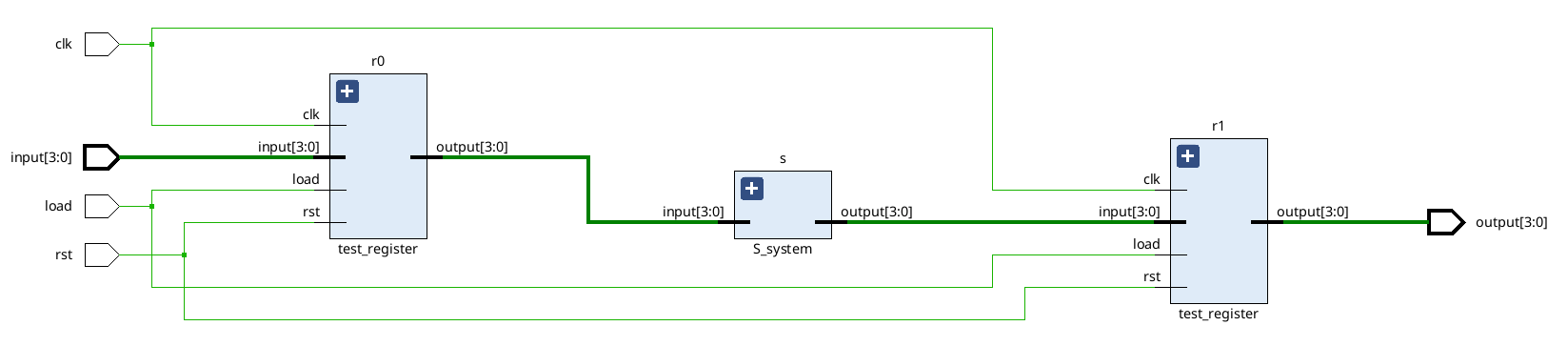
Schematics:



Codice:

Quindi sono stati utilizzati due registri, uno in input ed uno in output per il sistema ROM + M.

Schematics:



Codice:

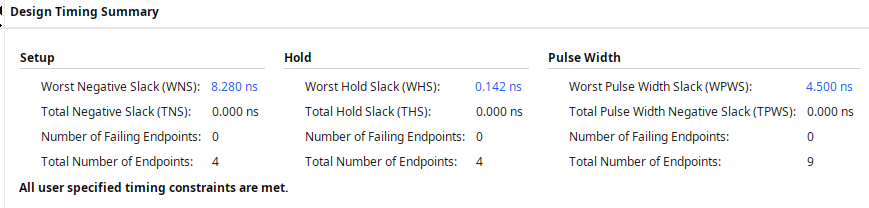
Per generare un clock, si è reso necessario aggiungere un file di *constraint* con le seguenti istruzioni:

## 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}];

L’analisi ha riportato i seguenti risultati:



Il report temporale generato evidenzia che tutti i vincoli temporali specificati sono stati rispettati. In particolare, i parametri relativi ai tempi di ***setup***, ***hold*** e ***pulse width*** risultano ampiamente soddisfatti (come, tra l’altro, confermato anche dalla dicitura "*All user specified timing constraints are met*").

##### Setup Timing

Per quanto riguarda il Setup Timing, il Worst Negative Slack (WNS) è pari a 8,280 ns, con un Total Negative Slack (TNS) di 0,000 ns e nessun endpoint fallito su 4.

I valori appena visti sono una chiara indicazione del fatto che il design abbia un margine temporale significativo rispetto ai vincoli imposti dal clock.

La presenza di uno slack positivo, specialmente di tale entità, suggerisce che il circuito potrebbe operare anche a frequenze superiori rispetto a quella attualmente specificata.

##### Hold Timing

Nel caso dell'Hold Timing, il Worst Hold Slack (WHS) risulta essere di 0,142 ns, anch'esso positivo, con un Total Hold Slack (THS) di 0,000 ns.

Anche in questo caso non si rilevano endpoint falliti, il che conferma la corretta gestione dei ritardi di hold all'interno del progetto.

##### Pulse Width Timing

Infine, l'analisi della Pulse Width mostra un Worst Pulse Width Slack (WPWS) pari a 4,500 ns, con nessuna violazione registrata sui 16 endpoint analizzati.

Questo significa che la larghezza minima degli impulsi del clock e di eventuali segnali asincroni è stata rispettata in tutte le condizioni simulate.

##### Analisi finale

Le tre categorie hanno riportato ottimi risultati, il che lascia intuire che il sistema funzioni piuttosto bene e non vi sono particolari problemi di tempificazione. Potenzialmente, il risultato del Setup Timing potrebbe anche suggerire di lavorare a frequenze superiori senza riscontrare ritardi di propagazione significativi.

# **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.

In ogni stato, viene dapprima riconosciuto il modo , in modo da poter *switchare* tra le due modalità a partire da ogni stato.

**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 in una finestra temporale pari al periodo completo del clock della board. È stato poi modificato il valore dell’input del riconoscitore, in modo da osservare il comportamento dell’uscita. 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

Per mappare il riconoscitore sul dispositivo, è bastato utilizzare dei Button Debouncer per assegnare input e modo senza disturbi e oscillazioni. Un singolo LED mostra l’uscita del riconoscitore (alta/bassa).

Per catturare gli switch, è stato utilizzato un componente switch\_capture che opera come segue:

entity switch\_capture is

Port ( clock : in std\_logic;

button : in std\_logic;

input : in std\_logic;

output : out std\_logic

);

end switch\_capture;

architecture Behavioral of switch\_capture is

begin

capture: process(clock, button, input)

begin

if(rising\_edge(clock)) then

if(button='1') then

output<=input;

end if;

end if;

end process;

end Behavioral;

Il componente definitivo risulta essere il seguente:

entity riconoscitore\_101\_fpga is

Port ( SW\_in : in std\_logic ;

SW\_mode : in std\_logic;

BTN\_in : in std\_logic;

BTN\_mode : in std\_logic;

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

RST : in std\_logic;

LED\_out : out std\_logic

);

end riconoscitore\_101\_fpga;

architecture Structural of riconoscitore\_101\_fpga is

signal input\_signal : std\_logic ;

signal mode\_signal : std\_logic ;

signal switch\_input\_signal:std\_logic;

signal switch\_mode\_signal:std\_logic;

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;

component switch\_capture is

Port ( clock : in std\_logic;

button : in std\_logic;

input : in std\_logic;

output : out std\_logic

);

end component;

component ButtonDebouncer is

Port ( RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

BTN : in STD\_LOGIC;

CLEARED\_BTN : out STD\_LOGIC);

end component;

begin

button\_input : ButtonDebouncer

Port map(RST => RST,

CLK => CLK,

BTN => BTN\_in,

CLEARED\_BTN => input\_signal

);

button\_mode : ButtonDebouncer

Port map(RST => RST,

CLK => CLK,

BTN => BTN\_mode,

CLEARED\_BTN => mode\_signal

);

switch\_input: switch\_capture

Port map( clock=>CLK,

button=>input\_signal, -- BTNL

input=>SW\_in, -- primo switch

output=>switch\_input\_signal

);

switch\_mode: switch\_capture

Port map( clock=>CLK,

button=>mode\_signal, -- BTNR

input=>SW\_mode, -- secondo switch

output=>switch\_mode\_signal

);

riconoscitore: riconoscitore\_101

port map(

input=> switch\_input\_signal,

a=>input\_signal ,

reset=>RST,

m=>switch\_mode\_signal,

output=>LED\_out

);

end Structural;

I *constraints* del progetto risultano dunque essere:

## 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\_in }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

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

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

## LEDs

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

## Buttons

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

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

## 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

##### 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.

Al momento dello shift, il flip-flop deve 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.

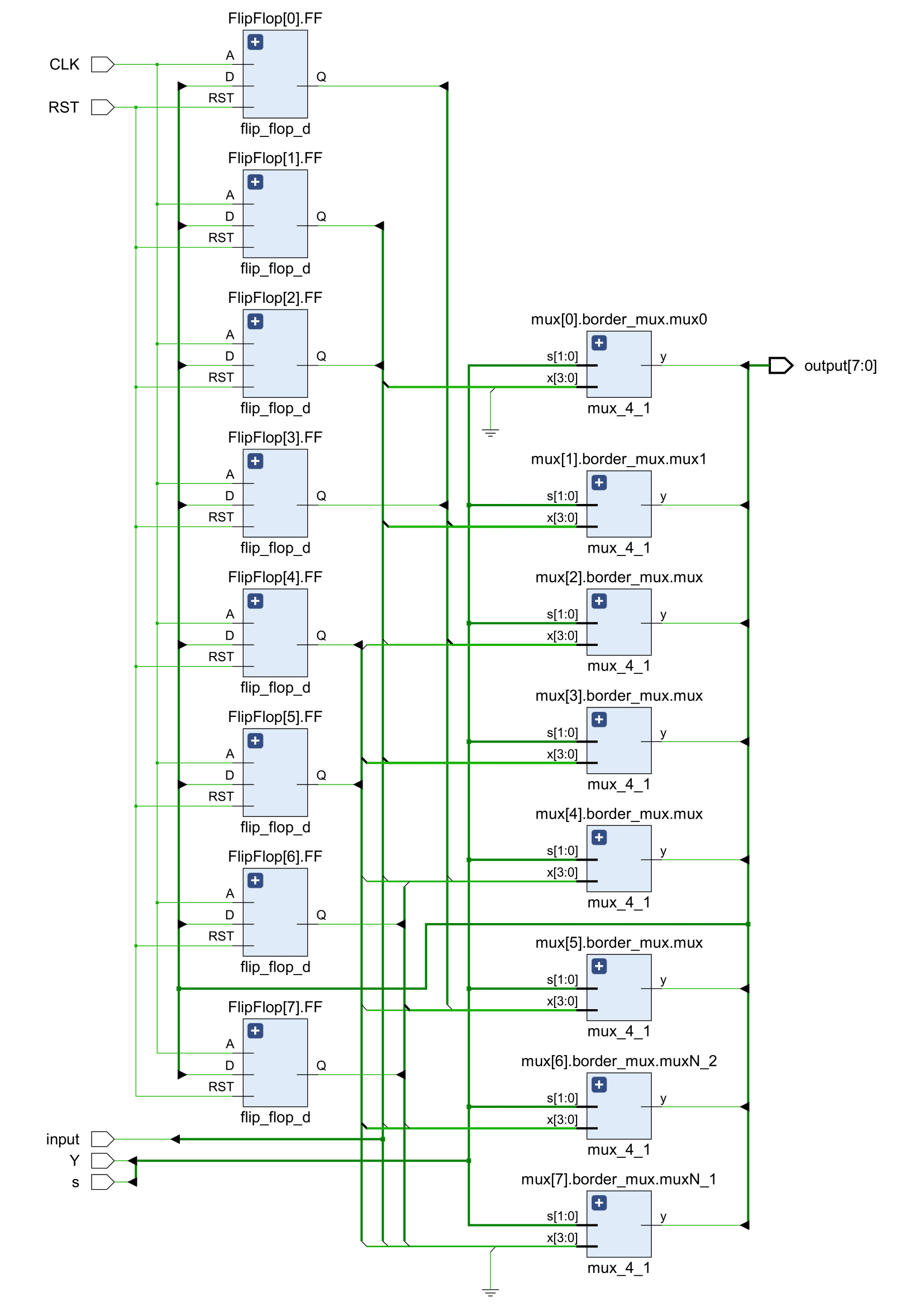
Nel caso di shift a due posizioni, il secondo valore da shiftare sarà un bit ‘0’. In tal caso:

* 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.

Anche il secondo e il penultimo flip-flop, nel caso di shift register ad almeno 3 locazioni, vanno trattati in maniera lievemente diversa rispetto ai generici flip-flop al fine di realizzare lo shift descritto sopra.

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 la 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;

load\_data: in std\_logic\_vector(N-1 downto 0);

EN : 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

output\_temp<= load\_data;

elsif(EN = '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 in precedenza.

entity shift\_register is

Generic(N : integer := 8);

Port ( CLK : in std\_logic; -- Abilitazione

RST : in std\_logic;

input : in std\_logic;

LOAD : in std\_logic;

load\_data: in std\_logic\_vector(N-1 downto 0);

EN : in std\_logic;

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

Y : in std\_logic:='0'; -- 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);

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

signal mux\_4\_to\_mux\_2:std\_logic\_vector (N-1 downto 0);

signal clock: std\_logic;

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

clock<=CLK AND EN;

mux\_load:for i in N-1 downto 0 generate

mux:mux\_2\_1

Port Map( x(0)=>mux\_4\_to\_mux\_2(i),

x(1)=>load\_data(i),

s=>LOAD,

y=>mux\_ff(i)

);

end generate;

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\_4\_to\_mux\_2(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\_4\_to\_mux\_2(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\_4\_to\_mux\_2(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),

x(3) => '0',

s(0) => s,

s(1) => Y,

y => mux\_4\_to\_mux\_2(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\_4\_to\_mux\_2(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=>clock,

RST=>RST,

Q=>retro(i)

);

end generate FlipFlop;

output<=retro;

end Structural;

#### Simulazione

##### Approccio Behavioral

Il codice per la simulazione comportamentale è il seguente:

entity shift\_register\_behavioral\_tb is

Generic(N : integer := 4);

end shift\_register\_behavioral\_tb;

architecture Behavioral of shift\_register\_behavioral\_tb is

component shift\_register\_behavioral is

Generic(N : integer := 4);

Port ( input : in std\_logic;

RST : in std\_logic;

LOAD : in std\_logic;

load\_data: in std\_logic\_vector(N-1 downto 0);

EN : 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 component;

signal input : std\_logic;

signal load : std\_logic;

signal clock: std\_logic;

signal reset: std\_logic;

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

signal enable: std\_logic;

signal s: std\_logic;

signal y: std\_logic;

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

constant period: time := 10 ns;

begin

uut: shift\_register\_behavioral

Port Map( input=>input,

RST=>reset,

LOAD=>load,

load\_data=>data,

en=>enable,

CLK=>clock,

s=>s,

Y=>y,

output=>output

);

clock\_proc: process

begin

clock<='0';

wait for period/2;

clock<='1';

wait for period/2;

end process;

--1001

stim\_proc: process

begin

enable<='1';

data<="1010";

wait for 10 ns;

s<='0'; -- Shift a destra

y<='0'; -- 1 bit

wait for 5 ns;

load<='1';

input <= '1';

wait for 5 ns;

load<='0';

wait for 10 ns;

s<='1'; -- Shift a sinistra

y<='0'; -- 1 bit

wait for 5 ns;

load<='1';

input <= '1';

wait for 5 ns;

load<='0';

wait for 10 ns;

s<='0'; -- Shift a destra

y<='1'; -- 2 bit

wait for 5 ns;

load<='1';

input <= '1';

wait for 5 ns;

load<='0';

wait for 10 ns;

s<='1'; -- Shift a sinistra

y<='1'; -- 2 bit

wait for 5 ns;

enable<='0';

load<='1';

input <= '1';

wait for 5 ns;

load<='0';

wait;

end process;

end Behavioral;

L’output prodotto è il seguente:

Immagine che contiene schermata, Software multimediale, Software per la grafica, software

Descrizione generata automaticamente

Si osservano sia il corretto funzionamento dello shift che l’interruzione del funzionamento a fronte di un segnale basso di enable.

##### Approccio Structural

Il codice per la simulazione strutturale è il seguente:

entity shift\_register\_tb is

Generic(N : integer := 4);

end shift\_register\_tb;

architecture Behavioral of shift\_register\_tb is

component shift\_register is

Generic(N : integer := 4);

Port ( CLK : in std\_logic; -- Abilitazione

RST : in std\_logic;

input : in std\_logic;

LOAD : in std\_logic;

load\_data: in std\_logic\_vector(N-1 downto 0);

EN : in std\_logic;

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 component;

signal input : std\_logic;

signal load : std\_logic;

signal clock: std\_logic;

signal reset: std\_logic;

signal s: std\_logic;

signal y: std\_logic;

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

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

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

constant period: time := 10 ns;

begin

uut: shift\_register

Port Map( input=>input,

RST=>reset,

LOAD=>load,

load\_data=>data,

CLK=>clock,

EN =>enable,

s=>s,

Y=>y,

output=>output

);

clock\_proc: process

begin

clock<='0';

wait for period/2;

clock<='1';

wait for period/2;

end process;

--1001

stim\_proc: process

begin

wait for 10 ns;

reset<='0';

enable<='1';

load<='1';

data<="1111";

wait for 10 ns;

enable<='0';

reset<='0';

load<='0';

s<='1';

y<='1';

input <= '0';

wait for 10 ns;

wait for 5 ns;

enable<='1';

input <= '0';

wait for 5 ns;

enable<='1';

wait for 5 ns;

enable<='1';

input <= '0';

wait for 5 ns;

enable<='1';

wait for 5 ns;

enable<='1';

input <= '0';

wait for 5 ns;

enable<='1';

wait for 5 ns;

enable<='1';

input <= '0';

wait for 5 ns;

enable<='1';

wait;

end process;

end Behavioral;

L’output ottenuto risulta essere:

Immagine che contiene schermata, Software multimediale

Descrizione generata automaticamente

Lo shift register inserisce correttamente i valori iniziali ed effettua gli shift come previsto.

#### Timing Analysis

È stata effettuata un’analisi temporale sulla versione strutturale dello shift register. Eseguire una **Timing Analysis** su un dispositivo significa osservare i tempi di propagazione dei segnali nel circuito. Il timing deve rispettare specifici vincoli temporali imposti dal clock, per cui l’analisi richiede la presenza (o l’aggiunta, se non previsti) di elementi **clockati** (flip-flop e registri). Grazie alla tempificazione data dal clock, è possibile osservare il percorso dei dati e dunque la sincronizzazione complessiva del sistema.

Per generare un clock, si è reso necessario aggiungere un file di *constraint* con le seguenti istruzioni:

## 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}];

L’analisi ha riportato i seguenti risultati:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteIl report temporale generato evidenzia che tutti i vincoli temporali specificati sono stati rispettati. In particolare, i parametri relativi ai tempi di ***setup***, ***hold*** e ***pulse width*** risultano ampiamente soddisfatti (come, tra l’altro, confermato anche dalla dicitura "*All user specified timing constraints are met*").

##### Setup Timing

Per quanto riguarda il Setup Timing, il Worst Negative Slack (WNS) è pari a 6,340 ns, con un Total Negative Slack (TNS) di 0,000 ns e nessun endpoint fallito su 16.

I valori appena visti sono una chiara indicazione del fatto che il design abbia un margine temporale significativo rispetto ai vincoli imposti dal clock.

La presenza di uno slack positivo, specialmente di tale entità, suggerisce che il circuito potrebbe operare anche a frequenze superiori rispetto a quella attualmente specificata.

##### Hold Timing

Nel caso dell'Hold Timing, il Worst Hold Slack (WHS) risulta essere di 0,409 ns, anch'esso positivo, con un Total Hold Slack (THS) di 0,000 ns.

Anche in questo caso non si rilevano endpoint falliti, il che conferma la corretta gestione dei ritardi di hold all'interno del progetto.

##### Pulse Width Timing

Infine, l'analisi della Pulse Width mostra un Worst Pulse Width Slack (WPWS) pari a 4,500 ns, con nessuna violazione registrata sui 16 endpoint analizzati.

Questo significa che la larghezza minima degli impulsi del clock e di eventuali segnali asincroni è stata rispettata in tutte le condizioni simulate.

##### Analisi finale

Le tre categorie hanno riportato ottimi risultati, il che lascia intuire che il sistema funzioni piuttosto bene e non vi sono particolari problemi di tempificazione. Potenzialmente, il risultato del Setup Timing potrebbe anche suggerire di lavorare a frequenze superiori senza riscontrare ritardi di propagazione significativi.

## 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:

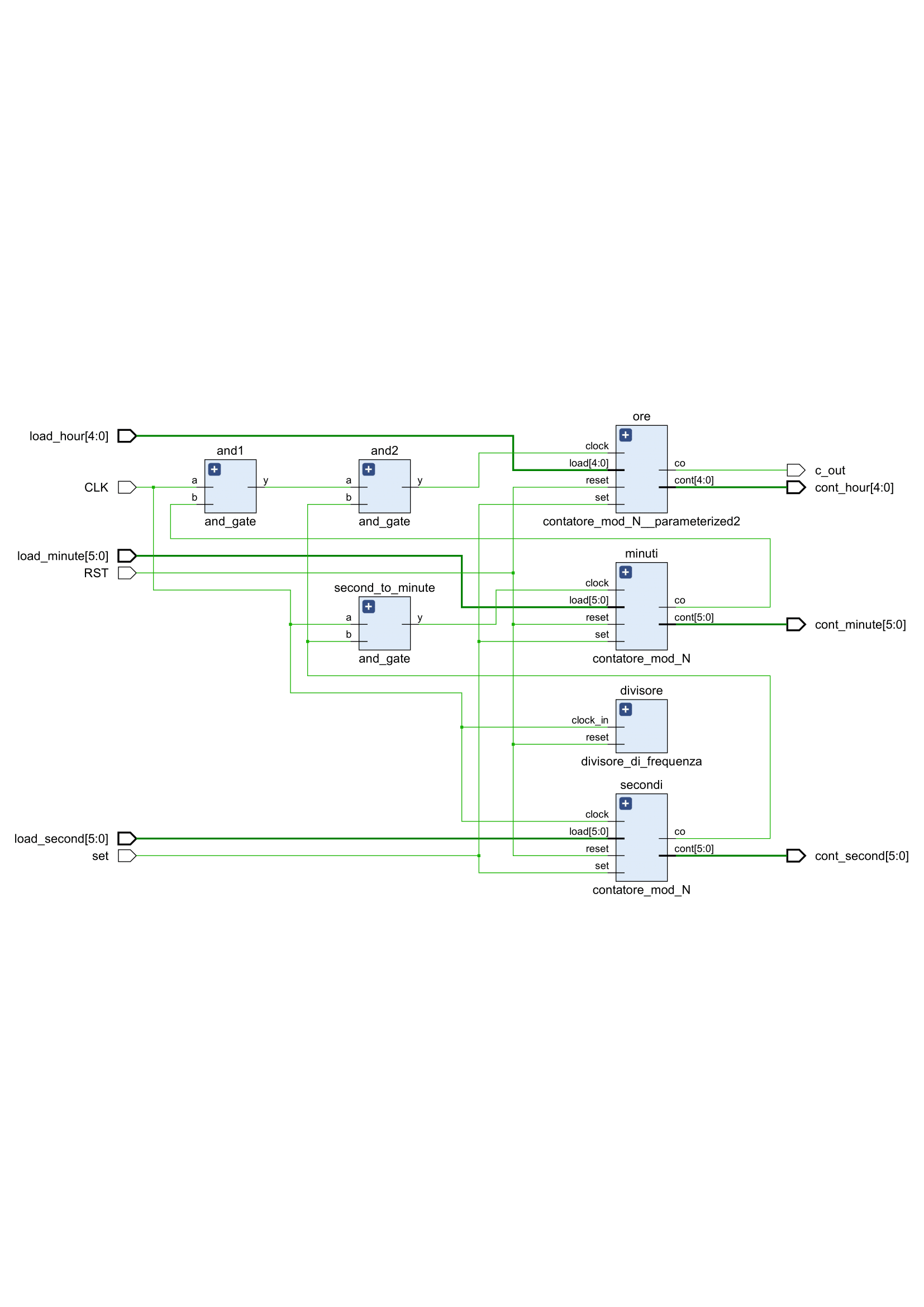
* un **contatore mod 60** per i secondi e i minuti;
* un **contatore mod 24** per le ore.

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, lo stimolo per avviare il conteggio dei minuti è dato proprio dal 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* parallelo dei valori.



#### Implementazione

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

Il contatore utilizzato, [analizzato nell’appendice](#_Contatore_modulo_N), è stato utilizzato tre volte per realizzare il contatore dei secondi, dei minuti e delle ore. È stato inoltre utilizzato il [divisore di frequenza](#_Divisore_di_frequenza), come detto in precedenza.

Il valore binario di conteggio avrà lunghezza .

Per realizzare le AND che abilitano i contatori, è stato realizzato un semplice componente **Dataflow** and\_gate:

entity and\_gate is

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

end and\_gate;

architecture Dataflow of and\_gate is

begin

y<=a AND b;

end Dataflow;

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.

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.

La gestione del display a 7 segmenti ha previsto l’implementazione di varie componenti:

* counter\_mod8
* cathodes\_manager
* anodes\_manager
* cathodes\_input\_manager
* divisore\_di\_frequenza
* display\_seven\_segments

Il display funziona seguendo uno schema circuitale basato su **catodi** e **anodi**: un anodo crea una differenza di potenziale, e la corrente inizia a scorrere se un catodo è attivo. L’anodo è unico e dunque può alimentare una sola cifra, ed è 1-attivo; invece, i catodi sono presenti in ogni segmento e sono 0-attivi.

Non è possibile accendere realmente tutte le cifre, ma se ne può accendere una alla volta a una frequenza che l’occhio umano non è in grado di distinguere, così da dare l’impressione che tutte le cifre siano accese insieme. Qui entra in gioco il divisore di frequenza.

Il cathodes\_manager gestisce la corretta accensione dei segmenti di una singola cifra. Essendo la visualizzazione decimale, è stata esclusa l’implementazione di costanti per rappresentare le lettere da A a F:

entity cathodes\_manager is

Port ( value : in integer; --dato da mostrare sulla cifra del display

dot : in STD\_LOGIC; --configurazione del punto: acceso o spento

cathodes\_dot : out STD\_LOGIC\_VECTOR (7 downto 0)); --indica i 7 catodi più il punto

end cathodes\_manager;

architecture Behavioral of cathodes\_manager is

-- i catodi sono collegati nel seguente ordine:

-- cathodes[0]=CA

-- cathodes[2]=CB

--....

-- i catodi sono 0-attivi

constant zero : std\_logic\_vector(6 downto 0) := "1000000";

constant one : std\_logic\_vector(6 downto 0) := "1111001";

constant two : std\_logic\_vector(6 downto 0) := "0100100";

constant three : std\_logic\_vector(6 downto 0) := "0110000";

constant four : std\_logic\_vector(6 downto 0) := "0011001";

constant five : std\_logic\_vector(6 downto 0) := "0010010";

constant six : std\_logic\_vector(6 downto 0) := "0000010";

constant seven : std\_logic\_vector(6 downto 0) := "1111000";

constant eight : std\_logic\_vector(6 downto 0) := "0000000";

constant nine : std\_logic\_vector(6 downto 0) := "0010000";

signal cathodes : std\_logic\_vector(6 downto 0); --segnale temporaneo per codificare

--il pattern sui 7 led in base alla cifra da mostrare

begin

seven\_segment\_decoder\_process: process

begin

case value is

when 0 => cathodes <= zero;

when 1 => cathodes <= one;

when 2 => cathodes <= two;

when 3 => cathodes <= three;

when 4 => cathodes <= four;

when 5 => cathodes <= five;

when 6 => cathodes <= six;

when 7 => cathodes <= seven;

when 8 => cathodes <= eight;

when 9 => cathodes <= nine;

when others => cathodes <= (others => '0');

end case;

end process seven\_segment\_decoder\_process;

cathodes\_dot <= (not dot)&cathodes; --segnale complessivo di 7+1 bit con catodi e punto

end Behavioral;

L’anodes\_manager gestisce l’accensione di una singola cifra

entity anodes\_manager is

Port ( counter : in STD\_LOGIC\_VECTOR (2 downto 0);

enable\_digit : in STD\_LOGIC\_VECTOR (7 downto 0);

anodes : out STD\_LOGIC\_VECTOR (7 downto 0)

);

end anodes\_manager;

architecture Behavioral of anodes\_manager is

signal anodes\_switching : std\_logic\_vector(7 downto 0) := (others => '0');

begin

anodes <= not anodes\_switching OR not enable\_digit;

anodes\_process: process(counter)

begin

--a seconda del valore di counter le cifre si illuminano una alla volta da destra a sinistra

case counter is

when "000" =>

anodes\_switching <= "00000001";

when "001" =>

anodes\_switching <= "00000010";

when "010" =>

anodes\_switching <= "00000100";

when "011" =>

anodes\_switching <= "00001000";

when "100" =>

anodes\_switching <= "00010000";

when "101" =>

anodes\_switching <= "00100000";

when "110" =>

anodes\_switching <= "01000000";

when "111" =>

anodes\_switching <= "10000000";

when others =>

anodes\_switching <= (others => '0');

end case;

end process;

end Behavioral;

Il cathodes\_input\_manager, in base al valore ricevuto dal contatore, accende una specifica cifra (il **nibble**) con valori e punti ricevuti sempre in input. Dovendo implementare un cronometro, e contando la prima cifra come cifra 0, le cifre 1, 3 e 5 dovranno mostrare le decine del valore ricevuto, mentre le cifre 0, 2 e 4 le unità. Per cui, in base alla cifra, si è effettuata un’operazione di modulo 10 o una divisione per 10, per ottenere la cifra desiderata da riportare sulla cifra singola del display.

entity cathodes\_input\_manager is

Port (

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

value\_in: in std\_logic\_vector(16 downto 0);

dots\_in: in std\_logic\_vector(7 downto 0); --configurazione dei punti degli 8 display

output : out integer; --nibble da mostrare in base alla selezione

dot\_out : out std\_logic --punto da mostrare in base alla selezione

);

end cathodes\_input\_manager;

architecture Behavioral of cathodes\_input\_manager is

begin

ciao:process(counter)

begin

case counter is

when "000" => output<=to\_integer(unsigned(value\_in(5 downto 0)))mod 10;

when "001" => output<=to\_integer(unsigned(value\_in(5 downto 0)))/10 ;

when "010" => output<=to\_integer(unsigned(value\_in(11 downto 6)))mod 10;

when "011" => output<=to\_integer(unsigned(value\_in(11 downto 6)))/10;

when "100" => output<=to\_integer(unsigned(value\_in(16 downto 12)))mod 10;

when "101" => output<=to\_integer(unsigned(value\_in(16 downto 12)))/10;

when others=> output<=0;

end case;

with counter select

dot\_out <= dots\_in(7)when "111",

dots\_in(6) when "110",

dots\_in(5) when "101",

dots\_in(4) when "100",

dots\_in(3) when "011",

dots\_in(2) when "010",

dots\_in(1) when "001",

dots\_in(0) when "000",

'0' when others;

end process;

end Behavioral;

Il Top Module si basa quindi su un [clock divider](#_Divisore_di_frequenza), che genera un segnale a frequenza sufficientemente alta da variare, senza che l’occhio umano possa accorgersene, l’accensione dei segmenti.

Il divisore di frequenza genera un segnale di abilitazione per un contatore modulo 8, che viene usato come segnale di conteggio nel cathodes\_input\_manager e quindi di fatto fornisce la frequenza con cui viene modificata la cifra da mostrare.

Il componente complessivo caricato su FPGA appare come segue:

entity cronometro\_on\_display is

Port ( CLK100MHZ: in std\_logic;

RST: in std\_logic;

SET: in std\_logic;

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

anodes : out STD\_LOGIC\_VECTOR (7 downto 0);

cathodes : out STD\_LOGIC\_VECTOR (7 downto 0)

);

end cronometro\_on\_display;

architecture Structural of cronometro\_on\_display 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;

component display\_on\_board is

Port(

clock : in STD\_LOGIC;

reset : in STD\_LOGIC;

load\_first\_part : in STD\_LOGIC;

load\_second\_part : in STD\_LOGIC;

load\_dots\_enable : in STD\_LOGIC;

value16\_in : in STD\_LOGIC\_VECTOR(15 downto 0);

anodes : out STD\_LOGIC\_VECTOR (7 downto 0);

cathodes : out STD\_LOGIC\_VECTOR (7 downto 0)

);

end component;

component display\_seven\_segments is

Generic(

clock\_frequency\_in : integer := 100000000;

clock\_frequency\_out : integer := 5000000

);

Port ( clock : in STD\_LOGIC;

reset : in STD\_LOGIC;

value32\_in : in STD\_LOGIC\_VECTOR (16 downto 0);

enable : in STD\_LOGIC\_VECTOR (7 downto 0);

dots : in STD\_LOGIC\_VECTOR (7 downto 0);

anodes : out STD\_LOGIC\_VECTOR (7 downto 0);

cathodes : out STD\_LOGIC\_VECTOR (7 downto 0));

end component;

component ButtonDebouncer is

generic (

CLK\_period: integer := 10;

btn\_noise\_time: integer := 10000000

);

Port ( RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

BTN : in STD\_LOGIC;

CLEARED\_BTN : out STD\_LOGIC);

end component;

signal button\_1: std\_logic;

signal button\_2: std\_logic;

signal chrono\_to\_display: std\_logic\_vector(16 downto 0);

signal seconds\_load : std\_logic\_vector(5 downto 0);

begin

button1:ButtonDebouncer

Port Map( RST=>'0',

CLK=>CLK100MHZ,

BTN=>RST,

CLEARED\_BTN=>button\_1

);

button2:ButtonDebouncer

Port Map( RST=>'0',

CLK=>CLK100MHZ,

BTN=>SET,

CLEARED\_BTN=>button\_2

);

seconds\_load <= "0"&SW(4 downto 0);

chrono: cronometro

Port Map( CLK=>CLK100MHZ,

RST=>button\_1,

set=>button\_2,

load\_second=>,seconds\_load

load\_minute=>SW(10 downto 5),

load\_hour=>SW(15 downto 11),

cont\_second=>chrono\_to\_display(5 downto 0),

cont\_minute=>chrono\_to\_display(11 downto 6),

cont\_hour=>chrono\_to\_display(16 downto 12)

);

display: display\_seven\_segments

Generic Map(

clock\_frequency\_in=>100000000,

clock\_frequency\_out=>1000

)

Port Map( clock=>CLK100MHZ,

reset=>button\_1,

value32\_in=>chrono\_to\_display,

enable=>"00111111",

dots=>"00000000",

anodes=>anodes,

cathodes=>cathodes

);

end Structural;

Quindi, mediante gli switch è possibile immettere i valori di ore, minuti e secondi. Vista l’assenza di uno switch, per consentire il precaricamento in tutti contatori, si è scelto di poter caricare come valore massimo per i secondi il numero “32”, anteponendo con una concatenazione uno “0”.

I *constraints* sono i seguenti:

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { CLK100MHZ }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {CLK100MHZ}];

## 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 { SW[14] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

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

## 7 segment display

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[0] }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[1] }]; #IO\_25\_14 Sch=cb

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[2] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[3] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[4] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[5] }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[6] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { cathodes[7] }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

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

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

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

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

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

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

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

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { anodes[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 { SET }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

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

### Esercizio 5.3 (solo 9 CFU)

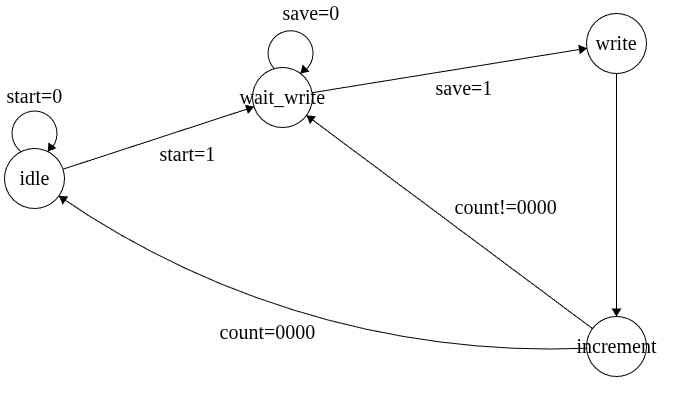
Estendere il componente sviluppato ai punti precedenti in modo che sia in grado di acquisire e memorizzare internamente fino ad un numero N di intertempi in corrispondenza di un ingresso di stop. Opzionalmente, il componente può prevedere una modalità di visualizzazione in cui, alla pressione di un bottone, vengano visualizzati sui display gli intertempi memorizzati (uno per ogni pressione).

#### Progetto e architettura

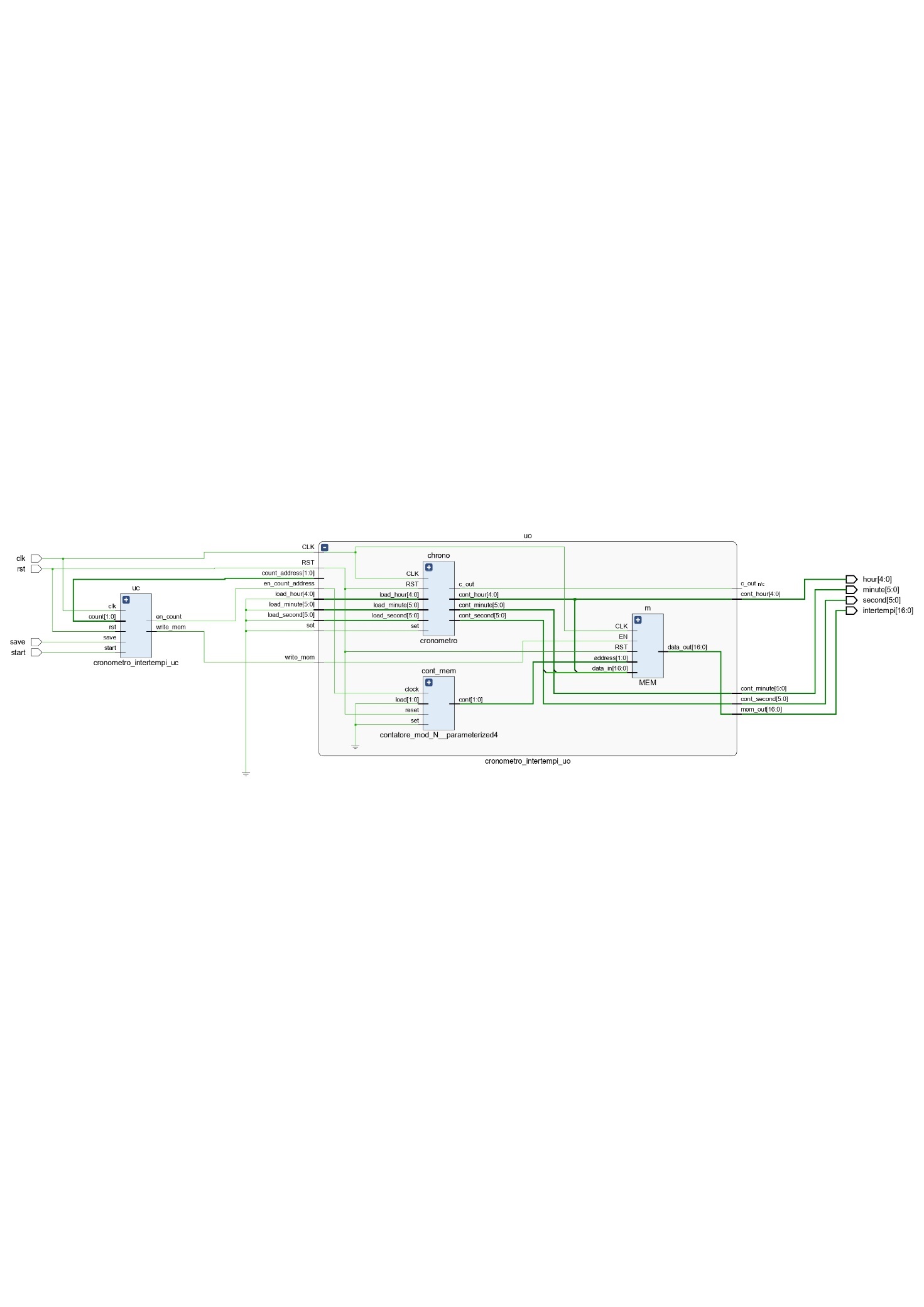
Per far sì che si possano salvare dei valori del cronometro è stato necessario progettare un Unità di Controllo come automa a stati finiti e un Unità operativa composta da:

* Una memoria su cui scrivere i valori da salvare.
* Un contatore per dare l’indirizzo di scrittura .
* Cronometro.

Unità di Controllo:



Unità Operativa:



#### Implementazione

Memoria MEM:

entity MEM is

Generic(N : natural :=8;

M : natural :=8);

Port ( CLK : in std\_logic;

RST : in std\_logic;

EN : in std\_logic;

data\_in : in std\_logic\_vector(M-1 downto 0);

address : in std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0); -- Codifica da 0 a N-1

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end MEM;

architecture Behavioral of MEM is

type MEMORY\_N\_M is array (0 to N-1) of std\_logic\_vector(M-1 downto 0); -- Matrice N x M

signal MEM\_DATA : MEMORY\_N\_M := (others => "UUUUUUUUUUUUUUUUU");

begin

process(CLK)

begin

if(CLKevent AND CLK = '1') then

if(EN = '1') then

MEM\_DATA(to\_integer(unsigned(address))) <= data\_in; -- Casting da unsigned a intero

elsif(RST = '1') then

MEM\_DATA <= (others => "00000000000000000");

end if;

end if;

end process;

data\_out <= MEM\_DATA(to\_integer(unsigned(address)));

end architecture Behavioral;

Unità operativa:

entity cronometro\_intertempi\_uo is

Generic(N: natural:=4);

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);

write\_mem: in std\_logic;

en\_count\_address: in std\_logic;

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

mem\_out: out std\_logic\_vector(16 downto 0);

c\_out : out std\_logic

);

end cronometro\_intertempi\_uo;

architecture Structural of cronometro\_intertempi\_uo 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;

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 MEM is

Generic(N : natural :=8;

M : natural :=8);

Port ( CLK : in std\_logic;

RST : in std\_logic;

EN : in std\_logic;

data\_in : in std\_logic\_vector(M-1 downto 0);

address : in std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0); -- Codifica da 0 a N-1

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end component;

signal chrono\_out:std\_logic\_vector(16 downto 0) ;

signal mem\_out\_signal:std\_logic\_vector(16 downto 0) ;

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

begin

chrono: cronometro

Port Map( CLK=>CLK,

RST=>RST,

set=>set,

load\_second=>load\_second,

load\_minute=>load\_minute,

load\_hour=>load\_hour,

cont\_second=>chrono\_out(5 downto 0),

cont\_minute=>chrono\_out(11 downto 6),

cont\_hour=>chrono\_out(16 downto 12),

c\_out=>c\_out

);

m:MEM

Generic Map(N=>N,M=>17)

Port Map( CLK =>CLK,

RST =>RST,

EN =>write\_mem,

data\_in=>chrono\_out,

address=>address,

data\_out=>mem\_out\_signal

);

cont\_mem:contatore\_mod\_N

Generic Map(N=>4)

Port Map( clock=>en\_count\_address,

reset=>rst,

set=>'0',

load=>(others=>'0'),

cont=>address

);

cont\_second<=chrono\_out(5 downto 0);

cont\_minute<=chrono\_out(11 downto 6);

cont\_hour<=chrono\_out(16 downto 12);

mem\_out<=mem\_out\_signal;

end Structural;

Unità di controllo:

entity cronometro\_intertempi\_uc is

Generic(N: natural:=4);

Port ( clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

start: in std\_logic;

en\_count : out STD\_LOGIC;

save: in std\_logic;

write\_mem : out STD\_LOGIC;

count : in STD\_LOGIC\_VECTOR (integer(ceil(log2(real(N))))-1 downto 0));

end cronometro\_intertempi\_uc;

architecture Behavioral of cronometro\_intertempi\_uc is

type state is (idle,wait\_write,write,increment);

signal stato\_corrente:state :=idle;

signal stato\_prossimo:state;

begin

comb:process(stato\_corrente,start,save)

begin

case stato\_corrente is

when idle =>

en\_count<='0';

if(start='0') then

stato\_prossimo<=idle;

else

stato\_prossimo<=wait\_write;

end if;

when wait\_write =>

en\_count<='0';

if(save='1') then

stato\_prossimo<=write;

else

stato\_prossimo<=wait\_write;

end if;

when write =>

write\_mem<='1';

stato\_prossimo<=increment;

when increment=>

write\_mem<='0';

en\_count<='1';

if(count="0000") then

stato\_prossimo<=idle;

else

stato\_prossimo<=wait\_write;

end if;

end case;

end process;

mem: process(clk)

begin

if clk'event and clk = '1' then

if rst = '1' then

stato\_corrente<=idle;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

Top Module:

entity cronometro\_intertempi is

Generic(N: natural :=4);

Port (

clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

start: in std\_logic;

save: in std\_logic;

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

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

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

intertempi : out std\_logic\_vector(16 downto 0)

);

end cronometro\_intertempi;

architecture Structural of cronometro\_intertempi is

component cronometro\_intertempi\_uc is

Generic(N: natural:=4);

Port ( clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

start: in std\_logic;

en\_count : out STD\_LOGIC;

save: in std\_logic;

write\_mem : out STD\_LOGIC;

count : in STD\_LOGIC\_VECTOR (integer(ceil(log2(real(N))))-1 downto 0)

);

end component;

component cronometro\_intertempi\_uo is

Generic(N: natural:=4);

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);

write\_mem: in std\_logic;

en\_count\_address: in std\_logic;

mem\_out: out std\_logic\_vector(16 downto 0);

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

c\_out : out std\_logic

);

end component;

signal en\_count\_sig:std\_logic;

signal write\_mem\_sig:std\_logic;

signal count\_sig:std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0);

signal sec\_sig:std\_logic\_vector(6 downto 0);

signal min\_sig:std\_logic\_vector(6 downto 0);

signal hour\_sig:std\_logic\_vector(5 downto 0);

begin

uc:cronometro\_intertempi\_uc

Generic Map(N=>N)

Port Map(

clk =>clk,

rst =>rst,

start =>start,

en\_count =>en\_count\_sig,

save =>save,

write\_mem =>write\_mem\_sig,

count =>count\_sig

);

uo:cronometro\_intertempi\_uo

Generic Map(N=>N)

Port Map(

CLK =>clk,

RST =>rst,

set =>'0',

load\_second =>(others=>'0'),

load\_minute =>(others=>'0'),

load\_hour =>(others=>'0'),

cont\_second =>second,

cont\_minute =>minute,

cont\_hour =>hour,

write\_mem =>write\_mem\_sig,

en\_count\_address =>en\_count\_sig,

count\_address=>count\_sig,

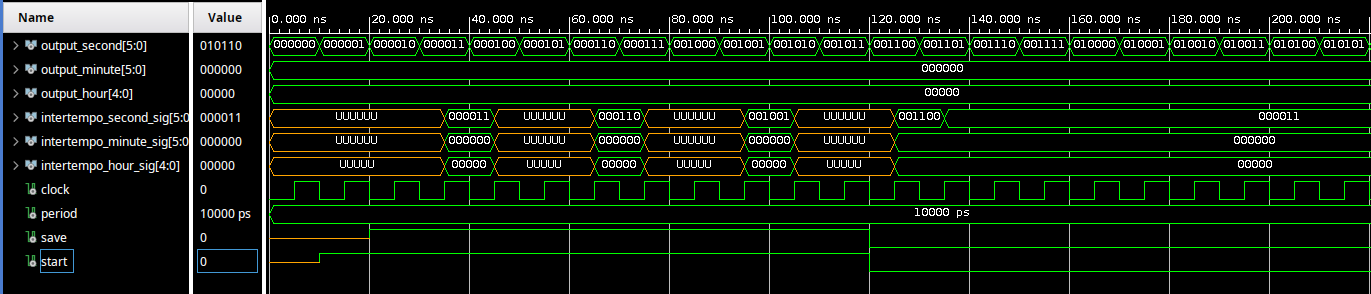
mem\_out =>intertempi

);

end Structural;

#### Simulazione

Testbench:

Risultato:

## 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.

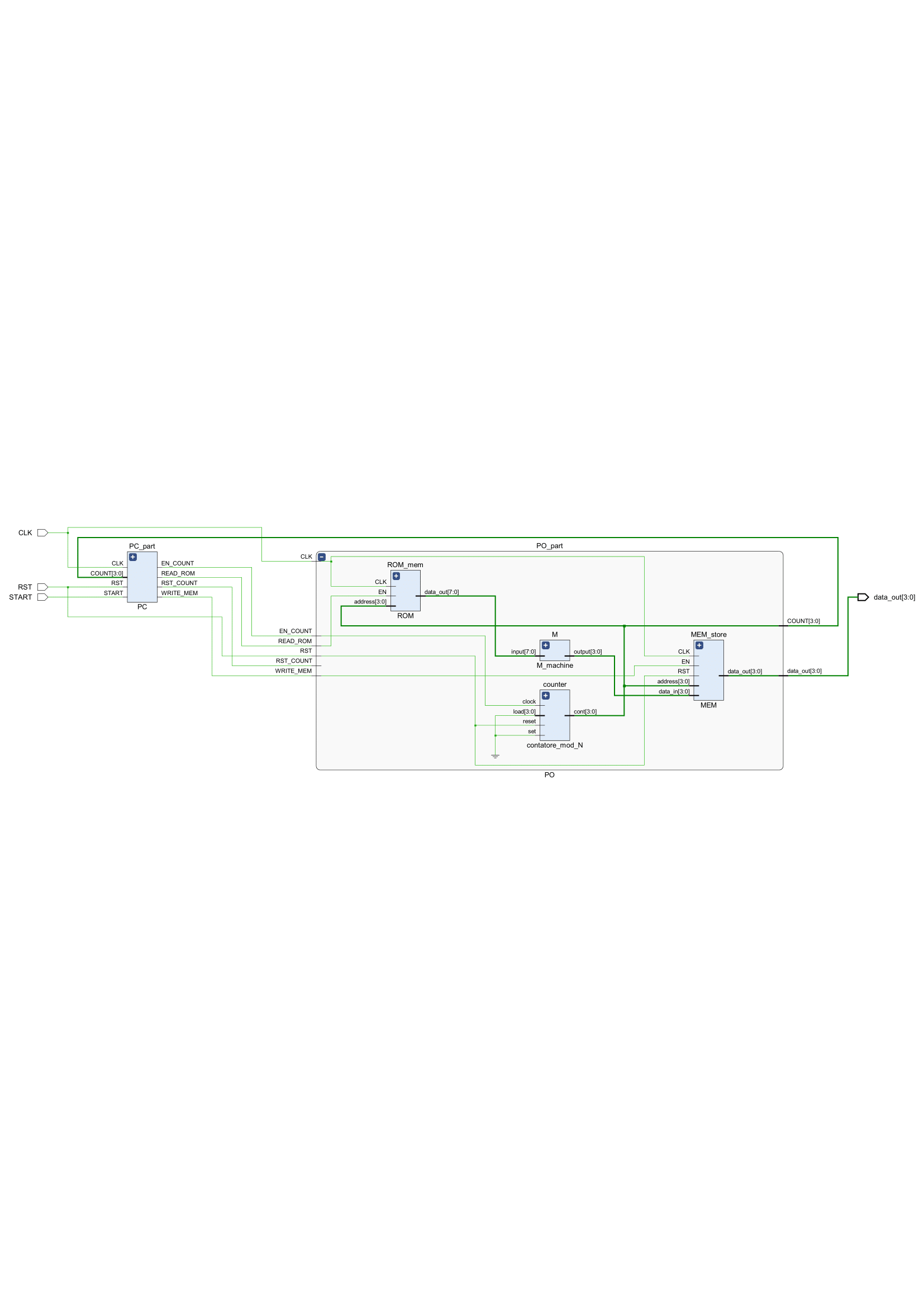
#### Progetto e architettura

Il sistema prevede una parte operativa (**PO**) e una parte di controllo (**PC**).

La parte di controllo, sul fronte di salita del clock del sistema, prende decisioni sulla base del segnale di START e del conteggio del contatore incaricato di scandire le locazioni della ROM e della MEM. Più nello specifico, implementa un automa a stati finiti per effettuare le operazioni desiderate in pochi cicli di clock.

La parte operativa, invece, contiene i componenti effettivi. Le macchine sequenziali della PO ricevono opportune abilitazioni dalla PC, al fine di ottenere un sincronismo completo nelle operazioni da seguire nel sistema. La macchina combinatoria M, invece, realizza la AND bitwise, come visto nel [capitolo 2.1](#_Esercizio_2.1).

L’architettura complessiva appare come segue:



#### Implementazione

L’implementazione del contatore si trova [nell’appendice](#_Contatore_modulo_N), mentre quella della macchina M è al [capitolo 2.1](#_Implementazione).

Si passerà dunque all’analisi delle memorie ROM e MEM con sincronismo.

La memoria ROM restituisce il dato in memoria, all’indirizzo dato, solo sul fronte di salita del clock e in caso di abilitazione attiva. Il tipo di dato per il contenuto della ROM è una matrice, come già visto in precedenza nel [capitolo 2.1](#_Esercizio_2.1):

entity ROM is

Generic(N : integer := 16);

Port ( CLK : in std\_logic;

EN : in std\_logic;

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

data\_out : out std\_logic\_vector(7 downto 0)

);

end ROM;

architecture Behavioral of ROM is

type MEMORY\_N\_8 is array (0 to N-1) of std\_logic\_vector(7 downto 0); -- Matrice N x 8

constant ROM\_DATA : MEMORY\_N\_8 := ( -- ROM content

"00010001",

"00110011",

"11111111",

"11111111",

"11111111",

"11111111",

"11111111",

"11111111",

"11111111",

"11111111",

"11111111",

"00000000",

"00000000",

"00000000",

"00000000",

"00000000"

);

begin

process(CLK)

begin

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

if(EN = '1') then

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

end if;

end if;

end process;

end architecture Behavioral;

Il contenuto della memoria MEM viene aggiornato all’indirizzo fornito in input e riportato di conseguenza in output. Un eventuale segnale di RESET, invece, causerebbe l’azzeramento della memoria:

entity MEM is

Generic(N : integer := 16);

Port ( CLK : in std\_logic;

RST : in std\_logic;

EN : in std\_logic;

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

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

data\_out : out std\_logic\_vector(3 downto 0)

);

end MEM;

architecture Behavioral of MEM is

type MEMORY\_N\_4 is array (0 to N-1) of std\_logic\_vector(3 downto 0); -- Matrice N x 4

signal MEM\_DATA : MEMORY\_N\_4 := (others => "0000");

begin

process(CLK)

begin

if(CLKevent AND CLK = '1') then

if(EN = '1') then

MEM\_DATA(to\_integer(unsigned(address))) <= data\_in; -- Casting da unsigned a intero

elsif(RST = '1') then

MEM\_DATA <= (others => "0000");

end if;

end if;

end process;

data\_out <= MEM\_DATA(to\_integer(unsigned(address)));

end architecture Behavioral;

La PO mette insieme, secondo un approccio **Structural**, i componenti visti.

entity PO is

Generic (N : integer := 16);

Port ( CLK : in std\_logic;

RST : in std\_logic;

READ\_ROM : in std\_logic;

WRITE\_MEM : in std\_logic;

EN\_COUNT : in std\_logic;

RST\_COUNT : in std\_logic;

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

data\_out : out std\_logic\_vector(3 downto 0)

);

end PO;

architecture Structural of PO is

component contatore\_mod\_N is

Generic (N: integer);

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 ROM is

Generic(N : integer);

Port ( CLK : in std\_logic;

EN : in std\_logic;

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

data\_out : out std\_logic\_vector(7 downto 0)

);

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;

component MEM is

Generic(N : integer);

Port ( CLK : in std\_logic;

RST : in std\_logic;

EN : in std\_logic;

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

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

data\_out : out std\_logic\_vector(3 downto 0)

);

end component;

signal rom\_out : std\_logic\_vector(7 downto 0) := (others => '0');

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

signal count\_end : std\_logic;

signal reset\_cont:std\_logic;

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

begin

counter : contatore\_mod\_N

Generic Map (N=>N)

Port Map ( clock=>EN\_COUNT,

reset=>RST ,

set=>'0',

load=>(others => '0'),

cont=>count\_in,

co=>count\_end

);

COUNT <= count\_in; -- Aggiornamento del contatore per il sistema completo

reset\_cont <= count\_end OR RST OR RST\_COUNT;

ROM\_mem : ROM

Generic Map (N=>N)

Port Map ( CLK=>CLK,

EN=>READ\_ROM,

address=>count\_in,

data\_out=>rom\_out

);

M : M\_machine

Port Map (

input=>rom\_out,

output=>mem\_in

);

MEM\_store: MEM

Generic Map (N=>N)

Port Map ( CLK=>CLK,

RST=>RST,

EN=>WRITE\_MEM,

data\_in=>mem\_in,

address=>count\_in,

data\_out=>data\_out

);

end Structural;

La PC realizza l’automa e l’aggiornamento dei segnali di controllo dei dispositivi. La FSM parte da uno stato di IDLE, in attesa del segnale di start; si passa poi allo stato READROM in cui viene scandita la -esima locazione della ROM; in WRITEMEM viene abilitata la scrittura della corrispondente locazione in MEM; infine, si procede al conteggio, e si torna in IDLE se la scansione è terminata (ovvero se il contatore ha raggiunto il valore massimo):

entity PC is

Generic (N : integer := 16);

Port ( CLK : in std\_logic;

RST : in std\_logic;

START : in std\_logic;

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

READ\_ROM : out std\_logic;

WRITE\_MEM : out std\_logic;

EN\_COUNT : out std\_logic;

RST\_COUNT : out std\_logic

);

end PC;

architecture Behavioral of PC is

type state is (IDLE, READROM, WRITEMEM, CNT);

signal stato\_corrente : state := IDLE;

signal stato\_prossimo : state;

begin

comb: process(stato\_corrente, START, COUNT)

begin

case stato\_corrente is

when IDLE =>

RST\_COUNT <= '1';

EN\_COUNT <= '0';

if(START = '1') then

stato\_prossimo <= READROM;

elsif(START ='0') then

stato\_prossimo <= IDLE;

end if;

when READROM =>

EN\_COUNT <= '0';

READ\_ROM <= '1';

stato\_prossimo <= WRITEMEM;

when WRITEMEM =>

READ\_ROM <= '0';

WRITE\_MEM <= '1';

stato\_prossimo <= CNT;

when CNT =>

WRITE\_MEM <= '0';

EN\_COUNT <= '1';

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

stato\_prossimo <= IDLE;

else

stato\_prossimo <= READROM;

end if;

end case;

end process;

mem: process(CLK)

begin

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

if(RST = '1') then

stato\_corrente <= IDLE;

else

stato\_corrente <= stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

Il sistema complessivo unisce PO e PC in un’unica entità *black-box*:

entity PO\_PC\_system is

Generic (N : integer := 16);

Port ( CLK : in std\_logic;

RST : in std\_logic;

START : in std\_logic;

data\_out : out std\_logic\_vector(3 downto 0)

);

end PO\_PC\_system;

architecture Structural of PO\_PC\_system is

component PO is

Generic (N : integer);

Port ( CLK : in std\_logic;

RST : in std\_logic;

READ\_ROM : in std\_logic;

WRITE\_MEM : in std\_logic;

EN\_COUNT : in std\_logic;

RST\_COUNT : in std\_logic;

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

data\_out : out std\_logic\_vector(3 downto 0)

);

end component;

component PC is

Generic (N : integer);

Port ( CLK : in std\_logic;

RST : in std\_logic;

START : in std\_logic;

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

READ\_ROM : out std\_logic;

WRITE\_MEM : out std\_logic;

EN\_COUNT : out std\_logic;

RST\_COUNT : out std\_logic

);

end component;

signal rom\_read : std\_logic;

signal mem\_write : std\_logic;

signal count\_en : std\_logic;

signal count\_rst : std\_logic;

signal count\_value : std\_logic\_vector(integer(ceil(log2(real(N))))-1 downto 0);

begin

PO\_part: PO

Generic Map (N=>N)

Port Map ( CLK=>CLK,

RST=>RST,

READ\_ROM=>rom\_read,

WRITE\_MEM=>mem\_write,

EN\_COUNT=>count\_en,

RST\_COUNT=>count\_rst,

COUNT=>count\_value,

data\_out=>data\_out

);

PC\_part: PC

Generic Map (N=>N)

Port Map ( CLK=>CLK,

RST=>RST,

START=>START,

COUNT=>count\_value,

READ\_ROM=>rom\_read,

WRITE\_MEM=>mem\_write,

EN\_COUNT=>count\_en,

RST\_COUNT=>count\_rst

);

end Structural;

#### Simulazione

La testbench fornisce semplicemente i segnali di START e RESET per verificare il funzionamento.

entity PO\_PC\_system\_tb is

end PO\_PC\_system\_tb;

architecture Behavioral of PO\_PC\_system\_tb is

component PO\_PC\_system is

Generic (N : integer);

Port ( CLK : in std\_logic;

RST : in std\_logic;

START : in std\_logic;

data\_out : out std\_logic\_vector(3 downto 0)

);

end component;

signal clock : std\_logic;

constant CLK\_period : time := 10 ns;

signal reset : std\_logic;

signal start : std\_logic;

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

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

begin

uut: PO\_PC\_system

Generic Map (N=>16)

Port Map ( CLK=>clock,

RST=>reset,

START=>start,

data\_out=>data

);

clk : process

begin

clock <= '0';

wait for CLK\_period/2;

clock <= '1';

wait for CLK\_period/2;

end process;

test : process

begin

start <= '1';

wait for 10\*3\*16 ns;

reset<='1';

start<='0';

wait for CLK\_PERIOD;

reset<='0';

wait for CLK\_PERIOD;

start<='1';

wait for 10000000 ns;

wait;

end process;

end Behavioral;

Alla prima iterazione, la MEM è vuota; per cui, in vari istanti (precisamente, quando il sistema incrementa il valore del contatore), vi sono degli istanti in cui in output viene visualizzato un valore non ancora aggiornato dalla macchina M (‘0000’):

Immagine che contiene schermata, Software multimediale, software, linea

Descrizione generata automaticamente

Si mostra anche il corretto funzionamento di RST, che provvede ad azzerare il contenuto della memoria (e tale effetto è visualizzabile in output) non appena vi sarà il fronte di salita del clock del sistema.

Immagine che contiene schermata, linea

Descrizione generata automaticamente

### 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

Per la sintesi, è stato necessario utilizzare due [Button Debouncer](#_Button_Debouncer_1) per il caricamento dei segnali di START e RESET, il clock e dei LED in output.

entity PO\_PC\_fpga is

Port ( CLK100MHZ : in std\_logic;

BTNL : in std\_logic;

BTNR : in std\_logic;

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

);

end PO\_PC\_fpga;

architecture Structural of PO\_PC\_fpga is

component PO\_PC\_system

Generic (N : integer := 16);

Port ( CLK : in std\_logic;

RST : in std\_logic;

START : in std\_logic;

data\_out : out std\_logic\_vector(3 downto 0)

);

end component;

component divisore\_di\_frequenza is

generic(

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

CLKOUT\_freq : integer := 1000000 -- frequenza desiderata 1Hz

);

Port (

clock\_in : in STD\_LOGIC;

reset : in STD\_LOGIC;

clock\_out : out STD\_LOGIC

);

end component;

component ButtonDebouncer is

generic (

CLK\_period: integer := 10;

btn\_noise\_time: integer := 10000000

);

Port ( RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

BTN : in STD\_LOGIC;

CLEARED\_BTN : out STD\_LOGIC);

end component;

signal left\_button : std\_logic;

signal right\_button : std\_logic;

signal wave: std\_logic;

begin

div: divisore\_di\_frequenza

Port Map(

clock\_in=>CLK100MHZ,

reset=>'0',

clock\_out=>wave

);

button1: ButtonDebouncer

Port Map( RST=>'0',

CLK=>CLK100MHZ,

BTN=>BTNL,

CLEARED\_BTN=>left\_button

);

button2: ButtonDebouncer

Port Map( RST=>'0',

CLK=>CLK100MHZ,

BTN=>BTNR,

CLEARED\_BTN=>right\_button

);

system: PO\_PC\_system

Port Map( CLK=> CLK100MHZ ,

RST=>left\_button,

START=>right\_button,

data\_out=>LED

);

end Structural;

Il file di *constraints* appare come segue:

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { CLK100MHZ }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10000000.00 -waveform {0 5} [get\_ports {CLK100MHZ}];

## 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]

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

##Buttons

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

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

# **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

Il moltiplicatore di Booth parte dalla codifica di Booth, che utilizza opportunamente moltiplicazioni bit a bit tra gli operandi, per ottenere il risultato completo e risparmiando notevolmente in termini computazionali e di risorse rispetto a una moltiplicazione classica.

In particolare, fissando:

e moltiplicando ogni equazione per , si può passare da una rappresentazione posizionale a una rappresentazione in complementi a due:

Una stringa può essere facilmente codificata seguendo la rappresentazione Booth analizzando gruppi di 2 bit alla volta:

|  |  |
| --- | --- |
|  | Codifica |
| 00/11 | 0 |
| 01 | +1 |
| 10 | -1 |

Fatta eccezione per la prima iterazione, nella quale non conosciamo il bit in posizione che verrà per tal motivo inizializzato a 0, la tabella della codifica di Booth permette, a ogni passo, di conoscere il segno dell’operazione. In tal modo, a partire da una codifica *unsigned*, è possibile effettuare anche operazioni con segno.

A partire da tale codifica, è stato sviluppato l’algoritmo da seguire per implementare la moltiplicazione tra numeri relativi.

In particolare, a seconda della cifra coinvolta nel prodotto verrà effettuata una somma seguita da uno shift, una differenza seguita da uno shift o un semplice shift.

In sostanza, viene effettuata una moltiplicazione bit a bit tra due operandi X e Y grazie allo shift di posizione.

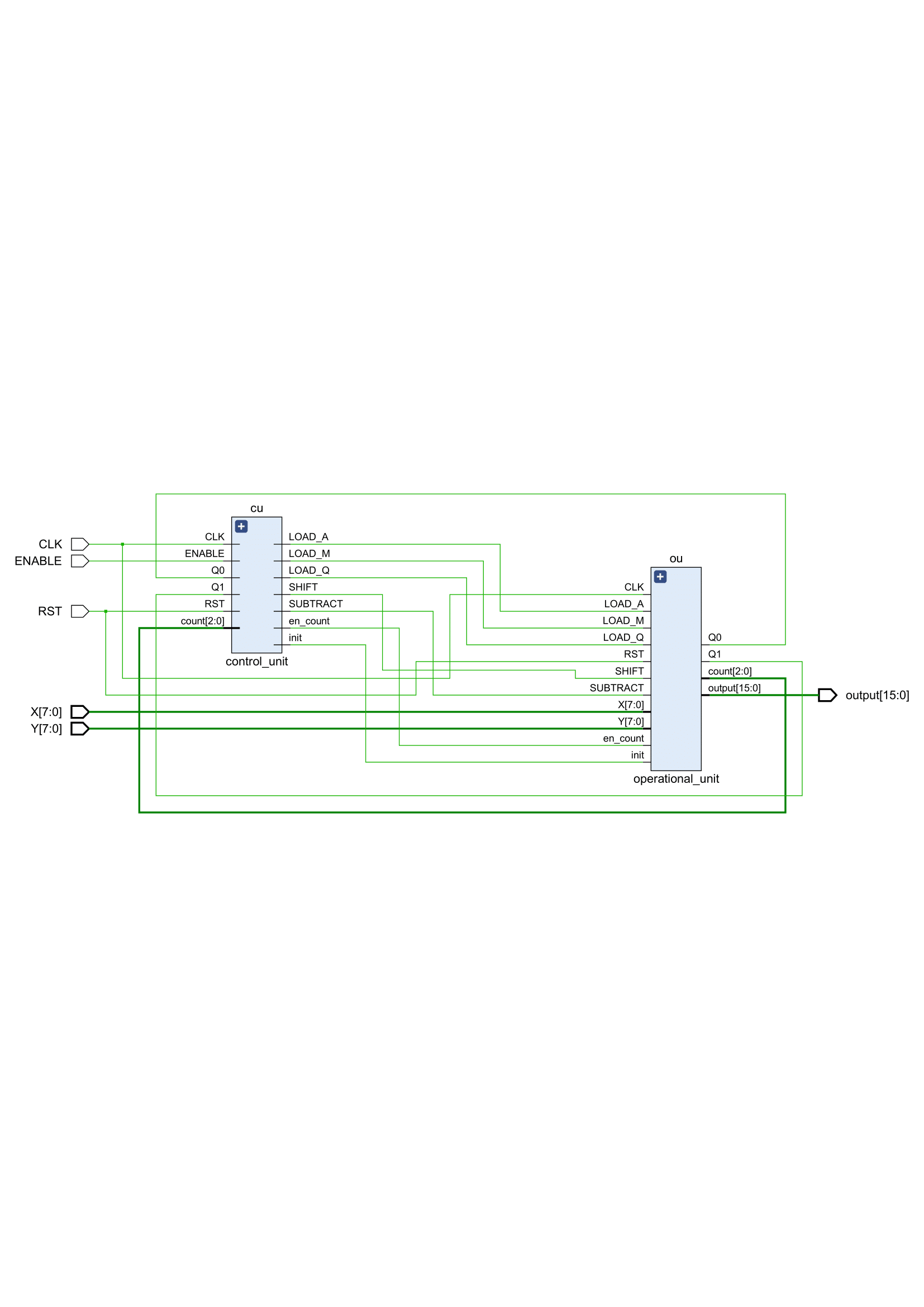
Analizzando i due bit dell’operando X, e mantenendo fisso l’operando Y, le operazioni da svolgere sono le seguenti:

|  |  |
| --- | --- |
|  | Operazione |
| 00/11 | Shift |
| 01 | Addizione |
| 10 | Sottrazione |

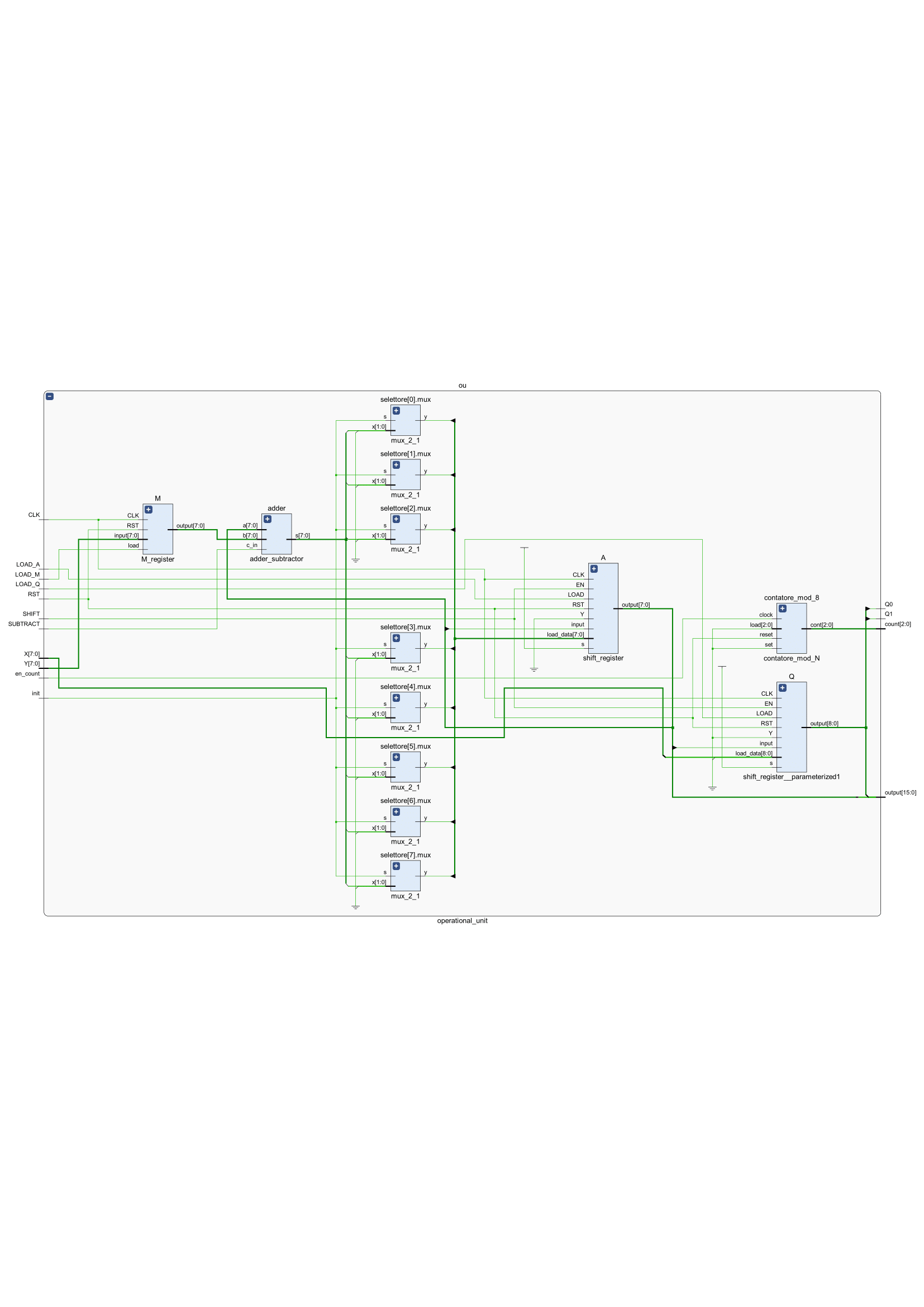
Per realizzare il componente, si è proceduto seguendo un approccio **Structural**. In particolare, vi sono:

* un **registro M**, che conterrà il valore di Y;
* un **registro a scorrimento Q**, che conterrà il valore di X e implementerà l’analisi bit a bit fornendo progressivamente gli ultimi due bit alla CU;
* un **registro a scorrimento A**, che salva progressivamente il risultato della moltiplicazione sommando man mano i prodotti parziali;
* un **Ripple Carry Adder parallelo**, che supporta l’operazione di sottrazione in caso di *carry* alto in ingresso, e che somma progressivamente l’operando Y con il risultato parziale di A;
* un **contatore modulo N**, per effettuare N volte l’operazione;
* una **Control Unit**, che valuta i bit e il valore del conteggio, e produce in output dei segnali di controllo per l’abilitazione del contatore, l’abilitazione della sottrazione sulla base dei bit di Q letti e quindi anche l’eventuale carry per il sommatore;
* un **banco di multiplexer 2:1**, per decidere, in fase iniziale, se inizializzare A con valori nulli oppure caricare il valore della somma.

L’architettura complessiva appare come segue:



La parte operativa, nel dettaglio, si trova alla pagina successiva:



#### Implementazione

Il multiplexer 2:1 è stato realizzato secondo un approccio **Dataflow**, sfruttando il componente realizzato per il [multiplexer 4:1 ad albero](#_Implementazione_1). Anche il contatore modulo N è consultabile [nell’appendice](#_Contatore_modulo_N). Sono stati, poi, utilizzati due registri a scorrimento per A e Q, la cui implementazione è analoga a quanto già è stato osservato [nell’esercizio 4.1](#_Esercizio_4.1).

Il sommatore è stato realizzato secondo un approccio **Structural**. Partendo da un semplice Full-Adder, e implementando N volte tale componente in un Ripple Carry Adder, si è realizzato un top-module che aggiunge una porta XOR per l’eventuale riporto alto, per svolgere la sottrazione.

Full-Adder:

entity full\_adder is

Port ( a:in std\_logic;

b:in std\_logic;

c\_in:in std\_logic;

s:out std\_logic;

c\_out:out std\_logic

);

end full\_adder;

architecture Dataflow of full\_adder is

begin

s<=((a xor b) xor c\_in);

c\_out<=( (a and b) or (c\_in and (a xor b)) );

end Dataflow;

Ripple Carry Adder parallelo:

entity ripple\_carry\_adder is

Generic(N : natural :=8);

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

b:in std\_logic\_vector(N-1 downto 0);

c\_in: in std\_logic;

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

c\_out : out std\_logic

);

end ripple\_carry\_adder;

architecture Structural of ripple\_carry\_adder is

component full\_adder is

Port ( a:in std\_logic;

b:in std\_logic;

c\_in:in std\_logic;

s:out std\_logic;

c\_out:out std\_logic

);

end component;

signal carry: std\_logic\_vector(N-2 downto 0):=(others=>'0');

begin

fa\_0: full\_adder

Port Map( a=>a(0),

b=>b(0),

c\_in=>c\_in,

s=>s(0),

c\_out=>carry(0)

);

fa\_chain: for i in N-2 downto 1 generate

fa:full\_adder

Port Map( a=>a(i),

b=>b(i),

c\_in=>carry(i-1),

s=>s(i),

c\_out=>carry(i)

);

end generate;

fa\_N\_1: full\_adder

Port Map( a=>a(N-1),

b=>b(N-1),

c\_in=>carry(N-2),

s=>s(N-1),

c\_out=>c\_out

);

end Structural;

Adder-Subtractor:

entity adder\_subtractor is

Generic(N : natural :=8);

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

b:in std\_logic\_vector(N-1 downto 0);

c\_in: in std\_logic;

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

c\_out : out std\_logic

);

end adder\_subtractor;

architecture Structural of adder\_subtractor is

component ripple\_carry\_adder is

Generic(N : natural :=8);

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

b:in std\_logic\_vector(N-1 downto 0);

c\_in: in std\_logic;

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

c\_out : out std\_logic

);

end component;

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

begin

complemento\_b:for i in N-1 downto 0 generate

complementob(i)<=b(i) XOR c\_in;

end generate;

rca:ripple\_carry\_adder

Generic Map(N=>N)

Port Map( a=>a,

b=>complementob,

c\_in=>c\_in,

s=>s,

c\_out=>c\_out

);

end Structural;

Il registro M opera come segue:

entity M\_register is

Generic(N: natural:=8);

Port ( CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

load : in STD\_LOGIC;

input : in STD\_LOGIC\_VECTOR(N-1 downto 0);

output : out STD\_LOGIC\_VECTOR(N-1 downto 0)

);

end M\_register;

architecture Behavioral of M\_register is

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

begin

process(CLK)

begin

if(rising\_edge(CLK)) then

if(RST='1') then

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

elsif(load='1') then

temp<=input;

end if;

end if;

end process;

output<=temp;

end Behavioral;

L’unità operativa complessiva appare come segue:

entity operational\_unit is

GENERIC(N: natural :=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

X: in std\_logic\_vector(N-1 downto 0);

Y: in std\_logic\_vector(N-1 downto 0);

init: in std\_logic;

SHIFT: in std\_logic;

LOAD\_A: in std\_logic;

LOAD\_Q: in std\_logic;

LOAD\_M: in std\_logic;

SUBTRACT: in std\_logic;

en\_count: in std\_logic;

Q0: out std\_logic;

Q1: out std\_logic;

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

output: out std\_logic\_vector( 2\*N-1 downto 0)

);

end operational\_unit;

architecture Structural of operational\_unit is

component shift\_register is

Generic(N : integer := 8);

Port ( CLK : in std\_logic; -- Abilitazione

RST : in std\_logic;

input : in std\_logic;

LOAD : in std\_logic;

load\_data: in std\_logic\_vector(N-1 downto 0);

EN : in std\_logic;

s : in std\_logic:='1';

Y : in std\_logic:='0';

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

);

end component;

component M\_register is

Generic(N: natural:=8);

Port ( CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

load : in STD\_LOGIC;

input : in STD\_LOGIC\_VECTOR(N-1 downto 0);

output : out STD\_LOGIC\_VECTOR(N-1 downto 0)

);

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 adder\_subtractor is

Generic(N : natural :=8);

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

b:in std\_logic\_vector(N-1 downto 0);

c\_in: in std\_logic;

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

c\_out : out std\_logic

);

end component;

component mux\_2\_1 is

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

s : in std\_logic;

y : out std\_logic

);

end component;

signal A\_to\_Q :std\_logic;

signal sum\_to\_A:std\_logic\_vector(N-1 downto 0);

signal M\_out:std\_logic\_vector(N-1 downto 0);

signal output\_temp : std\_logic\_vector(2\*N-1 downto 0);

signal a\_input:std\_logic:='0';

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

signal shift\_data : std\_logic\_vector(N downto 0);

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

begin

selettore:for i in N-1 downto 0 generate

mux:mux\_2\_1

Port Map( x(0)=>sum\_to\_A(i),

x(1)=>'0',

s=>init,

y=>mux\_to\_A(i)

);

end generate;

A: shift\_register

Generic Map(N=>N)

Port Map( CLK=>CLK,

RST=>RST,

input=>a\_input,

LOAD=>LOAD\_A,

load\_data=>mux\_to\_A,

EN=>SHIFT,

s=>'1',

Y=>'0',

output=>output\_temp(2\*N-1 downto N)

);

a\_input <= output\_temp(2\*N-1);

A\_to\_Q <= output\_temp(N);

shift\_data <= X&"0";

Q: shift\_register

Generic Map(N=>N+1)

Port Map( CLK=>CLK,

RST=>RST,

input=>A\_to\_Q,

LOAD=>LOAD\_Q,

load\_data=>shift\_data,

EN=>SHIFT,

s=>'1',

Y=>'0',

output(N downto 1)=>output\_temp(N-1 downto 0),

output(0)=>Q0

);

Q1<=output\_temp(0);

M:M\_register

Generic Map(N=>N)

Port Map( CLK=>CLK,

RST=>RST,

load=>LOAD\_M,

input=>Y,

output=>M\_out

);

contatore\_mod\_8:contatore\_mod\_N

Generic Map(N=>N)

Port Map( clock=>en\_count ,

reset=>RST,

set=>'0',

load=>(others=>'0'),

cont=>count

);

adder:adder\_subtractor

Generic Map(N=>N)

Port Map( a=>output\_temp(2\*N-1 downto N),

b=>M\_out,

c\_in=>SUBTRACT,

s=>sum\_to\_A

);

output<=output\_temp;

end Structural;

L’unità di controllo, invece, realizza la logica dell’algoritmo vero e proprio gestendo opportunamente i segnali di controllo di tutti i dispositivi, come spiegato in precedenza:

entity control\_unit is

Generic(N: natural:=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

ENABLE: in std\_logic;

Q0: in std\_logic;

Q1: in std\_logic;

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

SHIFT:out std\_logic;

LOAD\_A: out std\_logic;

LOAD\_Q: out std\_logic;

LOAD\_M: out std\_logic;

SUBTRACT: out std\_logic;

init: out std\_logic;

en\_count: out std\_logic

);

end control\_unit;

architecture Behavioral of control\_unit is

type state is (IDLE,SCAN,SUM\_STATE,SUBTRACT\_STATE,LOADA\_STATE,RSHIFT,INCREMENT);

signal stato\_corrente: state :=IDLE;

signal stato\_prossimo: state;

begin

comb:process(stato\_corrente,ENABLE,Q0,Q1,count)

begin

case stato\_corrente is

when IDLE =>

if(ENABLE='1') then

SHIFT<='0';

init<='1';

LOAD\_A<='1';

LOAD\_Q<='1';

LOAD\_M<='1';

stato\_prossimo<=SCAN;

else

stato\_prossimo<=IDLE;

end if;

when SCAN =>

en\_count<='0';

SHIFT<='0';

init<='0';

LOAD\_A<='0';

LOAD\_Q<='0';

LOAD\_M<='0';

if(Q1='0' AND Q0='1') then

stato\_prossimo<=SUM\_STATE;

elsif(Q1='1' AND Q0='0') then

stato\_prossimo<=SUBTRACT\_STATE;

else

stato\_prossimo<=RSHIFT;

end if;

when SUM\_STATE =>

SUBTRACT<='0';

stato\_prossimo<=LOADA\_STATE;

when SUBTRACT\_STATE =>

SUBTRACT<='1';

stato\_prossimo<=LOADA\_STATE;

when LOADA\_STATE =>

SHIFT<='0';

LOAD\_A<='1';

stato\_prossimo<=RSHIFT;

when RSHIFT =>

LOAD\_A<='0';

SHIFT<='1';

stato\_prossimo<=INCREMENT;

when INCREMENT =>

SHIFT<='0';

en\_count<='1';

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

stato\_prossimo<=IDLE;

else

stato\_prossimo<=SCAN;

end if;

end case;

end process;

mem: process(CLK)

begin

if(rising\_edge(CLK)) then

if(RST='1') then

stato\_corrente<=IDLE;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

Il Top-Module complessivo, in definitiva, è il seguente:

entity booth\_multiplier is

Generic(N: natural :=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

ENABLE: in std\_logic;

X: in std\_logic\_vector(N-1 downto 0);

Y: in std\_logic\_vector(N-1 downto 0);

output: out std\_logic\_vector(2\*N-1 downto 0)

);

end booth\_multiplier;

architecture Structural of booth\_multiplier is

component control\_unit is

Generic(N: natural:=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

ENABLE: in std\_logic;

Q0: in std\_logic;

Q1: in std\_logic;

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

SHIFT:out std\_logic;

LOAD\_A: out std\_logic;

LOAD\_Q: out std\_logic;

LOAD\_M: out std\_logic;

SUBTRACT: out std\_logic;

init: out std\_logic;

en\_count: out std\_logic

);

end component;

component operational\_unit is

GENERIC(N: natural :=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

X: in std\_logic\_vector( N-1 downto 0);

Y: in std\_logic\_vector( N-1 downto 0);

init: in std\_logic;

SHIFT: in std\_logic;

LOAD\_A: in std\_logic;

LOAD\_Q: in std\_logic;

LOAD\_M: in std\_logic;

SUBTRACT: in std\_logic;

en\_count: in std\_logic;

Q0: out std\_logic;

Q1: out std\_logic;

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

output: out std\_logic\_vector( 2\*N-1 downto 0)

);

end component;

signal Q0: std\_logic;

signal Q1: std\_logic;

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

signal SHIFT: std\_logic;

signal LOAD\_A: std\_logic;

signal LOAD\_Q: std\_logic;

signal LOAD\_M: std\_logic;

signal SUBTRACT: std\_logic;

signal en\_count: std\_logic ;

signal init: std\_logic;

begin

cu: control\_unit

Generic Map(N=>N)

Port Map( CLK=>CLK,

RST=>RST,

ENABLE=>ENABLE,

Q0=>Q0,

Q1=>Q1,

count=>count,

SHIFT=>SHIFT,

LOAD\_A =>LOAD\_A,

LOAD\_Q =>LOAD\_Q,

LOAD\_M=>LOAD\_M,

SUBTRACT=>SUBTRACT,

init=>init,

en\_count=>en\_count

);

ou: operational\_unit

Generic Map(N=>N)

Port Map( CLK=>CLK,

RST=>RST,

X=>X,

Y=>Y,

init=>init,

SHIFT=>SHIFT,

LOAD\_A=>LOAD\_A,

LOAD\_Q=>LOAD\_Q,

LOAD\_M=>LOAD\_M,

SUBTRACT=>SUBTRACT,

en\_count=>en\_count,

Q0=>Q0,

Q1=>Q1,

count=>count,

output=>output

);

end Structural;

#### Simulazione

Il codice della simulazione è il seguente, e simula l’operazione sfruttando l’algoritmo di Booth:

entity booth\_multiplier\_tb is

Generic(N: natural :=4);

end booth\_multiplier\_tb;

architecture Behavioral of booth\_multiplier\_tb is

component booth\_multiplier is

Generic(N: natural :=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

ENABLE: in std\_logic;

X: in std\_logic\_vector(N-1 downto 0);

Y: in std\_logic\_vector(N-1 downto 0);

output: out std\_logic\_vector(2\*N-1 downto 0)

);

end component;

signal clock: std\_logic;

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

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

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

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

signal output: std\_logic\_vector(2\*N-1 downto 0);

signal CLOCK\_PERIOD : time := 10 ns;

begin

uut:booth\_multiplier

Generic Map(N=>N)

Port Map( CLK=>clock,

RST=>rst,

ENABLE=>enable,

X=>x,

Y=>y,

output=>output

);

cwave:process

begin

clock<='0';

wait for CLOCK\_PERIOD/2;

clock<='1';

wait for CLOCK\_PERIOD/2;

end process;

stim:process

begin

rst<='0';

wait for 10 ns;

rst<='0';

x<="0111";

y<="0110";

enable<='1';

wait for 10 ns;

enable<='0';

wait for 800 ns;

rst<='1';

wait;

end process;

end Behavioral;

Si può osservare il corretto funzionamento sia del moltiplicatore che del segnale di RESET.

Immagine che contiene schermata, linea, Policromia

Descrizione generata automaticamente

### 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

Per implementare il componente su FPGA, è bastato utilizzare dei [Button Debouncer](#_Button_Debouncer_1) per realizzare rispettivamente i segnali di reset e di enable, i primi 8 switch per il caricamento di X, gli ultimi 8 per il caricamento di Y e i LED per la visualizzazione delle uscite.

entity booth\_fpga is

Generic ( N: natural :=8);

Port ( CLK100MHZ : in std\_logic;

BTNR: in std\_logic;

SW\_X: in std\_logic\_vector(N-1 downto 0);

SW\_Y: in std\_logic\_vector(N-1 downto 0);

BTNL: in std\_logic;

LED: out std\_logic\_vector(2\*N-1 downto 0)

);

end booth\_fpga;

architecture Structural of booth\_fpga is

component booth\_multiplier is

Generic(N: natural :=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

ENABLE: in std\_logic;

X: in std\_logic\_vector(N-1 downto 0);

Y: in std\_logic\_vector(N-1 downto 0);

output: out std\_logic\_vector(2\*N-1 downto 0)

);

end component;

component 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 ns

);

Port ( RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

BTN : in STD\_LOGIC;

CLEARED\_BTN : out STD\_LOGIC);

end component;

signal BTNR\_sig: std\_logic;

signal BTNL\_sig: std\_logic;

begin

rst:ButtonDebouncer

Port Map( RST=>BTNR\_sig,

CLK=>CLK100MHZ,

BTN=>BTNR,

CLEARED\_BTN=>BTNR\_sig

);

start:ButtonDebouncer

Port Map( RST=>BTNR\_sig,

CLK=>CLK100MHZ,

BTN=>BTNL,

CLEARED\_BTN=>BTNL\_sig

);

boot: booth\_multiplier

Generic Map(N=>N)

Port Map( CLK=>CLK100MHZ,

RST=>BTNR\_sig,

ENABLE=>BTNL\_sig,

X=>SW\_X(N-1 downto 0),

Y=>SW\_Y(N-1 downto 0),

output=>LED

);

end Structural;

I *constraints* sono i seguenti:

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { CLK100MHZ }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {CLK100MHZ}];

## Switches

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 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]

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]

## 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 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

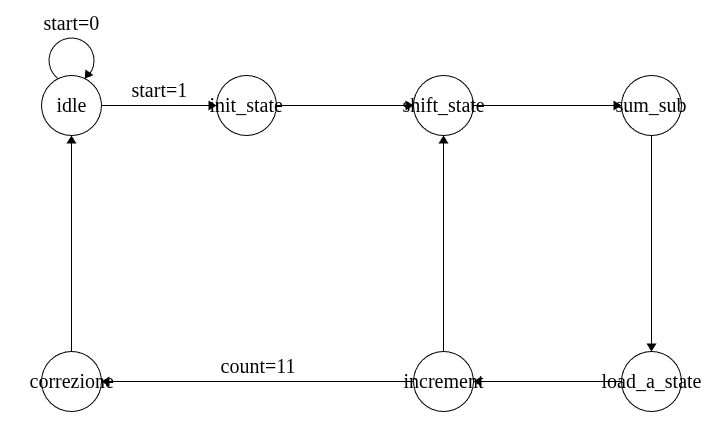
Per la realizzazione del divisore si è diviso il sistema in Parte Operativa e Parte di controllo, la cui ultima implementa tramite un automa a stati finiti i passi dell’algoritmo e controlla l’utilizzo delle componenti presenti nella parte operativa.

L’ Unità Operativa è composta da:

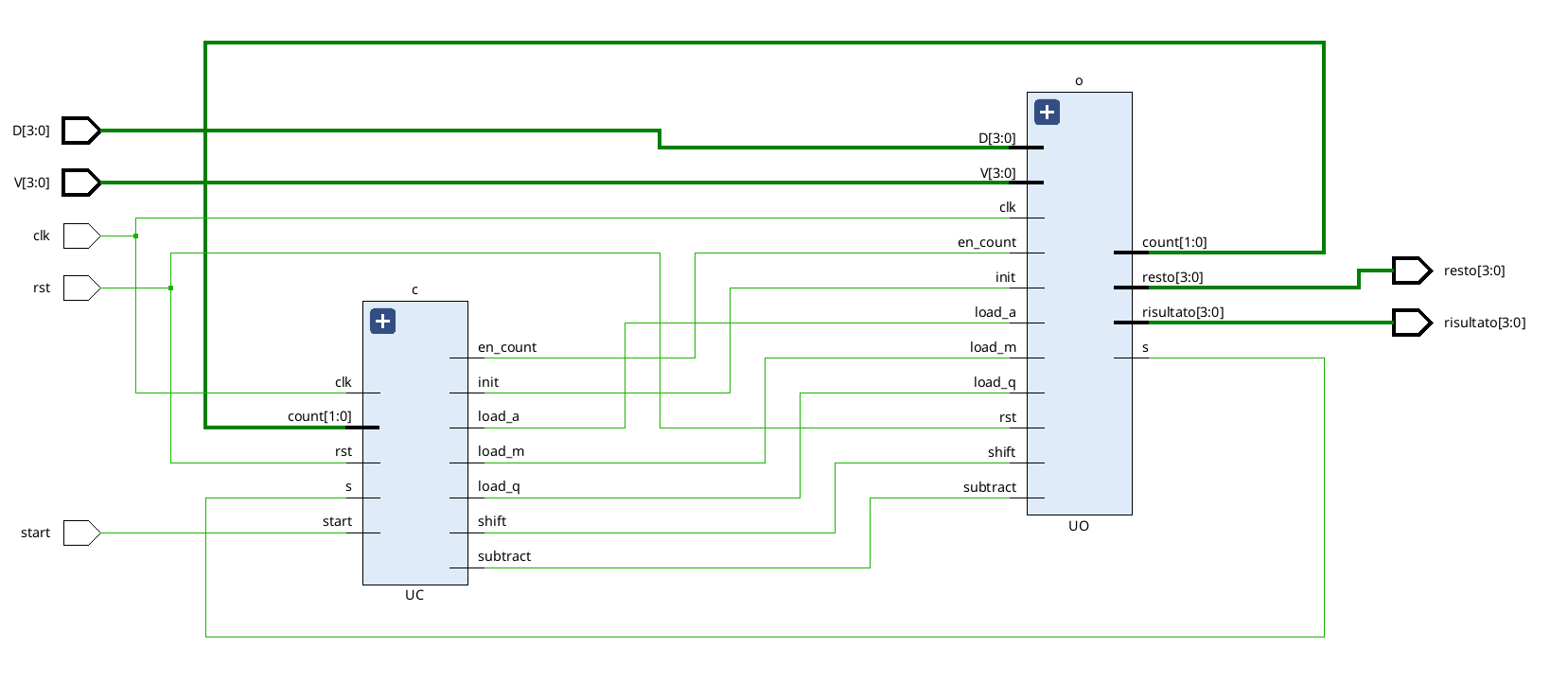
* Contatore Modulo 4.
* Un registro M per salvare il divisore.
* Uno Shift Register Q di 4 bit per salvare inizialmente il dividendo e poi il risultato.
* Un banco di multiplexer per selezionare l’ingresso di Q: il divisore in fase di inizializzazione, e i risultati parziali della divisione.
* Uno Shift Register A di 5 bit, l’ultimo per salvare il segno dell’operazione in corso, per salvare i passi intermedi dell’algoritmo e in fine il resto.
* Due banchi di Mux 2\_1 i quali si occupano di inizializzare il registro o inserire al suo interno il risultato in uscita dal sommatore sottrattore.
* Un sommatore/sottrattore.



L’Unità di Controllo ha il seguente automa :



Quindi il sistema complessivo



#### Implementazione

Segue il codice dell’unità Operativa

Unità di Controllo

Top Module Divsiore

#### Simulazione

Il divisore funziona su numeri interi positivi e anche per il Divisore negativo.

Testbench:

Risultato:



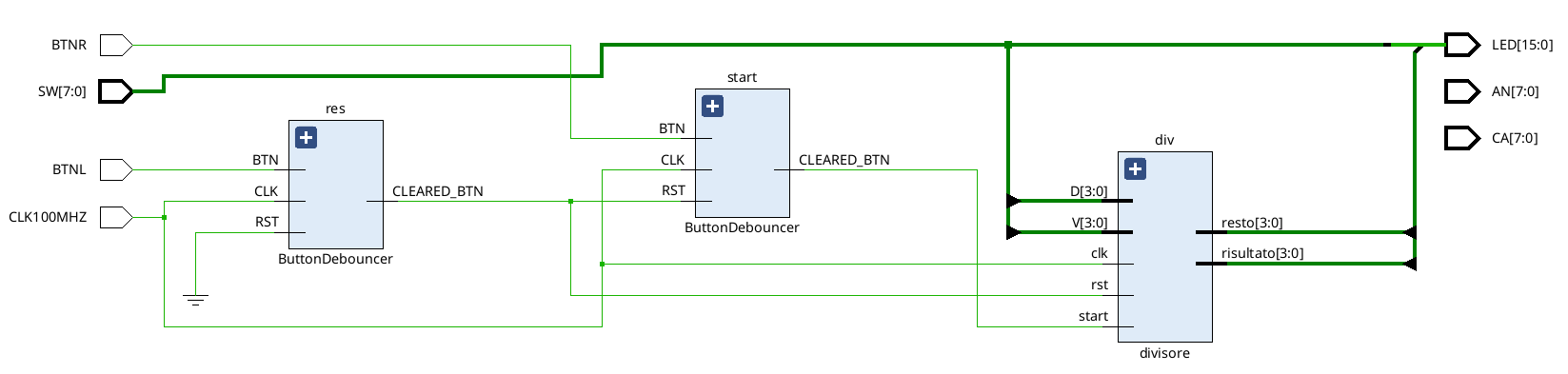
### Esercizio 7BIS.2

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

Per utilizzare il componente su board è stato previsto di utilizzare un bottone per il segnale di start, uno per il segnale di reset, gli switch per l’immissione degli operandi. Inoltre sono stati utilizzati i Led per visualizzare i dati immessi, il risultato e l’eventuale resto.

Schematics:



Codice:

Constraint:

# **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

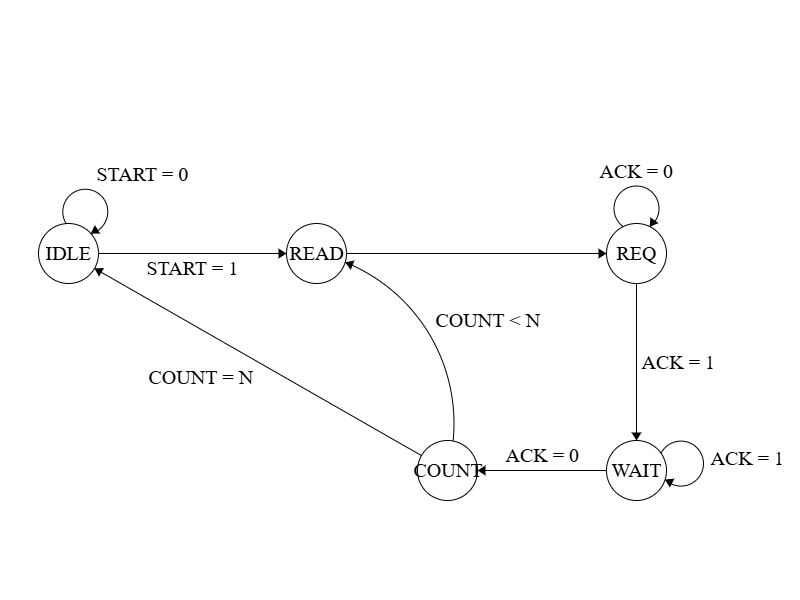
##### Sistema A

Il sistema A, come si evince dalla descrizione, è composto da:

* un **contatore modulo N**;
* una **memoria** **ROM**;
* un’**unità di controllo**.

La UC gestisce opportunamente il contatore, che fornisce l’indirizzo a cui accedere in ROM. Il dato prelevato viene inviato al nodo B mediante handshaking.

L’unità di controllo implementa il seguente automa a stati finiti:



/EN\_COUNT = 1

/READ\_ROM = 1

/EN\_COUNT = 0

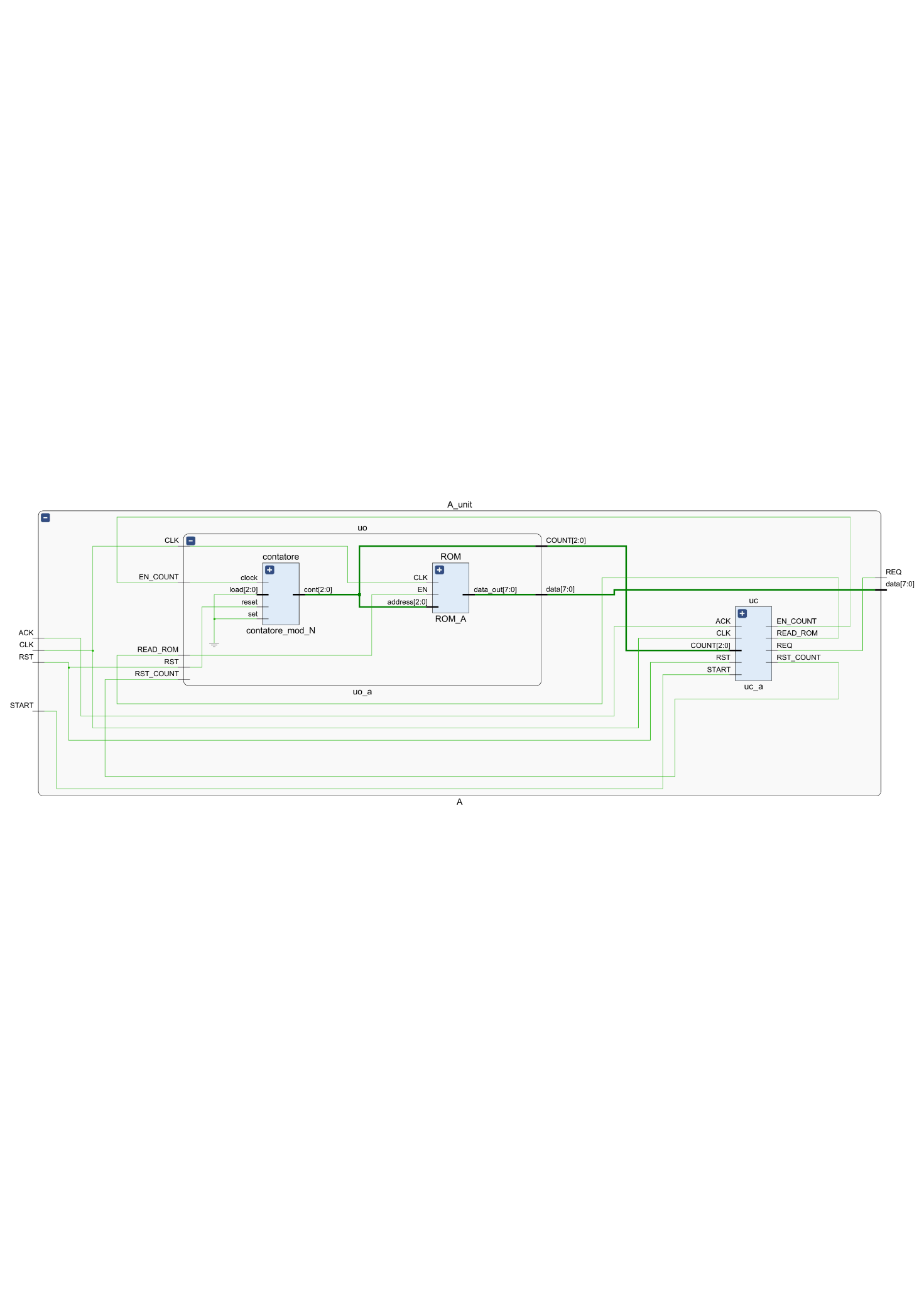
/EN\_COUNT = 0

/REQ = 1

/REQ = 0

/READ\_ROM = 0

Il nodo, invece, appare come segue:



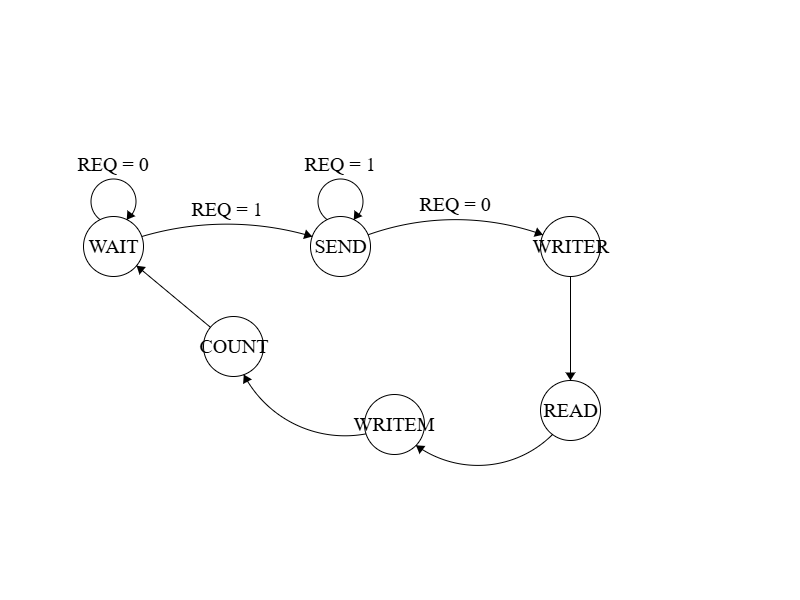
##### Sistema B

Il sistema B contiene:

* un **contatore modulo N**;
* una **memoria** **ROM**;
* una **memoria in scrittura** **MEM**;
* un **Ripple Carry Adder** parallelo;
* un **registro**;
* un’**unità di controllo**.

Il nodo riceve un dato da A e lo salva in un registro, successivamente la UC abilita il contatore per fornire l’indirizzo per l’accesso al valore in ROM, e infine allo stesso indirizzo viene effettuata una scrittura in MEM della somma tra il valore X ricevuto da A e il valore Y prelevato dalla ROM. La somma è ottenuta istantaneamente perché il RCA è puramente combinatorio.

L’unità di controllo implementa il seguente automa a stati finiti:



/READ\_ROM = 0

/EN\_COUNT = 1

/WRITE\_MEM = 1

/ACK = 0

/WRITE\_REG = 0

/READ\_ROM = 1

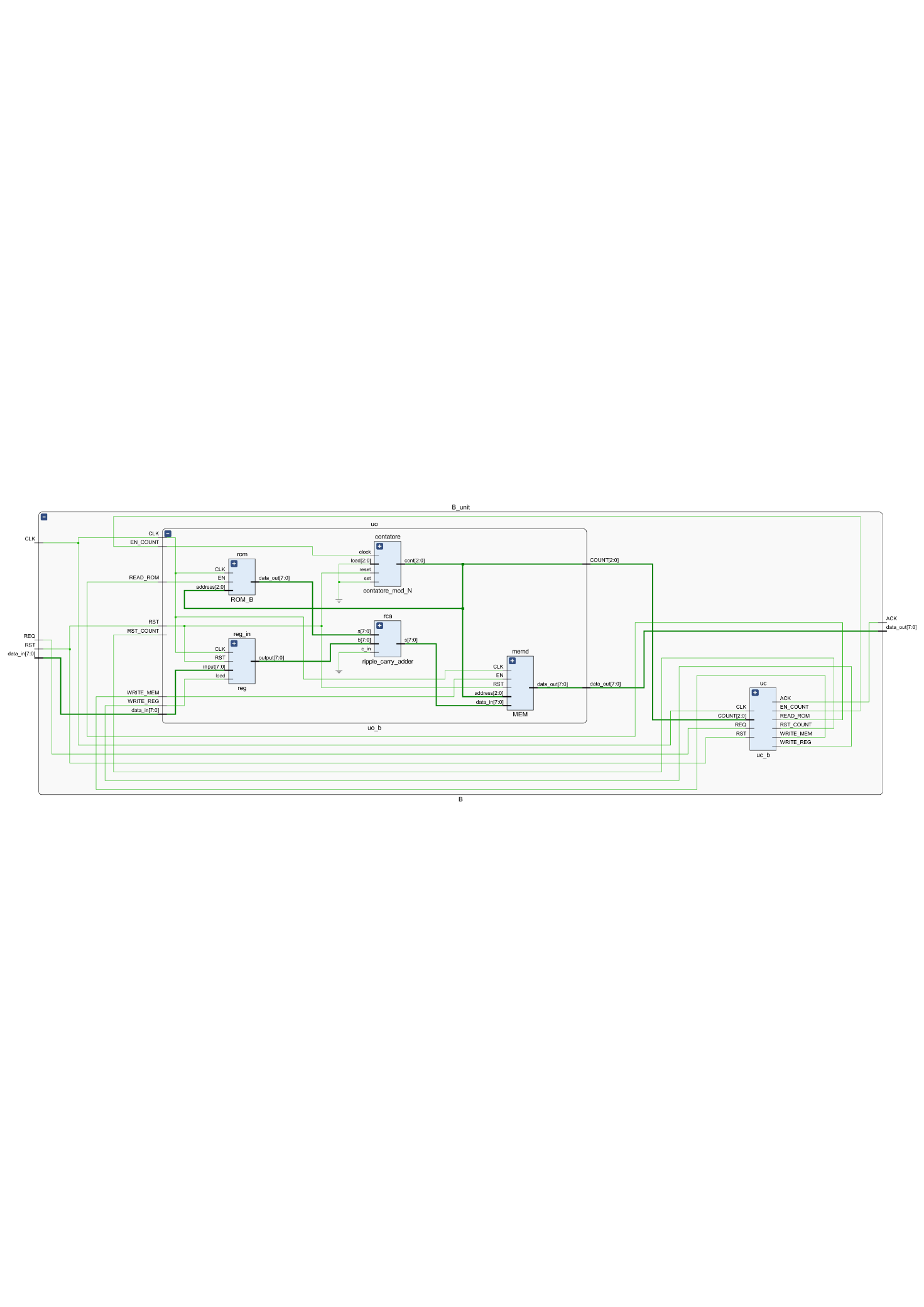
/EN\_COUNT = 0

/ACK = 1

/WRITE\_MEM = 0

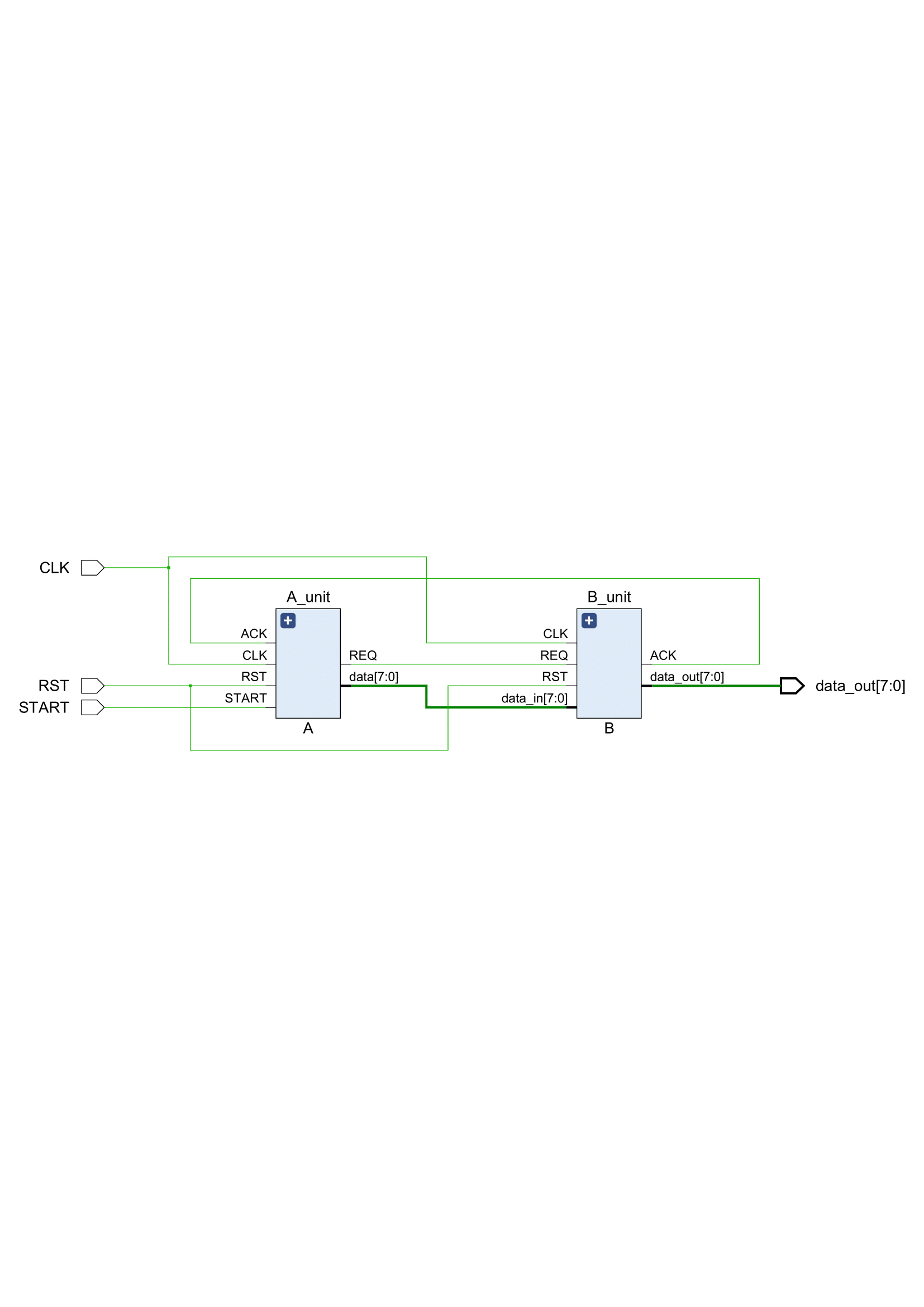
/WRITE\_REG = 1

In definitiva, il nodo è il seguente:



##### Sistema complessivo

Il Top-Module appare come segue:



Dunque, le unità A e B comunicano mediante handshaking attraverso due segnali REQ e ACK. A invia una REQ non appena il dato è pronto, B alza una ACK per comunicare al nodo A che ha preso in carico la richiesta, e riabbasserà il segnale non appena termina di utilizzare il dato.

#### Implementazione

##### Sistema A

Il contatore è descritto in appendice. La memoria ROM, pressoché simile a quanto già visto [nell’esercizio 6.1](#_Esercizio_6.1), è stata implementata in questo modo, aggiungendo un parametro M nel Generic:

entity ROM\_A is

Generic(N : natural := 8;

M : natural := 8);

Port ( CLK : in std\_logic;

EN : in std\_logic;

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

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end ROM\_A;

architecture Behavioral of ROM\_A is

type MEMORY\_N\_M is array (0 to N-1) of std\_logic\_vector(M-1 downto 0); -- Matrice N x M

constant ROM\_DATA : MEMORY\_N\_M := ( -- ROM content

"00000001",

"00000010",

"00000011",

"00000100",

"00000101",

"00000110",

"00000111",

"00001000"

);

begin

process(CLK)

begin

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

if(EN = '1') then

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

end if;

end if;

end process;

end architecture Behavioral;

L’unità di controllo è la seguente:

entity uc\_a is

Generic(N:natural:=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

START: in std\_logic;

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

ACK : in std\_logic;

REQ : out std\_logic;

EN\_COUNT: out std\_logic;

RST\_COUNT: out std\_logic;

READ\_ROM: out std\_logic

);

end uc\_a;

architecture Behavioral of uc\_a is

type stato is (IDLE,READ,REQ\_STATE,WAIT\_ACK,COUNT\_STATE);

signal stato\_corrente: stato:=IDLE;

signal stato\_prossimo: stato;

begin

comb:process(stato\_corrente,START,ACK,COUNT)

begin

case stato\_corrente is

when IDLE =>

en\_count<='0';

if(START='1') then

stato\_prossimo<=READ;

else

stato\_prossimo<=IDLE;

end if;

when READ =>

en\_count<='0';

READ\_ROM<='1';

stato\_prossimo<=REQ\_STATE;

when REQ\_STATE =>

REQ<='1';

if(ACK='1') then

stato\_prossimo<=WAIT\_ACK;

else

stato\_prossimo<=REQ\_STATE;

end if;

when WAIT\_ACK =>

REQ<='0';

if(ACK='0') then

stato\_prossimo<=COUNT\_STATE;

else

stato\_prossimo<=WAIT\_ACK;

end if;

when COUNT\_STATE =>

READ\_ROM<='0';

en\_count<='1';

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

stato\_prossimo<=IDLE;

else

stato\_prossimo<=READ;

end if;

end case;

end process;

mem:process(CLK)

begin

if(rising\_edge(CLK)) then

if(RST='1') then

RST\_COUNT<='0';

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

##### Sistema B

Per il sistema B, il contatore resta implementato come descritto in appendice e la ROM è identica, se non per il nome dell’entity, che è chiaramente ROM\_B. Anche il contenuto della memoria è analogo:

entity ROM\_B is

Generic(N : natural := 8;

M : natural := 8);

Port ( CLK : in std\_logic;

EN : in std\_logic;

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

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end ROM\_B;

architecture Behavioral of ROM\_B is

type MEMORY\_N\_M is array (0 to N-1) of std\_logic\_vector(M-1 downto 0); -- Matrice N x M

constant ROM\_DATA : MEMORY\_N\_M := ( -- ROM content

"00000001",

"00000010",

"00000011",

"00000100",

"00000101",

"00000110",

"00000111",

"00001000"

);

begin

process(CLK)

begin

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

if(EN = '1') then

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

end if;

end if;

end process;

end architecture Behavioral;

Il RCA e il registro per salvare il dato ricevuto dal nodo A sono implementati in maniera analoga ai componenti utilizzati [nell’esercizio 7.1](#_Implementazione_2). In particolare, è stato usato l’RCA semplice (senza modulo Adder-Subtractor) per la somma, e il registro che nel moltiplicatore salvava Y è stato riutilizzato per salvare la stringa ricevuta da A.

La memoria MEM è stata implementata in questo modo:

entity MEM is

Generic(N : natural :=8;

M : natural :=8);

Port ( CLK : in std\_logic;

RST : in std\_logic;

EN : in std\_logic;

data\_in : in std\_logic\_vector(M-1 downto 0);

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

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end MEM;

architecture Behavioral of MEM is

type MEMORY\_N\_M is array (0 to N-1) of std\_logic\_vector(M-1 downto 0); -- Matrice N x M

signal MEM\_DATA : MEMORY\_N\_M := (others => "00000000");

begin

process(CLK)

begin

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

if(EN = '1') then

MEM\_DATA(to\_integer(unsigned(address))) <= data\_in; -- Casting da unsigned a intero

elsif(RST = '1') then

MEM\_DATA <= (others => "00000000");

end if;

end if;

end process;

data\_out <= MEM\_DATA(to\_integer(unsigned(address)));

end architecture Behavioral;

L’unità di controllo, infine, è la seguente:

entity uc\_b is

Generic(N:natural:=8 );

Port ( CLK: in std\_logic;

RST: in std\_logic;

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

REQ: in std\_logic;

EN\_COUNT: out std\_logic;

RST\_COUNT: out std\_logic;

READ\_ROM: out std\_logic;

WRITE\_MEM: out std\_logic;

WRITE\_REG: out std\_logic;

ACK: out std\_logic

);

end uc\_b;

architecture Behavioral of uc\_b is

type stato is (WAIT\_REQ,SEND\_ACK ,WRITE\_REG\_STATE ,READ,WRITE\_MEM\_STATE,COUNT\_STATE);

signal stato\_corrente: stato:= WAIT\_REQ;

signal stato\_prossimo: stato;

begin

comb:process(stato\_corrente,COUNT,REQ)

begin

case stato\_corrente is

when WAIT\_REQ =>

en\_count<='0';

if(REQ='1') then

stato\_prossimo<=SEND\_ACK;

else

stato\_prossimo<=WAIT\_REQ;

end if;

when SEND\_ACK =>

ACK<='1';

if(req='0') then

stato\_prossimo<= WRITE\_REG\_STATE;

else

stato\_prossimo<=SEND\_ACK;

end if;

when WRITE\_REG\_STATE =>

WRITE\_REG<='1';

stato\_prossimo<=READ;

when READ =>

ACK<='0';

WRITE\_REG<='0';

READ\_ROM<='1';

stato\_prossimo<=WRITE\_MEM\_STATE;

when WRITE\_MEM\_STATE =>

WRITE\_MEM<='1';

stato\_prossimo<=COUNT\_STATE;

when COUNT\_STATE =>

WRITE\_MEM<='0';

READ\_ROM<='0';

en\_count<='1';

stato\_prossimo<=WAIT\_REQ;

end case;

end process;

mem:process(CLK)

begin

if(rising\_edge(CLK)) then

if(RST='1') then

RST\_COUNT<='0';

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

##### Sistema complessivo

Il Top Module in VHDL (approccio **Structural**) è:

entity system is

Generic(N:natural:=8;

M:natural:=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

START: in std\_logic;

data\_out: out std\_logic\_vector(M-1 downto 0)

);

end system;

architecture Structural of system is

component A is

Generic(N:natural:=8;

M:natural:=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

START: in std\_logic;

ACK: in std\_logic;

REQ: out std\_logic;

data: out std\_logic\_vector(M-1 downto 0) );

end component;

component B is

Generic(N:natural:=8;

M:natural:=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

REQ: in std\_logic;

ACK: out std\_logic;

data\_in: in std\_logic\_vector(M-1 downto 0);

data\_out: out std\_logic\_vector(M-1 downto 0)

);

end component;

signal ack:std\_logic;

signal req:std\_logic;

signal data:std\_logic\_vector(M-1 downto 0);

begin

A\_unit: A

Generic Map(N=>N,

M=>M)

Port Map( CLK=>CLK,

RST=>RST,

START=>START,

ACK=>ack,

REQ=>req,

data=>data

);

B\_unit: B

Generic Map(N=>N,

M=>M)

Port Map( CLK=>CLK,

RST=>RST,

REQ=>req,

ACK=>ack,

data\_in=>data,

data\_out=>data\_out

);

end Structural;

#### Simulazione

La testbench in questione si occupa semplicemente di visualizzare il comportamento del sistema a fronte di un segnale di START alto.

entity a\_b\_tb is

Generic(N:natural:=8;

M:natural:=8);

end a\_b\_tb;

architecture Behavioral of a\_b\_tb is

component system is

Generic(N:natural:=8;

M:natural:=8);

Port ( CLK: in std\_logic;

RST: in std\_logic;

START: in std\_logic;

data\_out: out std\_logic\_vector(M-1 downto 0)

);

end component;

signal clock: std\_logic;

signal rst: std\_logic;

signal start: std\_logic;

signal output: std\_logic\_vector(M-1 downto 0);

signal period: time := 10 ns;

begin

uut: system

Generic Map ( N=>N,

M=>M)

Port Map( CLK=>clock,

RST=>rst,

START=>start,

data\_out=>output

);

clock\_proc: process

begin

clock<='0';

wait for period/2;

clock<='1';

wait for period/2;

end process;

stim:process

begin

start<='1';

rst<='0';

wait for 10 ns;

start<='0';

wait;

end process;

end Behavioral;

Il risultato complessivo dimostra il corretto funzionamento del prelievo dei dati e delle somme:

Immagine che contiene schermata, linea, software, Software multimediale

Descrizione generata automaticamenteImmagine che contiene schermata, linea

Descrizione generata automaticamente

# **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.

#### MIC-1

Il MIC-1 è un’implementazione, realizzata da Tanenbaum, della Integer Java Virtual Machine (**IJVM**), sottoinsieme della Java Virtual Machine. È una microarchitettura basata su stack.

Possiamo suddividere l’analisi della microarchitettura in:

* analisi del **Datapath**;
* analisi della **Control Unit**.

##### Datapath

L’architettura del MIC-1 è composta da 10 registri a 32 bit, 2 bus, 1 ALU e 1 shift register.

Immagine che contiene diagramma, schizzo, testo, Parallelo

Descrizione generata automaticamente

* **ALU:** componenteche si occupa di effettuare operazioni a 2 operandi**.** Utilizza 6 linee di controllo: F0, F1 codificano l’operazione; ENA, ENB abilitano gli input; INVA inverte l’ingresso di A; INC consente l’incremento di un’unità. La ALU possiede 3 uscite: il risultato dell’operazione, che sarà l’input dello shift register, e i bit N e Z per segnalare il risultato negativo o nullo dell’operazione. Nel momento in cui si vuole effettuare un’operazione, il primo operando viene trasmesso tramite il bus B. Tramite la configurazione di controllo, che dà in uscita il dato sul bus B, viene posizionato nel registro H. Fatto ciò, il secondo operando viene mandato nuovamente tramite il bus B, così l’ALU è pronta a effettuare l‘operazione.
* **MAR:** Memory Address Register, contiene l’indirizzo della word in lettura o scrittura. È indirizzato a word di 32 bit;
* **MDR:** Memory Data Register, contiene il valore effettivo della word. È indirizzato a word di 32 bit;
* **PC:** Program Counter, contiene l’indirizzo dell’istruzione attuale in esecuzione;
* **MBR:** Memory Byte Register, memorizza i byte dell’istruzione man mano che viene interpretata. È indirizzato a byte;
* **SP:** Stack Pointer, punta alla cima dello stack;
* **LV:** Local Variable, punta alla prima variabile del Local Variable Frame, la porzione di stack che contiene le variabili del metodo in esecuzione;
* **CPP:** Constant Pool Pointer, puntatore alla base dell’area di memoria Constant Pool;
* **TOS:** Top Of Stack, contiene una copia del dato puntato dallo Stack Pointer;
* **OPC:** Old Program Counter, memorizza l’istruzione precedente del Program Counter;
* **H:** Holding,contiene uno dei due operandi per l’ALU, generalmente contiene il primo operando che arriva sul bus B.

MAR e MDR controllano un porto a 32 bit per comunicare con la memoria, mentre PC e MBR fanno lo stesso su 8 bit.

Nota: i registri scrivono i dati sul bus B e li leggono dal bus C. Difatti, C è semplicemente l’uscita dell’ALU, e il bus non può essere bidirezionale.

##### Control Unit

La Control Unit del MIC-1 è realizzata secondo un approccio **micro-programmato**, ovvero le microistruzioni che esegue sono memorizzate in una ROM, la **Control Store**.

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

Ogni microistruzione contiene i segnali di controllo per i bus B e C, per l’ALU, per la memoria, per le istruzioni di salto e l’indirizzo della prima microistruzione. Tale indirizzo viene caricato nel micro Program Counter, che serve ad accedere alla giusta microistruzione nella Control Store.

L’istruzione puntata da MPC viene caricata nel micro Instruction Register.

Immagine che contiene testo, Carattere, linea, numero

Descrizione generata automaticamente

In scrittura i registri vengono attivati direttamente, mentre in lettura viene utilizzato un decoder 4:16 per l’abilitazione.

L’unità di controllo prevede, infine, due ulteriori componenti per la gestione dei salti:

* **Jump Logic**, che, in base al valore di JMPC, determina se scrivere nel registro MPC l’indirizzo contenuto nella sezione address della microistruzione oppure fare *bit or-ing* tra il vettore address e il vettore MB;
* **High Bit**, che contiene le condizioni di salto, che cambiano in base ai lavori di JAMN e JAMZ (salto per uguaglianza o disuguaglianza). Tale componente, infatti, è in grado di modificare il bit Zero del Program Counter.

#### Analisi delle istruzioni

Tutte le istruzioni sono codificate in MAL (**Micro Assembly Language**).

##### IADD

L’istruzione IADD (Immediate ADD) è l’operazione algebrica di somma. È immediata perché non calcola la somma degli operandi in input alla ALU, ma sfrutta il valore attuale dello stack e quello immediatamente precedente.

Ciò viene fatto sfruttando il TOS, che contiene il valore immediatamente precedente dello stack, e lo SP, che contiene quello attuale.

Per prima cosa lo Stack Pointer viene decrementato di 1, in modo che il valore sottostante alla cima dello stack diventi il nuovo valore puntato dallo SP.

Lo SP diventa, successivamente, l’indirizzo contenuto in MAR. Nel mentre, il valore contenuto in TOS sarà ancora pari al valore puntato in precedenza dallo SP. Tale valore sarà il primo operando, e viene salvato nel registro H.

Infine, è possibile eseguire la somma tra H e il secondo operando, contenuto in MDR perché puntato dal nuovo SP. In tal modo, il registro H memorizzerà la somma dei valori, e la somma diventerà il nuovo TOS. L’ultima operazione da compiere è l’aggiornamento del contenuto di MDR, che correttamente avrà il valore puntato dal MAR “SP”, cioè TOS.

IADD = 0x65 :

MAR = SP = SP-1; rd

H = TOS

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

La simulazione mediante GTK-WAVE dell’istruzione ha prodotto i seguenti risultati:

Immagine che contiene schermata, testo, software, Software multimediale

Descrizione generata automaticamente

L’andamento dell’istruzione è quello atteso.

In corrispondenza della barra rossa avviene il fetch della microistruzione all’indirizzo 0x65 della control store. Quindi, al prossimo colpo di clock saranno caricate le micro-operazioni della IADD.

##### BIPUSH

L’istruzione BIPUSH (Byte Immediate Push) consente di inserire un byte in testa allo stack.

La prima cosa da fare per predisporre l’operazione è incrementare lo SP, salvarlo nel MAR e aggiornare lo SP effettivo alla nuova posizione.

Successivamente, bisogna incrementare il PC ed eseguire la fetch del prossimo OPCODE. Bisogna, infatti, leggere l’operando all’interno della prossima istruzione al fine di salvarlo in cima allo stack.

Il MBR contiene ora il prossimo operando, in seguito al fetch della prossima istruzione. Tale dato viene posizionato come TOS e da ciò segue l’aggiornamento del MDR, coerentemente al fatto che MAR contiene il nuovo stack pointer.

BIPUSH = 0x10 :

SP = MAR = SP + 1

PC = PC + 1 ; FETCH

MDR = TOS = MBR; wr ; goto main

La simulazione dell’istruzione ha prodotto i seguenti risultati:

Immagine che contiene schermata, testo, software, Software multimediale

Descrizione generata automaticamente

Anche in questo caso il comportamento della microistruzione è analogo.

Anche in questo caso, la barra rossa indica, il *fetching* dell’istruzione. Ciò consente di caricare nel registro MBR il codice operativo che accede alla parola del control store.

In tal modo verrà avviata la prima microistruzione della micro-procedura relativa a BIPUSH. Chiaramente, affinché l’istruzione venga resa disponibile, occorre attendere un ciclo di clock.

#### Nuova istruzione

##### THREEAND

L’istruzione scelta è una AND a tre operandi: .

Per prima cosa si decrementa lo SP, facendolo puntare così all’operando B.

In seguito, assegno TOS a H, ovvero H conterrà l’operando A. Infatti, prima del decremento, l’operando A si trovava in cima allo stack.

Si esegue la prima AND tra H e il MDR, che è puntato da MAR e che dunque punta a B, e la si salva in H.

Ciò che si dovrà fare in seguito, dunque, è semplicemente decrementare nuovamente lo SP per ottenere l’operando C, effettuare la AND tra il nuovo MDR e H e salvarla nel MDR.

Tra l’aggiornamento di SP e l’operazione di AND è presente una NO OPERATION per far sì che l’operando C sia effettivamente disponibile per l’operazione.

Quando si decrementa la seconda volta, si è ottenuto il terzo operando, ricordando il modo in cui era strutturato lo SP prima di tutti i decrementi. Se H contiene già una AND, e si effettua una AND tra tale stringa e il nuovo operando, la AND a tre operandi è andata a buon fine.

THREEAND = 0x6A :

MAR = SP = SP-1; rd

H = TOS

H = H AND MDR

MAR = SP = SP -1; rd

NO OPERATION

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

# **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.

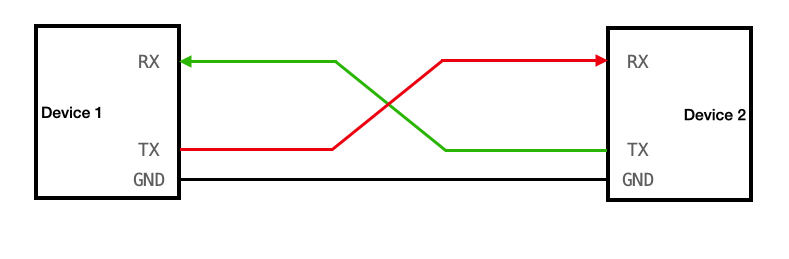
#### Progetto e architettura

La **UART** (Universal Asynchronous Receiver-Transmitter) è un dispositivo che permette la comunicazione tra due sistemi inviando serialmente dati a gruppi di 8 bit. Il meccanismo utilizzato è asincrono, per cui non vi è una reale sincronizzazione tra i clock dei due sistemi, ma avviene una gestione di flusso e degli errori mediante lo scambio di bit quali bit di parità, bit di start e bit di stop.

Tuttavia, ciò che devono fare i due nodi che hanno intenzione di comunicare con la UART è sincronizzarsi sul **baud rate**, ovvero sulla quantità di bit trasmessi in un secondo. Si può dire che:

Il baud è una misura legata alla banda del canale e allo scambio di segnali, ma è facile notare un parallelismo con i bit per secondo che vengono scambiati sul canale. Il valore più comune di baud rate, nonché quello utilizzato nel progetto, è di 9600 Baud.

L’interfaccia UART consiste di 2 segnali TX e RX su cui viaggiano i dati trasmessi/ricevuti, più un segnale di GROUND.



Lo stato di IDLE della seriale è identificato da un valore logico alto; non appena il valore logico diventa basso, il sistema ricevitore comprende che sta per avere inizio la comunicazione. Il bit 0, dunque, funge da **start bit**. In seguito, avviene lo scambio effettivo dei bit, che nel caso in analisi sono 8 (e che in ogni caso va pre-concordato tra i sistemi). Infine, segue un bit di parità (opzionale) e 1 o 2 bit di stop, identificati dal valore logico alto, per terminare la trasmissione.

Il bit di parità è 0 se il numero di caratteri è pari, 1 se è dispari. Se Rx riceve, ad esempio, 7 caratteri, ma il bit di parità è 0, intuisce che qualche bit è andato perduto.

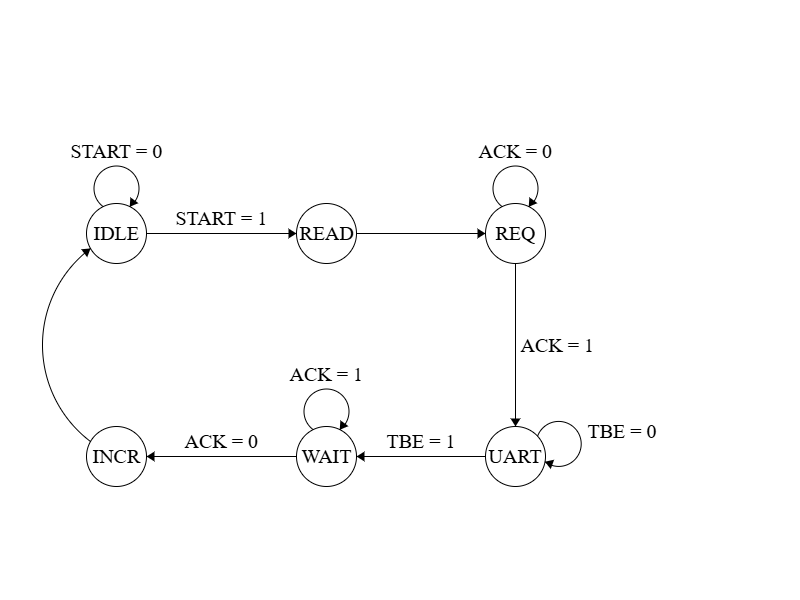
L’invio seriale dei bit viene eseguito mediante un registro a scorrimento. La sezione di trasmissione della UART fornisce un segnale di interruzione **TBE** (Transfer Bus Empty) che segnala quando è stato caricato un dato in tale registro. È un segnale di fondamentale importanza per realizzare l’handshake tra i due sistemi. La sezione di ricezione, invece, fornisce un segnale di interruzione **RDA** (Read Data Available) che segnala che un frame sia stato completamente ricevuto.

##### Sistema A

Il nodo A possiede:

* un **contatore modulo N**;
* una **memoria** **ROM**;
* una **UART**;
* un’**unità di controllo**.

L’unità di controllo implementa il seguente automa a stati finiti:



/EN\_COUNT = 1

/READ\_ROM = 1

/EN\_COUNT = 0

/WR = 0

/WR = 0

/REQ = 0

/REQ = 1

/WR = 1

Il nodo, invece, appare come segue:

Immagine che contiene testo, schermata, diagramma, linea

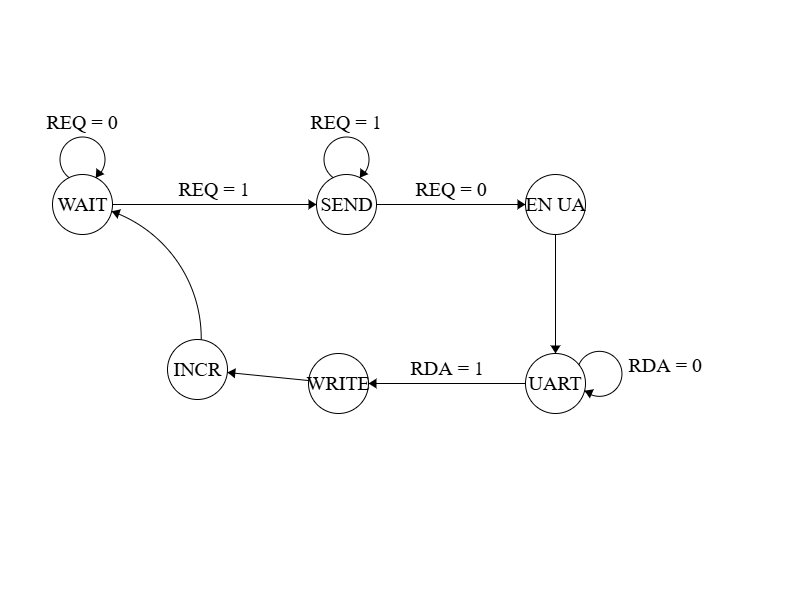
Descrizione generata automaticamente

##### Sistema B

Il nodo A possiede:

* un **contatore modulo N**;
* una **memoria in scrittura MEM**;
* una **UART**;
* un’**unità di controllo**.

L’unità di controllo implementa il seguente automa a stati finiti:



/EN\_COUNT = 1

/WRITE\_MEM = 0

/WRITE\_MEM = 1

/RD = 0

/ACK = 0

/ACK = 1

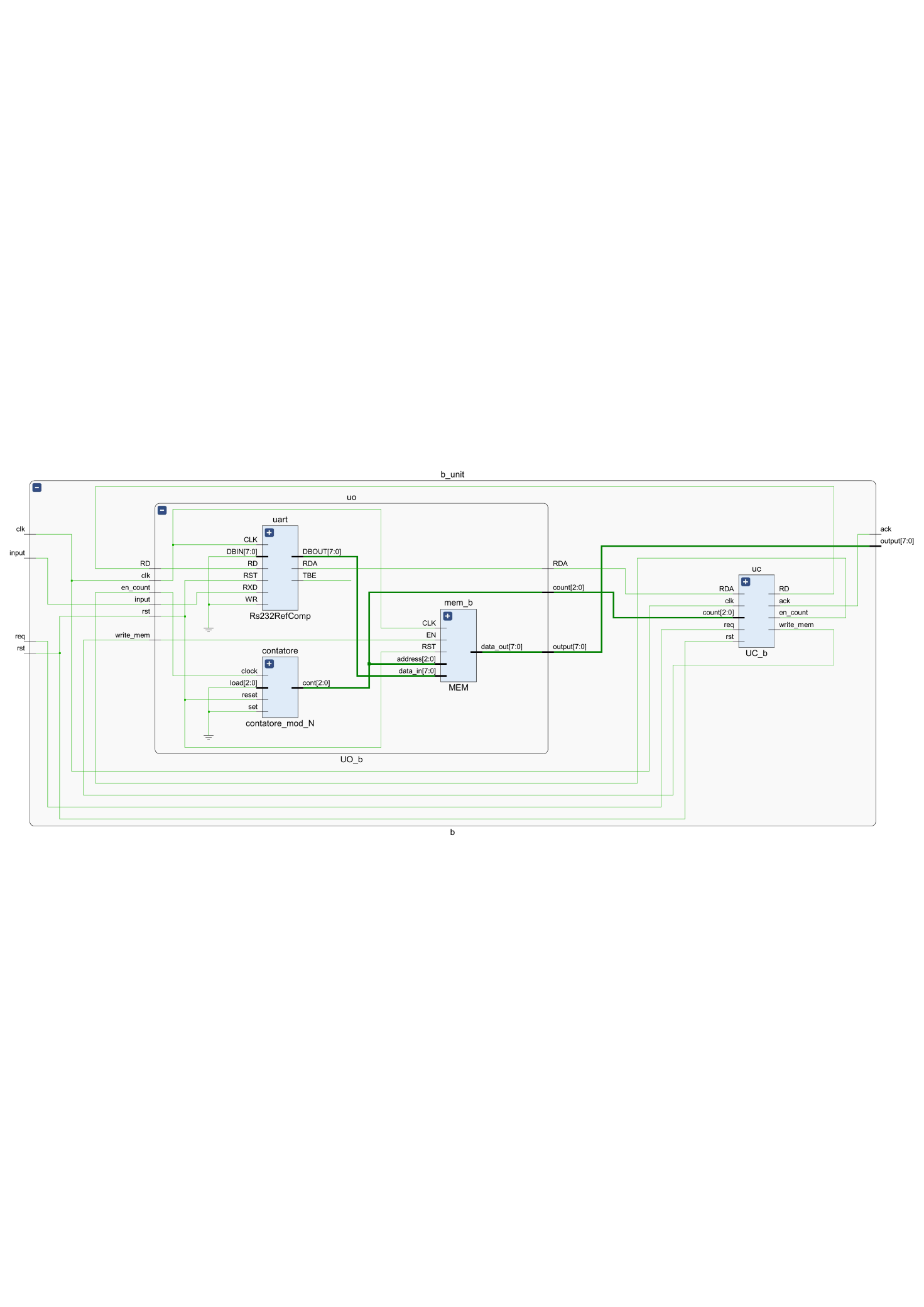
/RD = 0

/RD = 1

/RD = 0

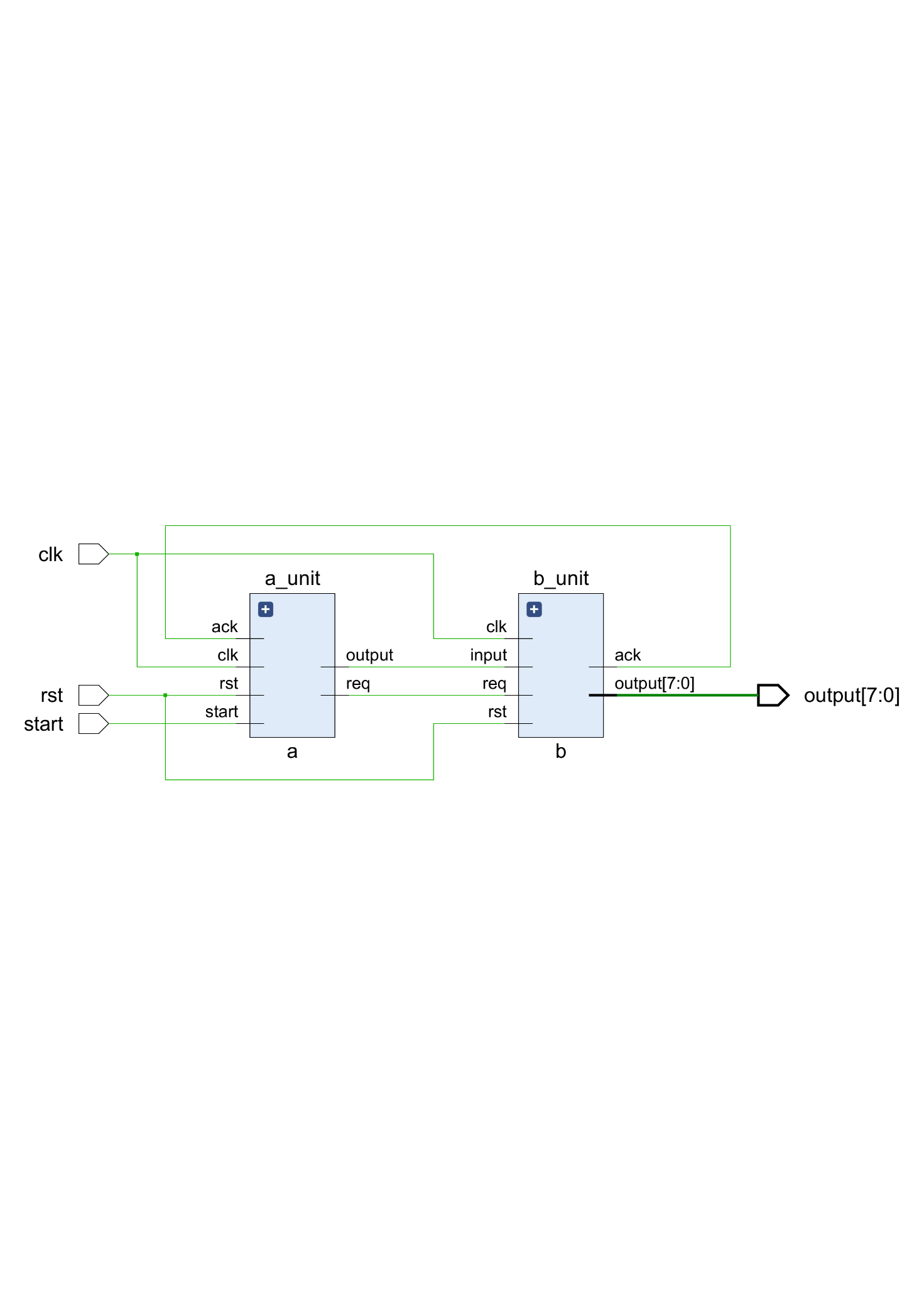
/EN\_COUNT = 0

Il nodo, infine, appare come segue:



##### Sistema complessivo

Il sistema complessivo, oltre i dati, condivide dei segnali di REQ e ACK per accordarsi, mediante handshaking, sull’invio seriale del byte:



#### Implementazione

##### Sistema A

La parte operativa del nodo A possiede un contatore modulo N, approfondito [nell’appendice](#_Contatore_modulo_N), e una memoria ROM di contenuto e struttura analoghe all’implementazione [dell’esercizio 8.1](#_Sistema_A).

Il nodo connette semplicemente i tre componenti:

entity UO\_a is

Generic ( N : natural :=8;

M : natural := 8);

Port ( clk: in std\_logic;

rst: in std\_logic;

en\_count: in std\_logic;

read\_rom: in std\_logic;

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

TBE : out std\_logic;

WR : in std\_logic;

output: out std\_logic

);

end UO\_a;

architecture Structural of UO\_a is

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 ROM\_A is

Generic(N : natural := 8;

M : natural := 8);

Port ( CLK : in std\_logic;

EN : in std\_logic;

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

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end component;

component Rs232RefComp is

Port (

TXD : out std\_logic := '1';

RXD : in std\_logic;

CLK : in std\_logic; --Master Clock

DBIN : in std\_logic\_vector (7 downto 0); --Data Bus in

DBOUT : out std\_logic\_vector (7 downto 0); -- Data Bus out

RDA : inout std\_logic; --Read Data Available(1 quando il dato è disponibile nel registro rdReg)

TBE : inout std\_logic := '1';--Transfer Bus Empty(1 quando il dato da inviare è stato caricato nello shift register)

RD : in std\_logic; --Read Strobe(se 1 significa "leggi" --> fa abbassare RDA)

WR : in std\_logic; --Write Strobe(se 1 significa "scrivi" --> fa abbassare TBE)

PE : out std\_logic; --Parity Error Flag

FE : out std\_logic; --Frame Error Flag

OE : out std\_logic; --Overwrite Error Flag

RST : in std\_logic := '0'); --Master Reset

end component;

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

signal rom\_to\_uart: std\_logic\_vector(M-1 downto 0);

signal TBE\_bus: std\_logic;

begin

contatore :contatore\_mod\_N

Generic Map(N=>N)

Port Map( clock=>en\_count,

reset=>rst,

set=>'0',

load=>(others=>'0'),

cont=>address

);

count<=address;

rom: ROM\_A

Generic Map(N=>N,

M=>M)

Port Map( CLK=>clk,

EN=>read\_rom,

address=>address,

data\_out=>rom\_to\_uart

);

-- TBE per dire quando il dato è stato messo nello sr uart

uart:Rs232RefComp

Port Map(

TXD =>output, --

RXD =>'0',

CLK =>clk, --

DBIN =>rom\_to\_uart, --

--DBOUT =>,

--RDA =>,

TBE =>TBE\_bus,--

RD =>'0',

WR =>WR,

RST =>rst

);

TBE<=TBE\_bus;

end Structural;

Unità di controllo:

entity UC\_a is

Generic ( N : natural :=8;

M : natural := 8);

Port ( clk: in std\_logic;

rst: in std\_logic;

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

start : in std\_logic;

ack: in std\_logic;

req: out std\_logic;

en\_count: out std\_logic;

read\_rom: out std\_logic;

TBE : in std\_logic;

WR : out std\_logic

);

end UC\_a;

architecture Behavioral of UC\_a is

type state is (idle,read,req\_state,uart\_ready,wait\_ack,increment);

signal stato\_corrente: state:=idle;

signal stato\_prossimo: state;

begin

comb:process(stato\_corrente,start,count,ack,TBE)

begin

case stato\_corrente is

when idle=>

en\_count<='0';

if(start='1') then

stato\_prossimo<=read;

else

stato\_prossimo<=idle;

end if;

when read =>

read\_rom<='1';

WR<='1';

stato\_prossimo<=req\_state;

when req\_state =>

req<='1';

if(ack='1') then

stato\_prossimo<=uart\_ready;

else

stato\_prossimo<=req\_state;

end if;

when uart\_ready =>

req<='0';

wr<='0';

if(TBE='1') then

stato\_prossimo<=wait\_ack;

else

stato\_prossimo<=uart\_ready;

end if;

when wait\_ack =>

WR<='0';

if(ack<='0') then

stato\_prossimo<=increment;

else

stato\_prossimo<=wait\_ack;

end if;

when increment =>

en\_count<='1';

stato\_prossimo<=idle;

end case;

end process;

mem:process(clk)

begin

if(rising\_edge(clk)) then

if(rst='1') then

stato\_corrente<=idle;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

##### Sistema B

La parte operativa del nodo A possiede un contatore modulo N, approfondito [nell’appendice](#_Contatore_modulo_N), e una memoria MEM di struttura analoga all’implementazione [dell’esercizio 8.1](#_Sistema_A).

Il nodo connette semplicemente i tre componenti:

entity UO\_b is

Generic ( N : natural :=8;

M : natural := 8);

Port ( clk: in std\_logic;

rst: in std\_logic;

en\_count: in std\_logic;

write\_mem: in std\_logic;

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

input: in std\_logic;

RDA: out std\_logic;

RD: in std\_logic;

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

);

end UO\_b;

architecture Structural of UO\_b is

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 MEM is

Generic(N : natural :=8;

M : natural :=8);

Port ( CLK : in std\_logic;

RST : in std\_logic;

EN : in std\_logic;

data\_in : in std\_logic\_vector(M-1 downto 0);

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

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end component;

component Rs232RefComp is

Port (

TXD : out std\_logic := '1';

RXD : in std\_logic;

CLK : in std\_logic; --Master Clock

DBIN : in std\_logic\_vector (7 downto 0); --Data Bus in

DBOUT : out std\_logic\_vector (7 downto 0); -- Data Bus out

RDA : inout std\_logic; --Read Data Available(1 quando il dato è disponibile nel registro rdReg)

TBE : inout std\_logic := '1';--Transfer Bus Empty(1 quando il dato da inviare è stato caricato nello shift register)

RD : in std\_logic; --Read Strobe(se 1 significa "leggi" --> fa abbassare RDA)

WR : in std\_logic; --Write Strobe(se 1 significa "scrivi" --> fa abbassare TBE)

PE : out std\_logic; --Parity Error Flag

FE : out std\_logic; --Frame Error Flag

OE : out std\_logic; --Overwrite Error Flag

RST : in std\_logic := '0'); --Master Reset

end component;

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

signal uart\_to\_mem: std\_logic\_vector(M-1 downto 0);

signal RDA\_bus: std\_logic;

begin

contatore :contatore\_mod\_N

Generic Map(N=>N)

Port Map( clock=>en\_count,

reset=>rst,

set=>'0',

load=>(others=>'0'),

cont=>address

);

count<=address;

mem\_b:MEM

Generic Map(N=>N,M=>M)

Port Map(

CLK=>clk,

RST=>rst,

EN=>write\_mem,

data\_in=>uart\_to\_mem,

address=>address,

data\_out=>output

);

--rda va verso la cu b per dire ho ricevuto

uart:Rs232RefComp

Port Map(

--TXD =>,

RXD =>input,

CLK =>clk,

DBIN =>(others=>'0'),

DBOUT =>uart\_to\_mem,

RDA =>RDA\_bus,

RD =>RD,

WR =>'0',

RST => rst

);

RDA<=RDA\_bus;

end Structural;

Unità di controllo:

entity UC\_b is

Generic ( N : natural :=8;

M : natural := 8);

Port ( clk: in std\_logic;

rst: in std\_logic;

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

--start : in std\_logic;

ack: out std\_logic;

req: in std\_logic;

en\_count: out std\_logic;

write\_mem: out std\_logic; RDA: in std\_logic;

RD: out std\_logic

);

end UC\_b;

architecture Behavioral of UC\_b is

type state is (wait\_req,send\_ack,en\_uart,write\_mem\_state,increment,uart\_ready);

signal stato\_corrente: state:=wait\_req;

signal stato\_prossimo: state;

begin

comb:process(stato\_corrente,count,req,RDA)

begin

case stato\_corrente is

when wait\_req =>

en\_count<='0';

rd<='0';

if(req='1') then

stato\_prossimo<=send\_ack;

else

stato\_prossimo<=wait\_req;

end if;

when send\_ack =>

ack<='1';

if(req='0') then

stato\_prossimo<=en\_uart;

else

stato\_prossimo<=send\_ack;

end if;

when en\_uart =>

RD<='1';

stato\_prossimo<=uart\_ready;

when uart\_ready =>

rd<='0';

if(RDA='1') then

stato\_prossimo<=write\_mem\_state;

else

stato\_prossimo<=uart\_ready;

end if;

when write\_mem\_state =>

ack<='0';

RD<='0';

write\_mem<='1';

stato\_prossimo<=increment;

when increment =>

write\_mem<='0';

en\_count<='1';

stato\_prossimo<=wait\_req;

end case;

end process;

mem:process(clk)

begin

if(rising\_edge(clk)) then

if(rst='1') then

stato\_corrente<=wait\_req;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

##### Sistema complessivo

Il Top Module unisce in maniera strutturale i due sistemi e li connette mediante interfaccia seriale:

entity a\_b\_system is

Generic ( N : natural :=8;

M : natural := 8);

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

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

);

end a\_b\_system;

architecture Structural of a\_b\_system is

component a is

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

ack : in std\_logic;

req: out std\_logic;

output: out std\_logic

);

end component;

component b is

Generic ( N : natural :=8;

M : natural := 8);

Port (

clk: in std\_logic;

rst: in std\_logic;

ack : out std\_logic;

req: in std\_logic;

input: in std\_logic;

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

);

end component;

signal ack: std\_logic;

signal req: std\_logic;

signal a\_to\_b: std\_logic;

begin

a\_unit:a

Port Map(

clk=>clk,

rst=>rst,

start=>start,

ack=>ack,

req=>req,

output=>a\_to\_b

);

b\_unit:b

Port Map(

clk=>clk,

rst=>rst,

ack=>ack,

req=>req,

input=>a\_to\_b,

output=>output

);

end Structural;

#### Simulazione

Il protocollo della UART mostra i primi output dopo circa 1 secondo dall’inizio dell’invio, per cui si è dovuto aumentare significativamente la finestra temporale della simulazione.

La testbench appare come segue:

entity uart\_tb is

Generic ( N : natural :=8;

M : natural := 8);

end uart\_tb;

architecture Behavioral of uart\_tb is

component a\_b\_system is

Generic ( N : natural :=8;

M : natural := 8);

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

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

);

end component;

signal clock: std\_logic;

signal period:time := 10 ns;

signal rst: std\_logic;

signal start: std\_logic;

signal output: std\_logic\_vector(M-1 downto 0);

begin

uut:a\_b\_system

Port Map(

clk=>clock,

rst=>rst,

start=>start,

output=>output);

clock\_process: process

begin

clock<='0';

wait for period/2;

clock<='1';

wait for period/2;

end process;

stim: process

begin

wait for period;

wait for period;

rst<='0';

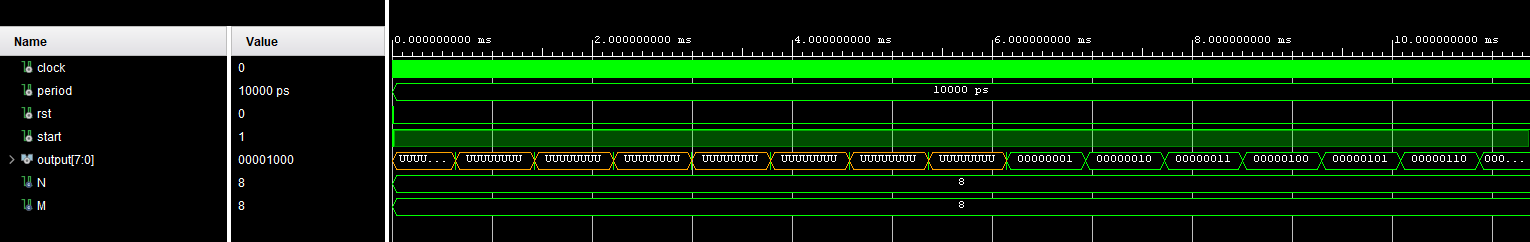
start<='1';

wait;

end process;

end Behavioral;

L’output è il seguente, e dimostra che lo scambio di dati avviene correttamente:

 Immagine che contiene schermata, Policromia, software, linea

Descrizione generata automaticamente

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

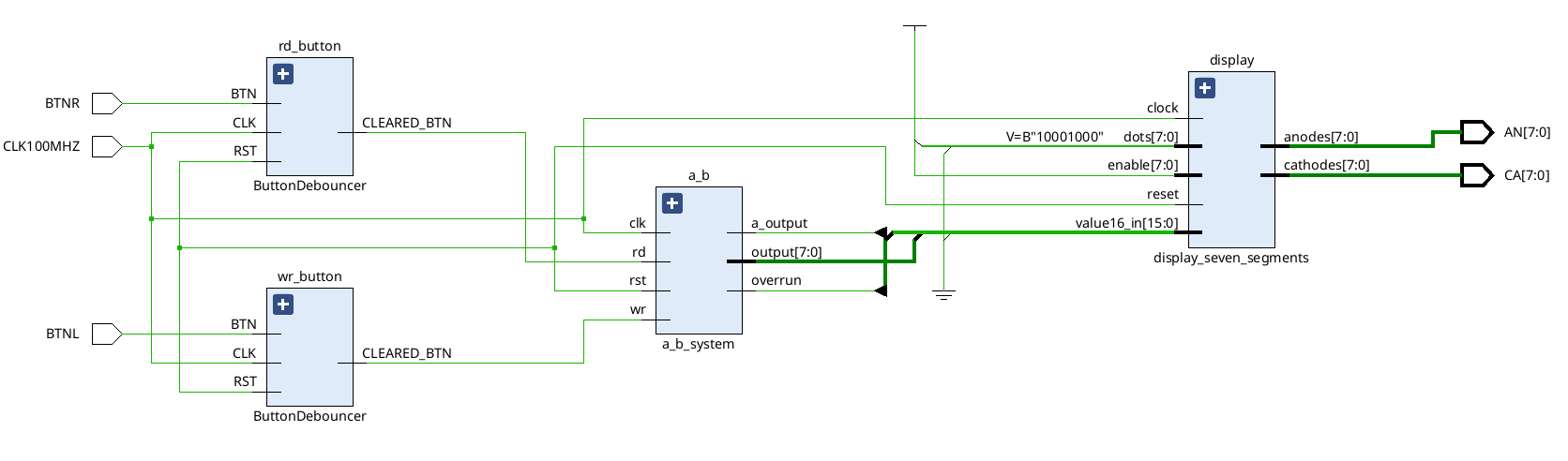
Dopo aver simulato il comportamento del sistema, lo si implementi su board usando un bottone per il segnale di WR in A, un bottone per il segnale di RD in B, e i display per la visualizzazione del dato correntemente trasmesso e memorizzato in MEM. Si testi l’errore di overrun.

#### Sintesi su board di sviluppo

Per utilizzare il componente su fpga è stato prima di tutto modificato il componente B del punto precedente per avere un segnale di start. In secondo luogo è stata realizzata un'architettura structural composta da:

* Due button debouncer per i bottoni WR e RD
* Il sistema A B del punto precedente modificato
* Il componente Display Seven Segments.

Schematics:



Codice:

Constraints:

# **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

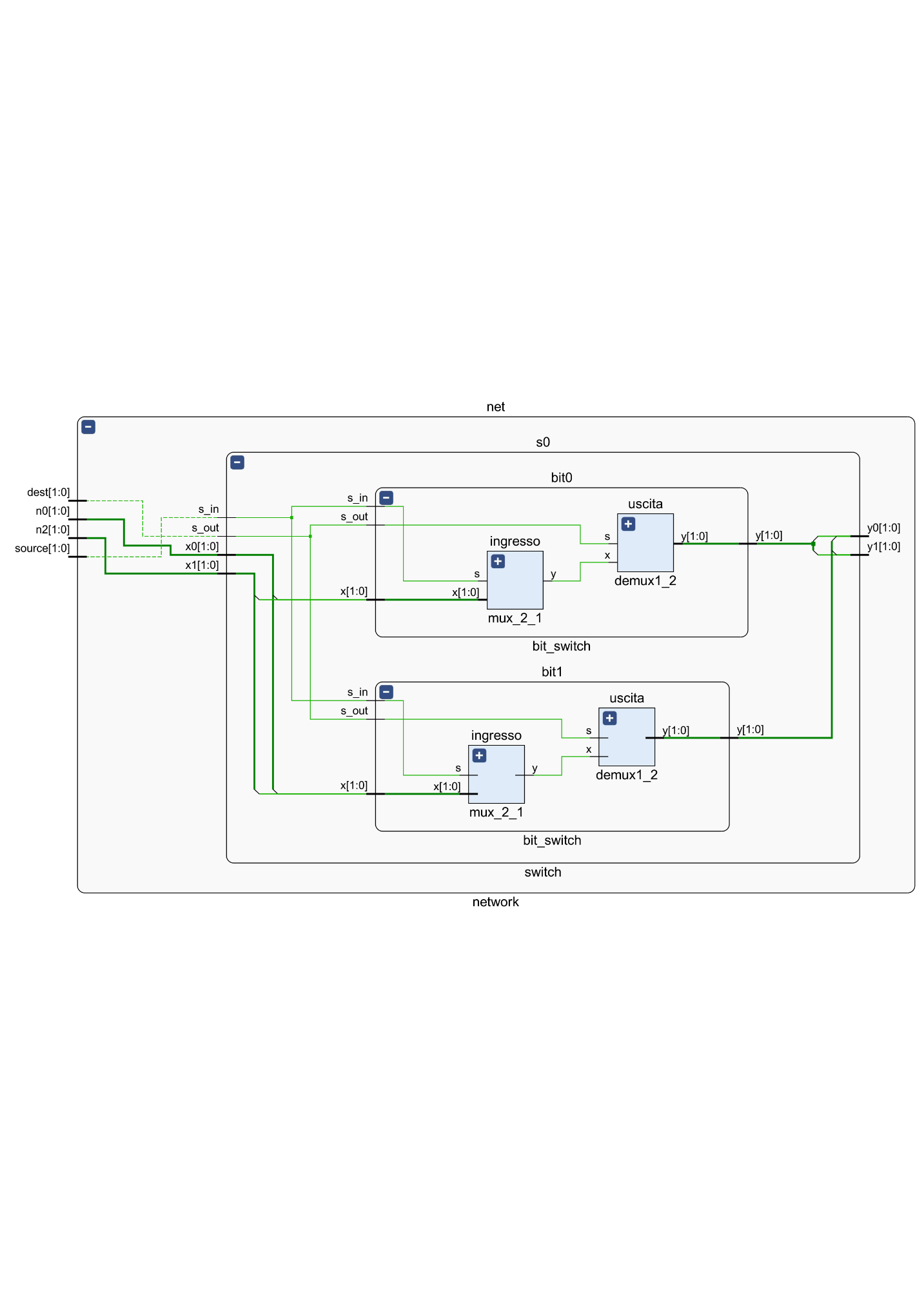
Una rete **omega network** è una rete che, mediante un componente elementare detto *switch* (concatenazione di mux 2:1 e demux 1:2) connette due sorgenti e due destinazioni.

Il modello in analisi implementa il *perfect shuffling*, algoritmo mutuato dal gioco delle carte che consiste nel mischiare le carte da gioco dividendo il mazzo in metà uguali e interlacciandole perfettamente, in modo da ritrovarsi con la prima carta della metà sinistra, seguita dalla prima carta della metà destra e così via.

Continuando a mischiare in questo modo, dopo un certo numero di mischiate ( ) il mazzo ritornerà alla situazione di partenza.

Seguendo l’algoritmo, dunque, è possibile dire che una omega network avrà stadi e switch per ogni stadio. Nel caso in analisi, dunque, ci saranno 2 stadi e 2 switch per stadio. Il primo stadio riceverà i bit già mescolati. Le selezioni dei MUX e DEMUX saranno, rispettivamente, identificate dal MSB e dal LSB della stringa sorgente.

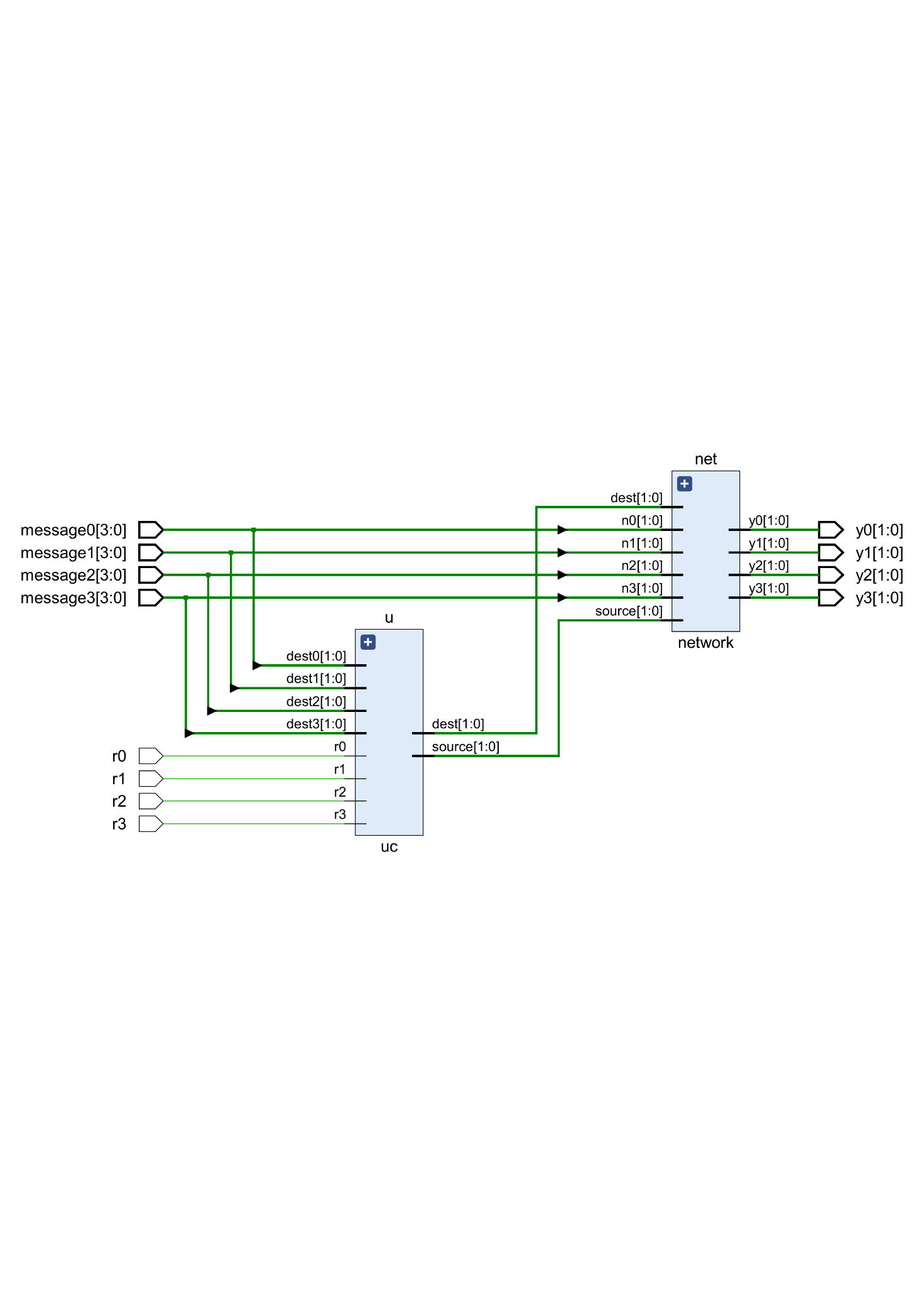
È stato implementato un componente in grado di ricevere e smistare un *payload* di due bit. Un singolo switch, quindi, è implementato nel seguente modo:



La net comprensiva dei 4 switch appare come segue:Immagine che contiene diagramma, Piano, testo, Disegno tecnico

Descrizione generata automaticamente

Infine, l’architettura complessiva è:



È prevista un’unità di controllo puramente combinatoria, che, sulla base di segnali di abilitazione, è in grado di definire una priorità e smistare il dato a priorità maggiore (dove = più prioritario e = meno prioritario).

Essendo tutto gestito in maniera combinatoria, non c’è coda e non c’è memoria; quindi, il *payload* più prioritario passerà nella network a discapito degli altri, che andranno persi (servizio *best-effort*).

#### Implementazione

Il multiplexer 2:1 è stato realizzato secondo un approccio **Dataflow**, sfruttando il componente realizzato per il [multiplexer 4:1 ad albero](#_Implementazione_1). Il demultiplexer 1:2, invece, è il seguente:

entity demux1\_2 is

Port (

x: in std\_logic;

s: in std\_logic;

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

);

end demux1\_2;

architecture Dataflow of demux1\_2 is

begin

y(0)<=x and not(s);

y(1)<=x and s;

end Dataflow;

Il componente che elabora lo smistamento dei due bit sfrutta dei MUX e DEMUX tradizionali, dunque con collegamenti fisici a un bit, e ne genera la trasmissione desiderata. Il componente è il seguente:

entity bit\_switch is

Port (

x : in STD\_LOGIC\_VECTOR (1 downto 0);

s\_in : in STD\_LOGIC;

s\_out : in STD\_LOGIC;

y : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end bit\_switch;

architecture Structural of bit\_switch is

component mux\_2\_1 is

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

s : in std\_logic;

y : out std\_logic

);

end component;

component demux1\_2 is

Port (

x: in std\_logic;

s: in std\_logic;

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

);

end component;

signal mux\_demux: std\_logic;

begin

ingresso: mux\_2\_1

Port Map( x=>x,

s=>s\_in,

y=>mux\_demux

);

uscita: demux1\_2

Port Map( x=>mux\_demux,

s=>s\_out,

y=>y

);

end Structural;

Lo switch utilizzato nella rete poggia proprio sul componente di cui sopra, ed è rappresentato come segue:

entity switch is

Port (

x0 : in STD\_LOGIC\_VECTOR (1 downto 0);

x1 : in STD\_LOGIC\_VECTOR (1 downto 0);

s\_in : in STD\_LOGIC;

s\_out : in STD\_LOGIC;

y0 : out STD\_LOGIC\_VECTOR (1 downto 0);

y1 : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end switch;

architecture Structural of switch is

component bit\_switch is

Port (

x : in STD\_LOGIC\_VECTOR (1 downto 0);

s\_in : in STD\_LOGIC;

s\_out : in STD\_LOGIC;

y : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end component;

begin

bit0:bit\_switch

Port Map(

x(0)=>x0(0),

x(1)=>x1(0),

s\_in=>s\_in,

s\_out=>s\_out,

y(0)=>y0(0),

y(1)=>y1(0)

);

bit1:bit\_switch

Port Map(

x(0)=>x0(1),

x(1)=>x1(1),

s\_in=>s\_in,

s\_out=>s\_out,

y(0)=>y0(1),

y(1)=>y1(1)

);

end Structural;

La rete utilizza gli switch con parallelismo a 2 bit e implementa l’algoritmo del perfect shuffling; per notare ciò, è sufficiente osservare i collegamenti all’ingresso degli switch:

entity network is

Port (

n0 : in STD\_LOGIC\_VECTOR (1 downto 0);

n1 : in STD\_LOGIC\_VECTOR (1 downto 0);

n2 : in STD\_LOGIC\_VECTOR (1 downto 0);

n3 : in STD\_LOGIC\_VECTOR (1 downto 0);

source : in STD\_LOGIC\_VECTOR(1 downto 0);

dest : in STD\_LOGIC\_VECTOR(1 downto 0);

y0 : out STD\_LOGIC\_VECTOR (1 downto 0);

y1 : out STD\_LOGIC\_VECTOR (1 downto 0);

y2 : out STD\_LOGIC\_VECTOR (1 downto 0);

y3 : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end network;

architecture Structural of network is

component switch is

Port (

x0 : in STD\_LOGIC\_VECTOR (1 downto 0);

x1 : in STD\_LOGIC\_VECTOR (1 downto 0);

s\_in : in STD\_LOGIC;

s\_out : in STD\_LOGIC;

y0 : out STD\_LOGIC\_VECTOR (1 downto 0);

y1 : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end component;

signal s0\_out0: std\_logic\_vector(1 downto 0);

signal s0\_out1: std\_logic\_vector(1 downto 0);

signal s1\_out0: std\_logic\_vector(1 downto 0);

signal s1\_out1: std\_logic\_vector(1 downto 0);

begin

s0:switch

Port Map(

x0=>n0, --0

x1=>n2, --2

s\_in=>source(1),

s\_out=>dest(1),

y0=>s0\_out0 , --0

y1=>s0\_out1 --2

);

s1:switch

Port Map(

x0=>n1, --1

x1=>n3, --3

s\_in=>source(1),

s\_out=>dest(1),

y0=>s1\_out0,

y1=>s1\_out1

);

s2:switch

Port Map(

x0=>s0\_out0,

x1=>s1\_out0,

s\_in=>source(0),

s\_out=>dest(0),

y0=>y0,

y1=>y1

);

s3:switch

Port Map(

x0=>s0\_out1,

x1=>s1\_out1,

s\_in=>source(0),

s\_out=>dest(0),

y0=>y2,

y1=>y3

);

end Structural;

L’unità di controllo, puramente combinatoria, filtra come desiderato le priorità in quanto, in VHDL, il primo if statement vero è quello che viene eseguito. Dunque, se dovesse essere alto, verrà eseguito immediatamente il ramo che porta alla soluzione di tale condizione. Se, invece,dovesse risultare basso, VHDL passerà al prossimo statement, e così via:

entity uc is

Port (

r0 : in std\_logic;

r1 : in std\_logic;

r2 : in std\_logic;

r3 : in std\_logic;

dest0 : in std\_logic\_vector(1 downto 0);

dest1 : in std\_logic\_vector(1 downto 0);

dest2 : in std\_logic\_vector(1 downto 0);

dest3 : in std\_logic\_vector(1 downto 0);

source : out std\_logic\_vector(1 downto 0);

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

);

end uc;

architecture Behavioral of uc is

signal source\_temp: std\_logic\_vector(1 downto 0);

signal dest\_temp: std\_logic\_vector(1 downto 0);

begin

process(r0,r1,r2,r3,dest0,dest1,dest2,dest3)

begin

if(r0='1') then

source\_temp<="00";

dest\_temp<=dest0;

elsif(r1='1') then

source\_temp<="01";

dest\_temp<=dest1;

elsif(r2='1') then

source\_temp<="10";

dest\_temp<=dest2;

elsif(r3='1') then

source\_temp<="11";

dest\_temp<=dest3;

end if;

end process;

source<=source\_temp;

dest<=dest\_temp;

end Behavioral;

Infine, si giunge al Top Module che unisce la network e l’unità di controllo che filtra le priorità. Il messaggio ricevuto è comprensivo di destinazione, che viene elaborata dalla UC, e di *payload*:

entity omega\_network\_prio\_fissa is

Generic(N: natural:=4);

Port (

message0 : in std\_logic\_vector(3 downto 0);

message1 : in std\_logic\_vector(3 downto 0);

message2 : in std\_logic\_vector(3 downto 0);

message3 : in std\_logic\_vector(3 downto 0);

r0 : in STD\_LOGIC;

r1 : in STD\_LOGIC;

r2 : in STD\_LOGIC;

r3 : in STD\_LOGIC;

y0 : out STD\_LOGIC\_VECTOR (1 downto 0);

y1 : out STD\_LOGIC\_VECTOR (1 downto 0);

y2 : out STD\_LOGIC\_VECTOR (1 downto 0);

y3 : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end omega\_network\_prio\_fissa;

architecture Structural of omega\_network\_prio\_fissa is

component network is

Port (

n0 : in STD\_LOGIC\_VECTOR (1 downto 0);

n1 : in STD\_LOGIC\_VECTOR (1 downto 0);

n2 : in STD\_LOGIC\_VECTOR (1 downto 0);

n3 : in STD\_LOGIC\_VECTOR (1 downto 0);

source : in STD\_LOGIC\_VECTOR(1 downto 0);

dest : in STD\_LOGIC\_VECTOR(1 downto 0);

y0 : out STD\_LOGIC\_VECTOR (1 downto 0);

y1 : out STD\_LOGIC\_VECTOR (1 downto 0);

y2 : out STD\_LOGIC\_VECTOR (1 downto 0);

y3 : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end component;

component uc is

Port (

r0 : in std\_logic;

r1 : in std\_logic;

r2 : in std\_logic;

r3 : in std\_logic;

dest0 : in std\_logic\_vector(1 downto 0);

dest1 : in std\_logic\_vector(1 downto 0);

dest2 : in std\_logic\_vector(1 downto 0);

dest3 : in std\_logic\_vector(1 downto 0);

source : out std\_logic\_vector(1 downto 0);

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

);

end component;

signal source\_signal : std\_logic\_vector(1 downto 0);

signal dest\_signal : std\_logic\_vector(1 downto 0);

begin

u:uc

Port Map(

r0=> r0,

r1=> r1,

r2=> r2,

r3=> r3,

dest0=>message0(3 downto 2),

dest1=>message1(3 downto 2),

dest2=>message2(3 downto 2),

dest3=>message3(3 downto 2),

source =>source\_signal,

dest=>dest\_signal

);

net:network

Port map(

n0=>message0(1 downto 0),

n1=>message1(1 downto 0),

n2=>message2(1 downto 0),

n3=>message3(1 downto 0),

source=>source\_signal,

dest=>dest\_signal,

y0=>y0,

y1=>y1,

y2=>y2,

y3=>y3

);

end Structural;

#### Simulazione

La simulazione tenta di controllare se il dato a maggior priorità venga smistato correttamente:

entity omega\_tb is

end omega\_tb;

architecture Behavioral of omega\_tb is

component omega\_network\_prio\_fissa is

Generic(N: natural:=4);

Port (

message0 : in std\_logic\_vector(3 downto 0);

message1 : in std\_logic\_vector(3 downto 0);

message2 : in std\_logic\_vector(3 downto 0);

message3 : in std\_logic\_vector(3 downto 0);

r0 : in STD\_LOGIC;

r1 : in STD\_LOGIC;

r2 : in STD\_LOGIC;

r3 : in STD\_LOGIC;

y0 : out STD\_LOGIC\_VECTOR (1 downto 0);

y1 : out STD\_LOGIC\_VECTOR (1 downto 0);

y2 : out STD\_LOGIC\_VECTOR (1 downto 0);

y3 : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end component;

signal message0 : std\_logic\_vector(3 downto 0);

signal message1 : std\_logic\_vector(3 downto 0);

signal message2 : std\_logic\_vector(3 downto 0);

signal message3 : std\_logic\_vector(3 downto 0);

signal r0 : STD\_LOGIC;

signal r1 : STD\_LOGIC;

signal r2 : STD\_LOGIC;

signal r3 : STD\_LOGIC;

signal y0 : STD\_LOGIC\_VECTOR (1 downto 0);

signal y1 : STD\_LOGIC\_VECTOR (1 downto 0);

signal y2 : STD\_LOGIC\_VECTOR (1 downto 0);

signal y3 : STD\_LOGIC\_VECTOR (1 downto 0) ;

begin

uut:omega\_network\_prio\_fissa

Port Map(

message0 =>message0 ,

message1 =>message1 ,

message2 =>message2 ,

message3 =>message3 ,

r0 =>r0 ,

r1 =>r1 ,

r2 =>r2 ,

r3 =>r3 ,

y0 =>y0 ,

y1 =>y1 ,

y2 =>y2 ,

y3 =>y3

);

stim:process

begin

wait for 10 ns;

message0<="0111";

message1<="0010";

message2<="0001";

message3<="0011";

r0<='1';

r1<='1';

r2<='0';

r3<='0';

wait for 10 ns;

r0<='0';

r1<='0';

r2<='0';

r3<='1';

wait;

end process;

end Behavioral;

È possibile osservare che ciò avvenga correttamente:

Immagine che contiene schermata, linea, Policromia

Descrizione generata automaticamente

### 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

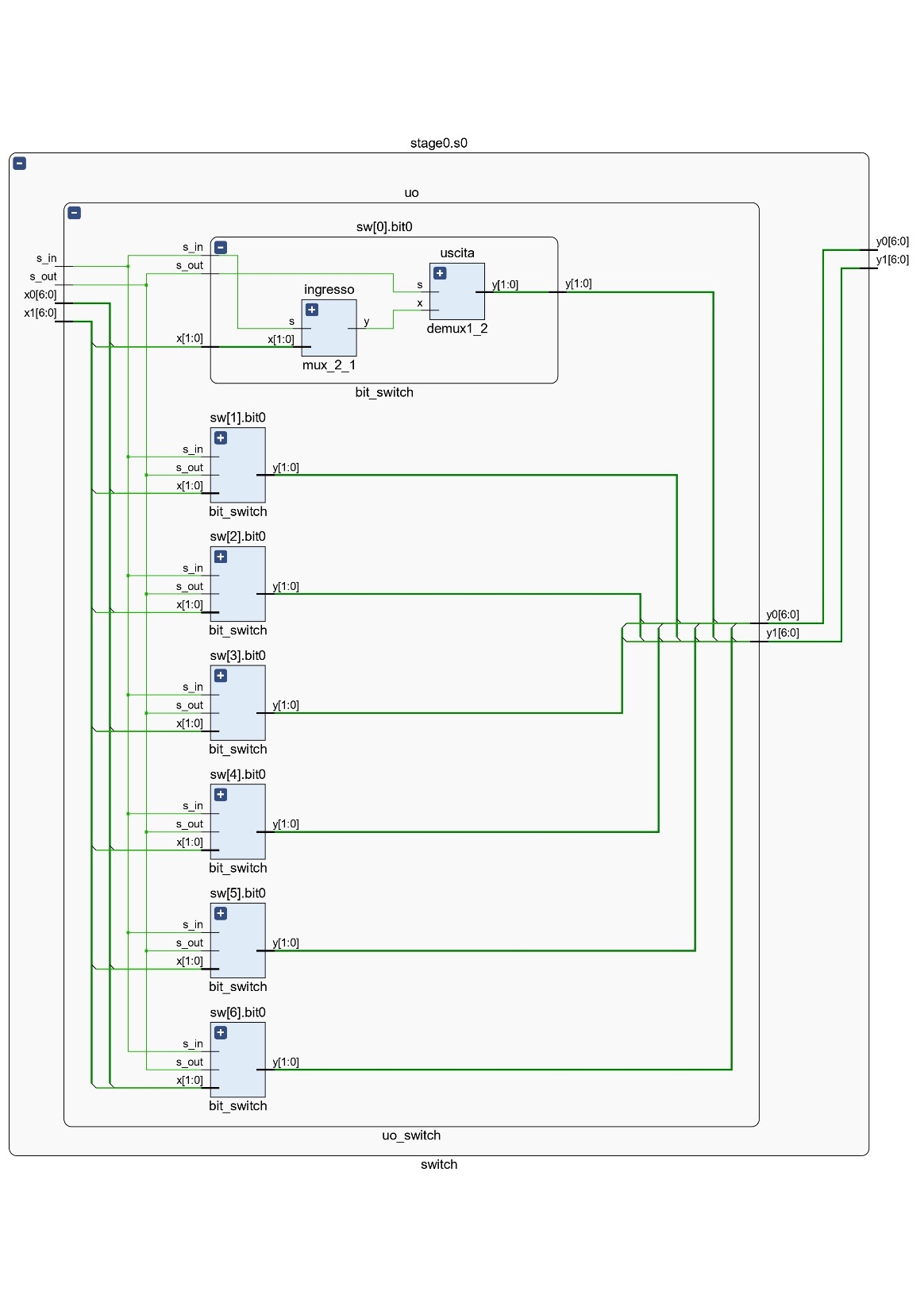
Per la realizzazione di questo sistema prima di tutto bisogna considerare il perfect shuffling per 8 nodi che quindi necessita di 3 stati di cui ognuno di 4 nodi. Inoltre non adottando uno schema a priorità fissa si è scelto di adottare una strategia di tipo store-and-forward.

In particolare, i messaggi saranno di 7 bit così composti: i primi 3 bit rappresentano il payload del messaggio, i bit da 4 a 6 indicano la destinazione a cui recapitare il messaggio e il settimo bit se alto indica la richiesta di invio da parte di un nodo.

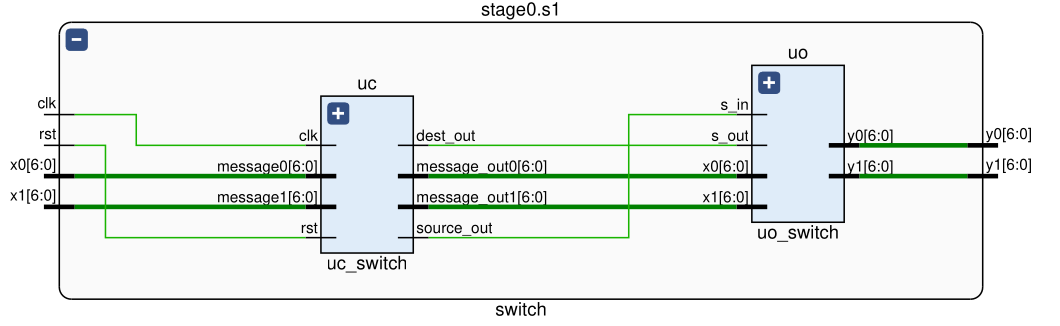
Per gestire l’instradamento dei messaggi i singoli switch sono stati riprogettati per avere una Unità di Controllo e un Unità operativa. L’unità di controllo adotta al suo interno una politica a priorità fissa, in cui se si ricevono due richieste contemporanea, il ricevente prioritario sarà autorizzato a mandare il messaggio e il secondo messaggio verrà memorizzato in un buffer. Al prossimo ciclo di clock se il buffer è pieno viene data priorità ad esso e inviato il messaggio, svuotando il buffer.

La parte Operativa del singolo switch ora è composta da 7 bit switch per poter gestire la dimensione del messaggio.

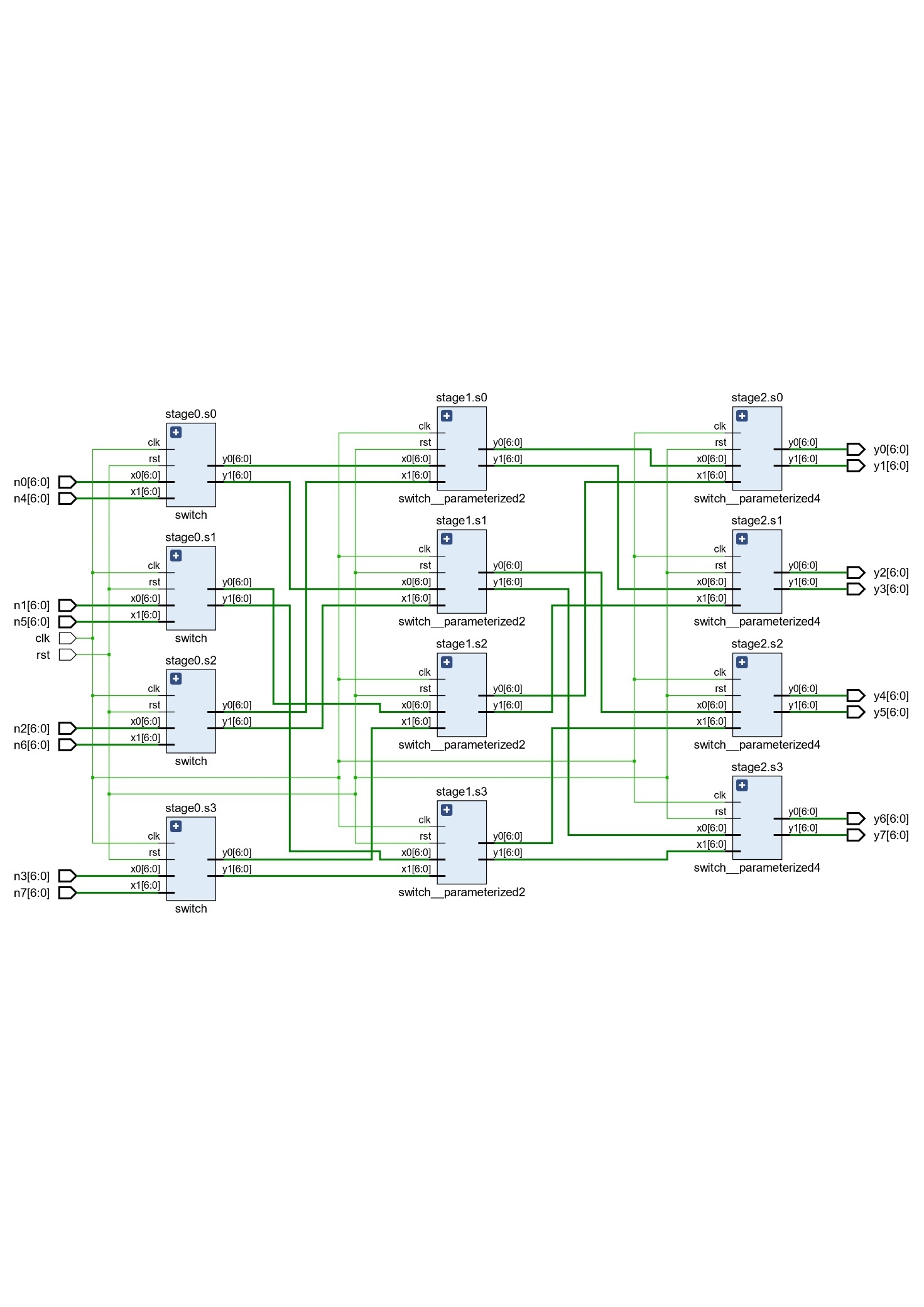
Unità Operativa dello Switch:



Switch complessivo:



Sistema complessivo:



#### Implementazione

Bit switch: analogo [all’esercizio 11.1](#_Implementazione_3).

Switch:

entity switch is

Generic(Stage:natural:=0);

Port (

clk : std\_logic;

rst: std\_logic;

x0 : in STD\_LOGIC\_VECTOR (6 downto 0);

x1 : in STD\_LOGIC\_VECTOR (6 downto 0);

y0 : out STD\_LOGIC\_VECTOR (6 downto 0);

y1 : out STD\_LOGIC\_VECTOR (6 downto 0)

);

end switch;

architecture Structural of switch is

component uc\_switch is

Generic(Stage:natural:=0);

Port (clk : std\_logic;

rst: std\_logic;

message0: in std\_logic\_vector(6 downto 0);

message1: in std\_logic\_vector(6 downto 0);

message\_out0:out std\_logic\_vector(6 downto 0);

message\_out1:out std\_logic\_vector(6 downto 0);

source\_out: out std\_logic;

dest\_out: out std\_logic

);

end component;

component uo\_switch is

Port (

x0 : in STD\_LOGIC\_VECTOR (6 downto 0);

x1 : in STD\_LOGIC\_VECTOR (6 downto 0);

s\_in : in STD\_LOGIC;

s\_out : in STD\_LOGIC;

y0 : out STD\_LOGIC\_VECTOR (6 downto 0);

y1 : out STD\_LOGIC\_VECTOR (6 downto 0)

);

end component;

signal sel:std\_logic;

signal dest: std\_logic;

signal message0\_uc: std\_logic\_vector(6 downto 0);

signal message1\_uc: std\_logic\_vector(6 downto 0);

begin

uc: uc\_switch

Generic Map(Stage=>Stage)

Port Map( clk=>clk,

rst=>rst,

message0 =>x0,

message1 =>x1,

message\_out0=>message0\_uc,

message\_out1=>message1\_uc,

source\_out=>sel,

dest\_out=>dest

);

uo:uo\_switch

Port Map(

x0 =>message0\_uc,

x1 =>message1\_uc,

s\_in =>sel,

s\_out =>dest,

y0 =>y0,

y1 =>y1

);

end Structural;

Unità di controllo:

entity uc\_switch is

Generic(Stage:natural:=0);

Port ( clk : std\_logic;

rst: std\_logic;

message0: in std\_logic\_vector(6 downto 0);

message1: in std\_logic\_vector(6 downto 0);

message\_out0:out std\_logic\_vector(6 downto 0);

message\_out1:out std\_logic\_vector(6 downto 0);

source\_out: out std\_logic;

dest\_out: out std\_logic

);

end uc\_switch;

architecture Behavioral of uc\_switch is

signal data\_store: std\_logic\_vector(6 downto 0);

begin

boh:process(clk,message0,message1)

begin

if(rising\_edge(clk)) then

if(data\_store(6)='1') then

message\_out1<=data\_store(6 downto 0);

source\_out<='1';

dest\_out<=data\_store(5-Stage);

data\_store(6)<='0';

elsif(message0(6)='1' and message1(6)='1') then

data\_store<=message1;

message\_out0<=message0;

source\_out<='0';

dest\_out<=message0(5-Stage);

elsif(message0(6)='1') then

message\_out0<=message0;

source\_out<='0';

dest\_out<=message0(5-Stage);

elsif(message1(6)='1') then

message\_out1<=message1;

source\_out<='1';

dest\_out<=message1(5-Stage);

else

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

message\_out0<=(others=>'0');

message\_out1<=(others=>'0');

end if;

end if;

end process;

end Behavioral;

Unità operativa:

entity uo\_switch is

Port (

x0 : in STD\_LOGIC\_VECTOR (6 downto 0);

x1 : in STD\_LOGIC\_VECTOR (6 downto 0);

s\_in : in STD\_LOGIC;

s\_out : in STD\_LOGIC;

y0 : out STD\_LOGIC\_VECTOR (6 downto 0);

y1 : out STD\_LOGIC\_VECTOR (6 downto 0)

);

end uo\_switch;

architecture Structural of uo\_switch is

component bit\_switch is

Port (

x : in STD\_LOGIC\_VECTOR (1 downto 0);

s\_in : in STD\_LOGIC;

s\_out : in STD\_LOGIC;

y : out STD\_LOGIC\_VECTOR (1 downto 0)

);

end component;

begin

sw:for i in 6 downto 0 generate

bit0:bit\_switch

Port Map(

x(0)=>x0(i),

x(1)=>x1(i),

s\_in=>s\_in,

s\_out=>s\_out,

y(0)=>y0(i),

y(1)=>y1(i)

);

end generate;

end Structural;

Rete complessiva:

entity network is

Port (

clk : std\_logic;

rst: std\_logic;

n0 : in STD\_LOGIC\_VECTOR (6 downto 0);

n1 : in STD\_LOGIC\_VECTOR (6 downto 0);

n2 : in STD\_LOGIC\_VECTOR (6 downto 0);

n3 : in STD\_LOGIC\_VECTOR (6 downto 0);

n4 : in STD\_LOGIC\_VECTOR (6 downto 0);

n5 : in STD\_LOGIC\_VECTOR (6 downto 0);

n6 : in STD\_LOGIC\_VECTOR (6 downto 0);

n7 : in STD\_LOGIC\_VECTOR (6 downto 0);

y0 : out STD\_LOGIC\_VECTOR (6 downto 0);

y1 : out STD\_LOGIC\_VECTOR (6 downto 0);

y2 : out STD\_LOGIC\_VECTOR (6 downto 0);

y3 : out STD\_LOGIC\_VECTOR (6 downto 0);

y4 : out STD\_LOGIC\_VECTOR (6 downto 0);

y5 : out STD\_LOGIC\_VECTOR (6 downto 0);

y6 : out STD\_LOGIC\_VECTOR (6 downto 0);

y7 : out STD\_LOGIC\_VECTOR (6 downto 0)

);

end network;

architecture Structural of network is

component switch is

Generic(Stage: natural :=0);

Port ( clk : std\_logic;

rst: std\_logic;

x0 : in STD\_LOGIC\_VECTOR (6 downto 0);

x1 : in STD\_LOGIC\_VECTOR (6 downto 0);

y0 : out STD\_LOGIC\_VECTOR (6 downto 0);

y1 : out STD\_LOGIC\_VECTOR (6 downto 0)

);

end component;

type switch\_out is array (0 to 1) of std\_logic\_vector(6 downto 0);

type stage\_out is array (0 to 3) of switch\_out;

signal stage0\_out:stage\_out;

signal stage1\_out:stage\_out;

begin

stage0: block

begin

s0:switch

Generic Map(Stage=>0)

Port Map(

clk=>clk,

rst=>rst,

x0=>n0,

x1=>n4,

y0=>stage0\_out(0)(0) ,

y1=>stage0\_out(0)(1)

);

s1:switch

Generic Map(Stage=>0)

Port Map( clk=>clk,

rst=>rst,

x0=>n1,

x1=>n5,

y0=>stage0\_out(1)(0),

y1=>stage0\_out(1)(1)

);

s2:switch

Generic Map(Stage=>0)

Port Map( clk=>clk,

rst=>rst,

x0=>n2,

x1=>n6,

y0=>stage0\_out(2)(0),

y1=>stage0\_out(2)(1)

);

s3:switch

Generic Map(Stage=>0)

Port Map( clk=>clk,

rst=>rst,

x0=>n3,

x1=>n7,

y0=>stage0\_out(3)(0),

y1=>stage0\_out(3)(1)

);

--stage 1

end block stage0;

stage1: block

begin

s0:switch

Generic Map(Stage=>1)

Port Map( clk=>clk,

rst=>rst,

x0=>stage0\_out(0)(0),

x1=>stage0\_out(2)(0),

y0=>stage1\_out(0)(0),

y1=>stage1\_out(0)(1)

);

s1:switch

Generic Map(Stage=>1)

Port Map( clk=>clk,

rst=>rst,

x0=>stage0\_out(0)(1),

x1=>stage0\_out(2)(1),

y0=>stage1\_out(1)(0),

y1=>stage1\_out(1)(1)

);

s2:switch

Generic Map(Stage=>1)

Port Map( clk=>clk,

rst=>rst,

x0=>stage0\_out(1)(0),

x1=>stage0\_out(3)(0),

y0=>stage1\_out(2)(0),

y1=>stage1\_out(2)(1)

);

s3:switch

Generic Map(Stage=>1)

Port Map( clk=>clk,

rst=>rst,

x0=>stage0\_out(1)(1),

x1=>stage0\_out(3)(1),

y0=>stage1\_out(3)(0),

y1=>stage1\_out(3)(1)

);

end block stage1;

stage2: block

begin

s0:switch

Generic Map(Stage=>2)

Port Map(

clk=>clk,

rst=>rst,

x0=>stage1\_out(0)(0),

x1=>stage1\_out(2)(0),

y0=>y0 ,

y1=>y1

);

s1:switch

Generic Map(Stage=>2)

Port Map( clk=>clk,

rst=>rst,

x0=>stage1\_out(0)(1),

x1=>stage1\_out(2)(1),

y0=>y2,

y1=>y3

);

s2:switch

Generic Map(Stage=>2)

Port Map( clk=>clk,

rst=>rst,

x0=>stage1\_out(1)(0),

x1=>stage1\_out(3)(0),

y0=>y4,

y1=>y5

);

s3:switch

Generic Map(Stage=>2)

Port Map( clk=>clk,

rst=>rst,

x0=>stage1\_out(1)(1),

x1=>stage1\_out(3)(1),

y0=>y6,

y1=>y7

);

--stage 1

end block stage2;

end Structural;

#### Simulazione

La testbench è la seguente:

entity omega\_collision\_tb is

-- Port ( );

end omega\_collision\_tb;

architecture Behavioral of omega\_collision\_tb is

component network is

Port ( clk : std\_logic;

rst: std\_logic;

n0 : in STD\_LOGIC\_VECTOR (6 downto 0);

n1 : in STD\_LOGIC\_VECTOR (6 downto 0);

n2 : in STD\_LOGIC\_VECTOR (6 downto 0);

n3 : in STD\_LOGIC\_VECTOR (6 downto 0);

n4 : in STD\_LOGIC\_VECTOR (6 downto 0);

n5 : in STD\_LOGIC\_VECTOR (6 downto 0);

n6 : in STD\_LOGIC\_VECTOR (6 downto 0);

n7 : in STD\_LOGIC\_VECTOR (6 downto 0);

y0 : out STD\_LOGIC\_VECTOR (6 downto 0);

y1 : out STD\_LOGIC\_VECTOR (6 downto 0);

y2 : out STD\_LOGIC\_VECTOR (6 downto 0);

y3 : out STD\_LOGIC\_VECTOR (6 downto 0);

y4 : out STD\_LOGIC\_VECTOR (6 downto 0);

y5 : out STD\_LOGIC\_VECTOR (6 downto 0);

y6 : out STD\_LOGIC\_VECTOR (6 downto 0);

y7 : out STD\_LOGIC\_VECTOR (6 downto 0)

);

end component;

signal n0 : std\_logic\_vector(6 downto 0);

signal n1 : std\_logic\_vector(6 downto 0);

signal n2 : std\_logic\_vector(6 downto 0);

signal n3 : std\_logic\_vector(6 downto 0);

signal n4 : std\_logic\_vector(6 downto 0);

signal n5 : std\_logic\_vector(6 downto 0);

signal n6 : std\_logic\_vector(6 downto 0);

signal n7 : std\_logic\_vector(6 downto 0);

signal y0 : std\_logic\_vector(6 downto 0);

signal y1 : std\_logic\_vector(6 downto 0);

signal y2 : std\_logic\_vector(6 downto 0);

signal y3 : std\_logic\_vector(6 downto 0);

signal y4 : std\_logic\_vector(6 downto 0);

signal y5 : std\_logic\_vector(6 downto 0);

signal y6 : std\_logic\_vector(6 downto 0);

signal y7 : std\_logic\_vector(6 downto 0);

signal clock: std\_logic;

signal rst: std\_logic;

signal period: time:= 10 ns;

begin

cprocess:process

begin

clock<='0';

wait for period/2;

clock<='1';

wait for period/2;

end process;

uut:network

Port Map(

clk=>clock,

rst=>rst,

n0 =>n0,

n1 =>n1,

n2 =>n2,

n3 =>n3,

n4 =>n4,

n5 =>n5,

n6 =>n6,

n7 =>n7,

y0 =>y0,

y1 =>y1,

y2 =>y2,

y3 =>y3,

y4 =>y4,

y5 =>y5,

y6 =>y6,

y7 =>y7

);

stim:process

begin

n0<="1000001";

n1<="1001001";

n2<="1010010";

n3<="1011011";

n4<="1100100";

n5<="1101101";

n6<="1110110";

n7<="1111111";

wait for 30 ns;

n0<="0000001";

n1<="0001001";

n2<="0010010";

n3<="0011011";

n4<="0100100";

n5<="0101101";

n6<="0110110";

n7<="0111111";

wait;

end process;

end Behavioral;

Output della simulazione:

Immagine che contiene schermata, Software multimediale, testo, software

Descrizione generata automaticamente

### Esercizio 11.3 (solo 9 CFU)

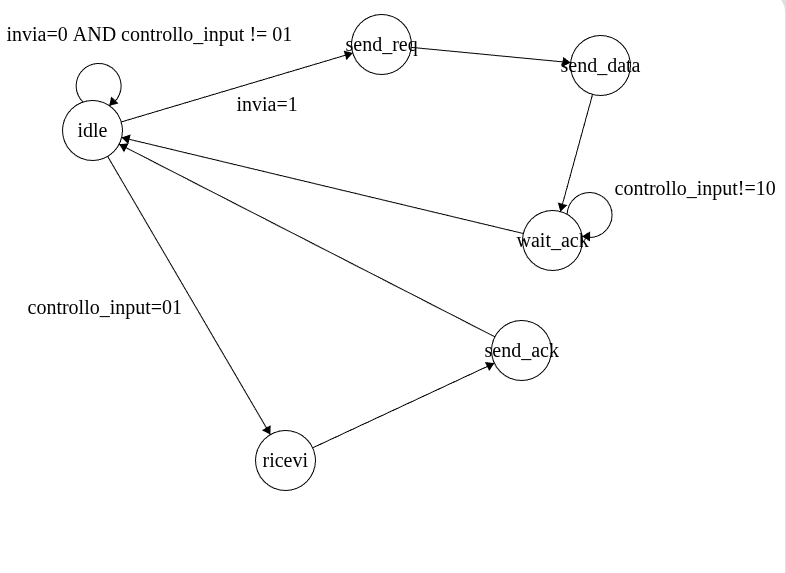
Si implementi un protocollo di handshaking semplice regolato da una coppia di segnali (pronto a inviare/pronto a ricevere) per l’invio di ciascun messaggio fra due nodi.

#### Progetto e architettura

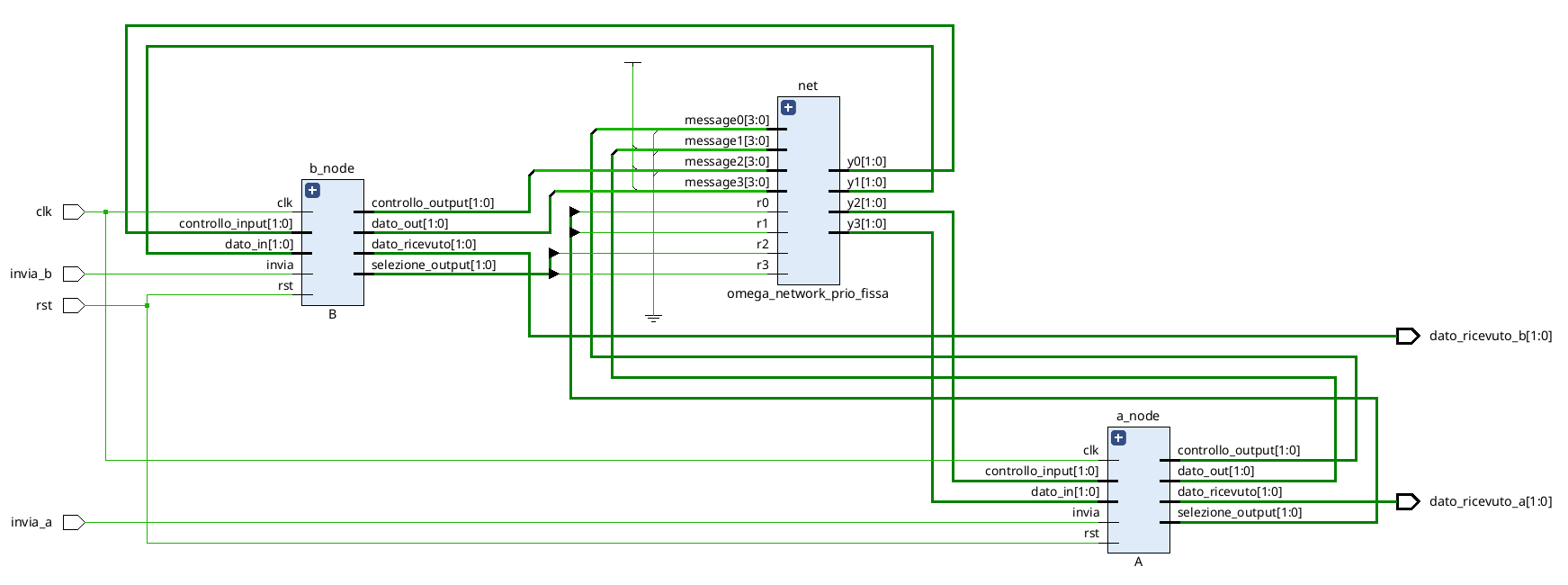
Il sistema è composto di due nodi, A e B, entrambi realizzati come automi a stati finiti. I nodi sono dotati di due linee di input e due linee di output. Le due linee di input sono una di controllo, per ricevere una richiesta di invio o ricevere l’ack di ricezione, le due linee di output sono una di dato e l’altra di controllo nella direzione inversa, per ricevere una richiesta di invio o mandare un ack.

I due nodi sono stati collegati alla omega network sviluppata all’esercizio 11.1, alla quale sono state collegate per ogni nodo due linee di input, di due output e due di richiesta.

Automa di A e B:



Sistema Complessivo:



#### Implementazione

La network è analoga [all’esercizio 11.1](#_Implementazione_3).

Nodo A:

entity A is

Port (

clk : in std\_logic;

rst : in std\_logic;

invia : in std\_logic;

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

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

controllo\_output : out std\_logic\_vector(1 downto 0);

dato\_out : out std\_logic\_vector(1 downto 0);

selezione\_output : out std\_logic\_vector(1 downto 0);

dato\_ricevuto : out std\_logic\_vector(1 downto 0)

);

end A;

-- controllo\_input = 01 richiesta di ricevere

-- controllo\_input = 10 richiesta di ricevere accettata

-- controllo\_output = 01 richiesta di mandare

-- controllo\_output = 10 richiesta di mandare accettata

-- selezione per selezionare linea controllo o linea dato

architecture Behavioral of A is

type state is (idle,send\_req,wait\_ack,send\_data,send\_ack,ricevi);

signal stato\_corrente : state := idle;

signal stato\_prossimo : state;

signal temp: std\_logic\_vector(1 downto 0);

begin

comb:process(stato\_corrente,invia,controllo\_input,dato\_in)

begin

case stato\_corrente is

when idle =>

selezione\_output<="00";

if(invia='1') then

stato\_prossimo<=send\_req;

elsif(controllo\_input="01") then

stato\_prossimo<=send\_ack;

else

stato\_prossimo<=idle;

end if;

when send\_req =>

selezione\_output(0)<='1';

controllo\_output<="01";

stato\_prossimo<=wait\_ack;

when wait\_ack =>

selezione\_output(0)<='0';

if(controllo\_input="10") then

stato\_prossimo<=send\_data;

else

stato\_prossimo<=wait\_ack;

end if;

when send\_data =>

selezione\_output(0)<='0';

selezione\_output(1)<='1';

dato\_out<="11";

stato\_prossimo<=idle;

when send\_ack =>

selezione\_output(0)<='1';

controllo\_output<="10";

stato\_prossimo<=ricevi;

when ricevi =>

selezione\_output(0)<='0';

stato\_prossimo<=idle;

end case;

end process;

mem:process(clk)

begin

if(rising\_edge(clk)) then

if(rst='1') then

stato\_corrente<=idle;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

dato\_ricevuto<=temp;

end Behavioral;

Nodo B:

entity B is

Port (

clk : in std\_logic;

rst : in std\_logic;

invia : in std\_logic;

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

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

controllo\_output : out std\_logic\_vector(1 downto 0);

dato\_out : out std\_logic\_vector(1 downto 0);

selezione\_output : out std\_logic\_vector(1 downto 0);

dato\_ricevuto : out std\_logic\_vector(1 downto 0)

);

end B;

-- controllo\_input = 01 richiesta di ricevere

-- controllo\_input = 10 richiesta di ricevere accettata

-- controllo\_output = 01 richiesta di mandare

-- controllo\_output = 10 richiesta di mandare accettata

-- selezione per selezionare linea controllo o linea dato

architecture Behavioral of B is

type state is (idle,send\_req,wait\_ack,send\_data,send\_ack,ricevi);

signal stato\_corrente : state := idle;

signal stato\_prossimo : state;

signal temp: std\_logic\_vector(1 downto 0);

begin

comb:process(stato\_corrente,invia,controllo\_input,dato\_in)

begin

case stato\_corrente is

when idle =>

selezione\_output<="00";

if(invia='1') then

stato\_prossimo<=send\_req;

elsif(controllo\_input="01") then

stato\_prossimo<=send\_ack;

else

stato\_prossimo<=idle;

end if;

when send\_req =>

selezione\_output(0)<='1';

controllo\_output<="01";

stato\_prossimo<=wait\_ack;

when wait\_ack =>

selezione\_output(0)<='0';

if(controllo\_input="10") then

stato\_prossimo<=send\_data;

else

stato\_prossimo<=wait\_ack;

end if;

when send\_data =>

selezione\_output(0)<='0';

selezione\_output(1)<='1';

dato\_out<="11";

stato\_prossimo<=idle;

when send\_ack =>

selezione\_output(0)<='1';

controllo\_output<="10";

stato\_prossimo<=ricevi;

when ricevi =>

selezione\_output(0)<='0';

temp<=dato\_in;

stato\_prossimo<=idle;

end case;

end process;

mem:process(clk)

begin

if(rising\_edge(clk)) then

if(rst='1') then

stato\_corrente<=idle;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

dato\_ricevuto<=temp;

end Behavioral;

#### Simulazione

La testbench è la seguente:

entity handshake\_tb is

end handshake\_tb;

architecture Behavioral of handshake\_tb is

component handshake\_omega\_network is

Port (

clk : in std\_logic;

rst : in std\_logic;

invia\_a : in std\_logic;

invia\_b : in std\_logic;

dato\_ricevuto\_a : out std\_logic\_vector(1 downto 0);

dato\_ricevuto\_b : out std\_logic\_vector(1 downto 0)

);

end component;

signal clock :std\_logic;

signal rst :std\_logic;

signal invia\_a :std\_logic;

signal invia\_b :std\_logic;

signal period :time :=10 ns;

signal datoa :std\_logic\_vector(1 downto 0);

signal datob :std\_logic\_vector(1 downto 0);

begin

uut:handshake\_omega\_network

Port Map(

clk =>clock,

rst =>rst,

invia\_a =>invia\_a,

invia\_b =>invia\_b,

dato\_ricevuto\_a=>datoa,

dato\_ricevuto\_b=>datoa

);

cprocess:process

begin

clock<='0';

wait for period/2;

clock<='1';

wait for period/2;

end process;

stim:process

begin

invia\_b<='0';

invia\_a<='1';

wait for 10 ns;

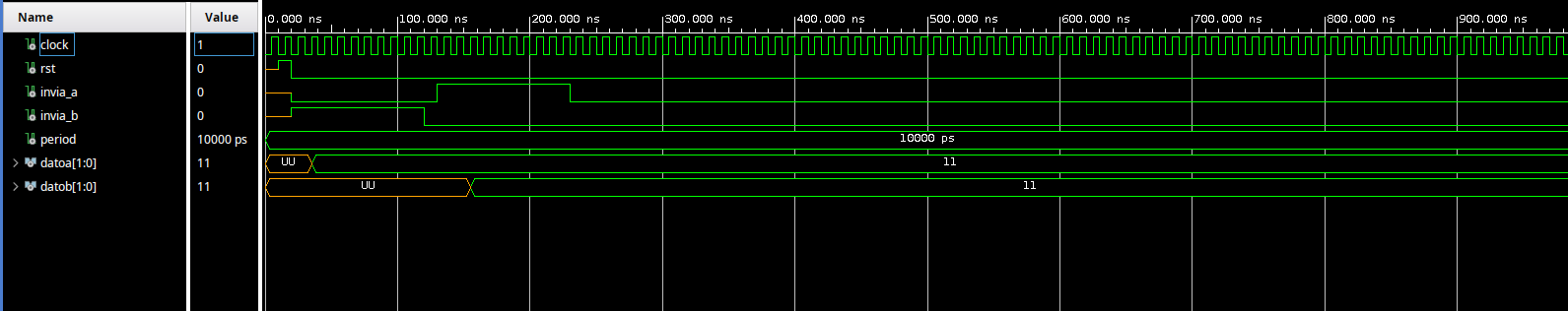
invia\_a<='0';

invia\_b<='1';

wait;

end process;

end Behavioral;

Risultato:

# **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

Il sistema funziona mediante handshaking, come approfondito [all’esercizio 8.1](#_Esercizio_8.1).

##### Sistema A

Il sistema A è dotato di:

* un **contatore modulo N**;
* una **memoria ROM**;
* una **Control Unit**.

L’architettura è la seguente:

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

L’unità di controllo implementa il seguente automa a stati finiti:

Immagine che contiene nero, oscurità

Descrizione generata automaticamente

/EN\_COUNT = 1

/READ\_ROM = 0

/REQ = 0

/REQ = 1

/READ\_ROM = 1

/EN\_COUNT = 0

/EN\_COUNT = 0

##### Sistema B

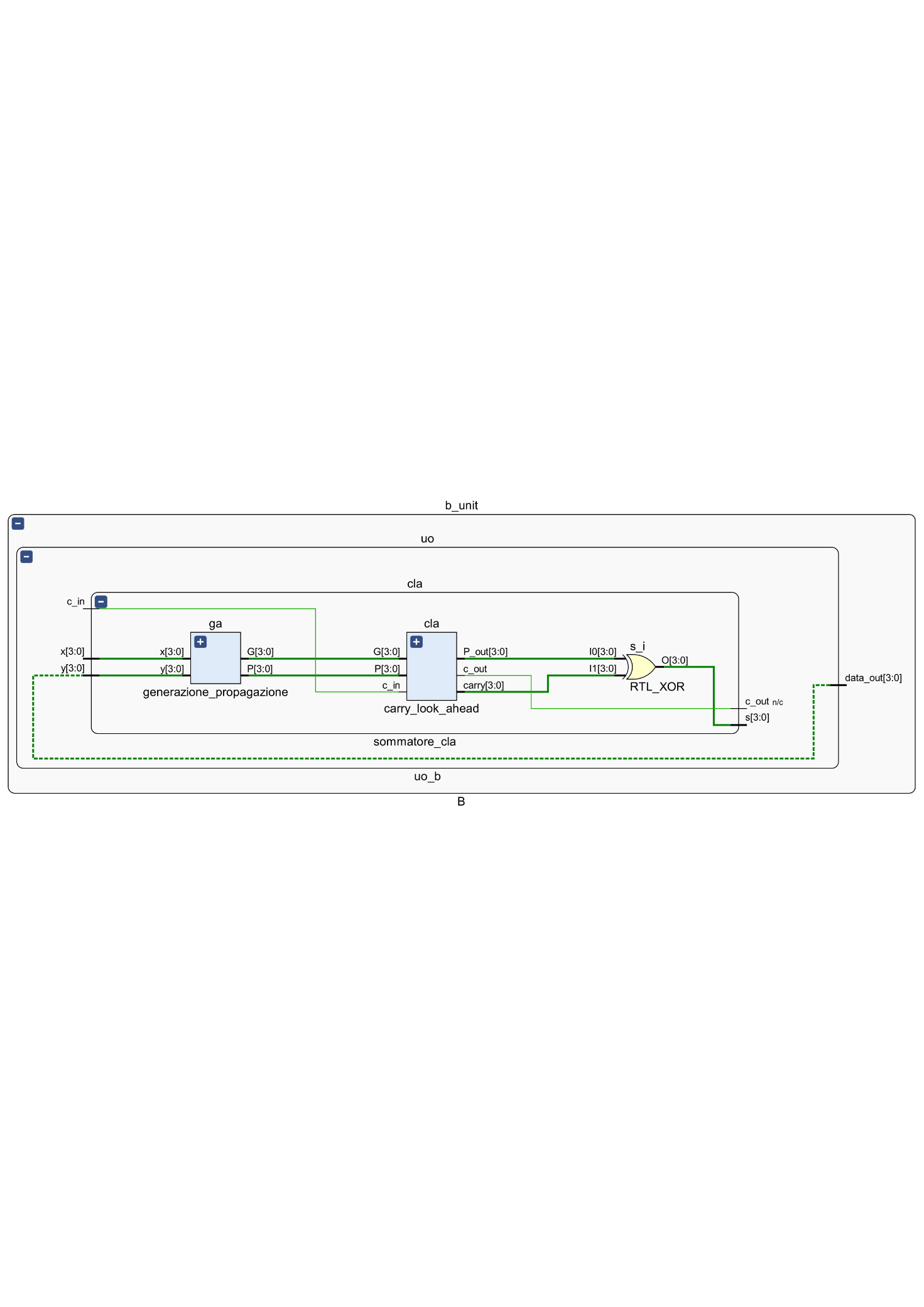
Il sistema B è dotato di:

* un **contatore modulo N**;
* un sommatore **Carry Look-Ahead**;
* un **registro di ingresso**;
* un **registro di uscita**;
* una **Control Unit**.

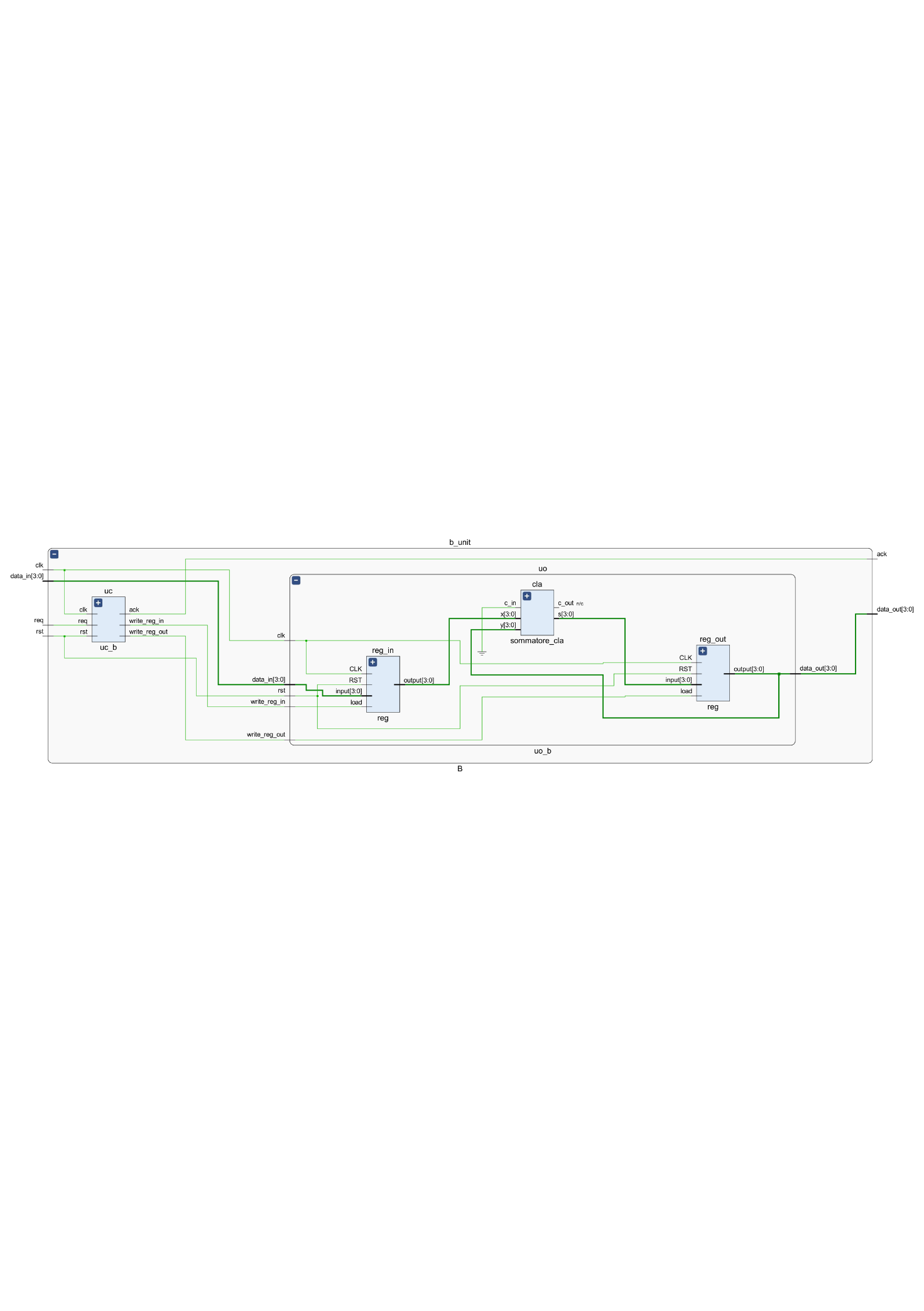
Il sommatore carry look-ahead “anticipa” il calcolo della somma e dei riporti, invece di propagarli come fa un RCA. Dunque, vi è un componente che genera due funzioni:

* **funzione di generazione**, , serve a generare un riporto dati gli input;
* **funzione di propagazione**, , serve a generare la somma effettiva senza tener conto del riporto (la somma algebrica è implementata, infatti, da una XOR).

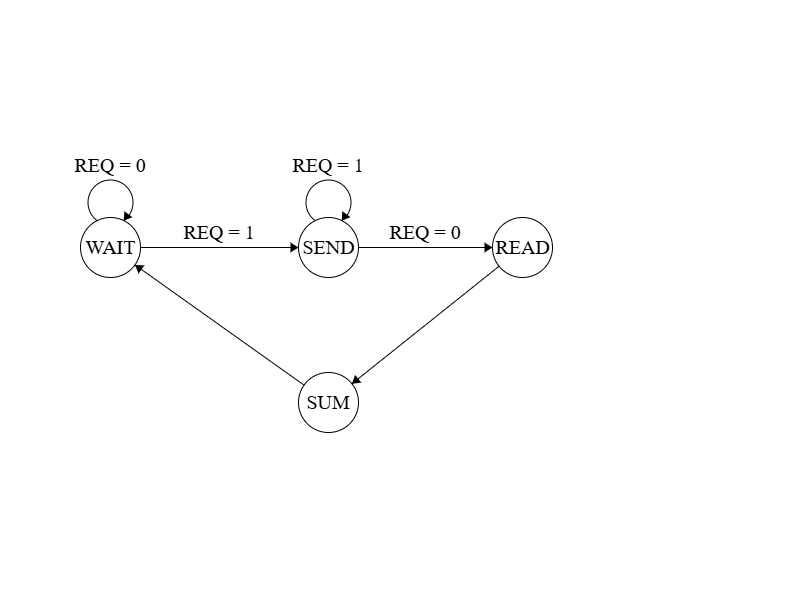
e vengono poi inviati a un componente che genera tutti i riporti in uscita (sia quelli della somma che quello di overflow), e infine viene effettuata una XOR tra la propagazione e i carry per effettuare la somma finale:



L’architettura complessiva del nodo B è la seguente:



L’unità di controllo implementa il seguente automa a stati finiti:



/WRITE\_REG\_OUT = 1

/WRITE\_REG\_IN = 1

/ACK = 0

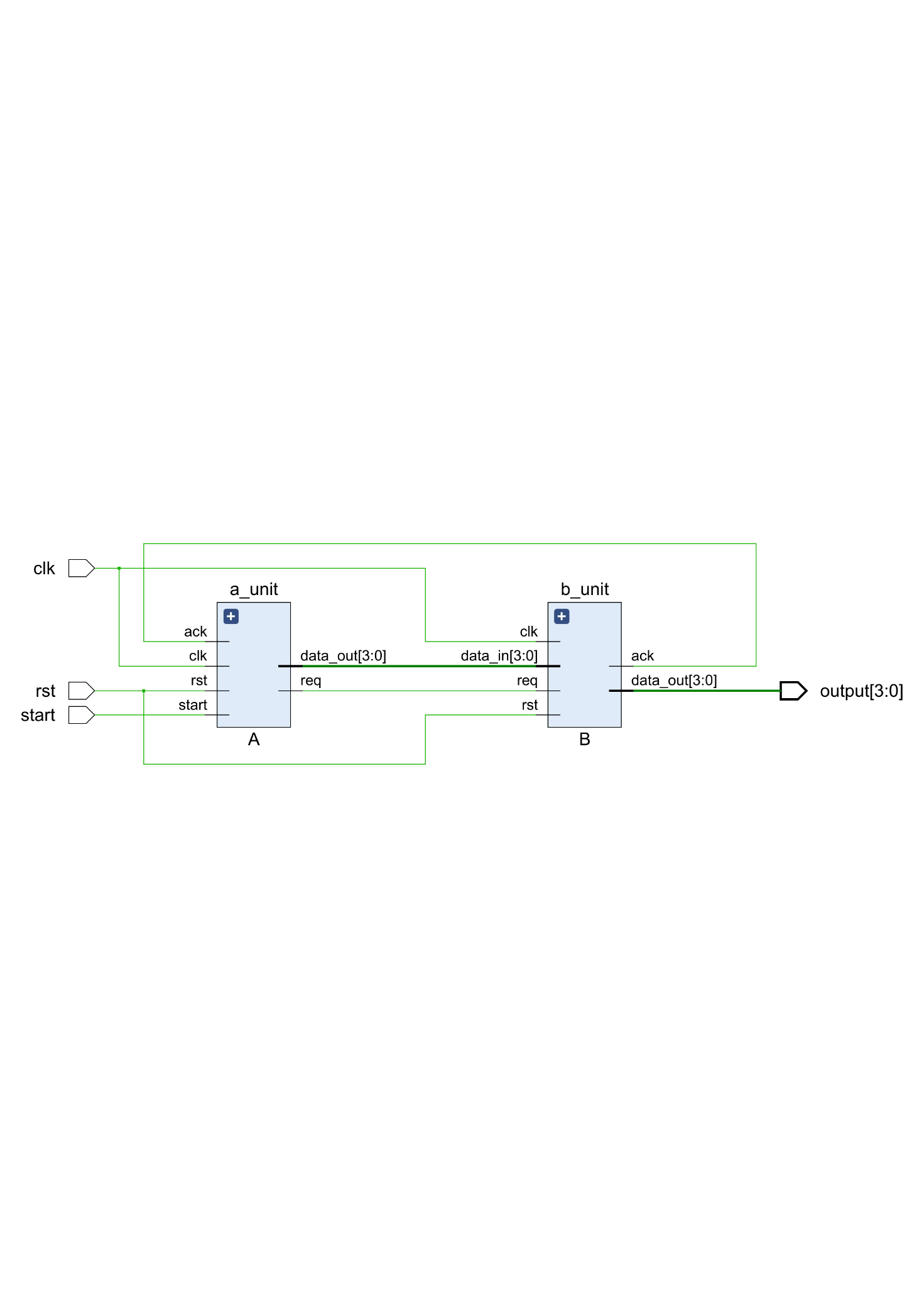
/ACK = 1

/WRITE\_REG\_OUT = 0

/WRITE\_REG\_IN = 0

##### Sistema complessivo

Il sistema complessivo appare nel seguente modo:



#### Implementazione

##### Sistema A

L’unità operativa è composta da un contatore, disponibile in [appendice](#_Contatore_modulo_N), e da una ROM, implementata in maniera analoga [all’esercizio 8.1](#_Sistema_A).

L’unità operativa è la seguente:

entity uo\_a is

Port (

clk: in std\_logic;

rst: in std\_logic;

en\_count: in std\_logic;

read\_rom: in std\_logic;

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

data\_out: out std\_logic\_vector(3 downto 0)

);

end uo\_a;

architecture Structural of uo\_a is

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 ROM\_A is

Generic(N : natural := 8;

M : natural := 8);

Port ( CLK : in std\_logic;

EN : in std\_logic;

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

data\_out : out std\_logic\_vector(M-1 downto 0)

);

end component;

signal address: std\_logic\_vector(2 downto 0);

begin

contatore: contatore\_mod\_N

Generic Map(N=>8)

Port map( clock=>en\_count,

reset=>rst,

set =>'0',

load =>(others=>'0'),

cont =>address

);

rom:ROM\_A

Generic Map(N=> 8,M=>4)

Port Map( CLK =>clk,

EN =>read\_rom,

address =>address,

data\_out=>data\_out

);

count<=address;

end Structural;

L’unità di controllo implementa l’automa a stati finiti:

entity uc\_a is

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

en\_count: out std\_logic;

read\_rom: out std\_logic;

req: out std\_logic;

ack: in std\_logic;

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

);

end uc\_a;

architecture Behavioral of uc\_a is

type stato is (idle,read\_rom\_state,req\_state,wait\_ack,increment);

signal stato\_corrente: stato:=idle;

signal stato\_prossimo: stato;

begin

comb:process(stato\_corrente,start,count,ack)

begin

case stato\_corrente is

when idle=>

en\_count<='0';

if(start='1') then

stato\_prossimo<=read\_rom\_state;

else

stato\_prossimo<=idle;

end if;

when read\_rom\_state=>

en\_count<='0';

read\_rom<='1';

stato\_prossimo<=req\_state;

when req\_state =>

req<='1';

if(ack='1') then

stato\_prossimo<=wait\_ack;

else

stato\_prossimo<=req\_state;

end if;

when wait\_ack =>

req<='0';

if(ack='0') then

stato\_prossimo<=increment;

else

stato\_prossimo<=wait\_ack;

end if;

when increment =>

read\_rom<='0';

en\_count<='1';

if(count="111") then

stato\_prossimo<=idle;

else

stato\_prossimo<=read\_rom\_state;

end if;

end case;

end process;

mem:process(clk)

begin

if(rising\_edge(clk)) then

if(rst='1') then

stato\_corrente<=idle;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

Il Top Module, infine, appare come segue:

entity A is

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

req: out std\_logic;

ack: in std\_logic;

data\_out: out std\_logic\_vector(3 downto 0)

);

end A;

architecture Structural of A is

component uo\_a is

Port (

clk: in std\_logic;

rst: in std\_logic;

en\_count: in std\_logic;

read\_rom: in std\_logic;

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

data\_out: out std\_logic\_vector(3 downto 0)

);

end component;

component uc\_a is

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

en\_count: out std\_logic;

read\_rom: out std\_logic;

req: out std\_logic;

ack: in std\_logic;

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

);

end component;

signal read\_rom: std\_logic;

signal en\_count: std\_logic;

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

begin

uo:uo\_a

Port Map(

clk =>clk,

rst =>rst,

en\_count=>en\_count,

read\_rom=>read\_rom,

count =>count,

data\_out=>data\_out

);

uc:uc\_a

Port Map(

clk =>clk,

rst =>rst,

start =>start,

en\_count=>en\_count,

read\_rom=>read\_rom,

req =>req,

ack =>ack,

count =>count

);

end Structural;

##### Sistema B

Il sistema B utilizza due registri analoghi a quanto visto [nell’esercizio 7.1](#_Implementazione_2).

Il CLA, invece, si basa su due componenti: generazione\_propagazione ecarry\_look\_ahead.

Il componente **generazione\_propagazione** riproduce in output, dati due operandi, le funzioni di generazione e propagazione ed è il seguente:

entity generazione\_propagazione is

Port (

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

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

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

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

);

end generazione\_propagazione;

architecture Dataflow of generazione\_propagazione is

begin

P<= x xor y;

G<= x and y;

end Dataflow;

Il componente **carry\_look\_ahead**, invece, è il componente incaricato di calcolare i riporti, date le funzioni di generazione e propagazione, ed è il seguente:

entity carry\_look\_ahead is

Port (

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

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

c\_in: in std\_logic;

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

c\_out: out std\_logic;

P\_out: out std\_logic\_vector(3 downto 0)

);

end carry\_look\_ahead;

architecture Dataflow of carry\_look\_ahead is

begin

carry(0)<=c\_in;

carry(1)<=G(0) xor (P(0) and c\_in);

carry(2)<=G(1) xor (P(1) and G(0)) xor (P(1) and P(0) and c\_in);

carry(3)<=G(2) xor (P(2) and G(1)) xor(P(2) and P(1) and G(0)) xor (P(2) and P(1) and P(0) and c\_in);

c\_out<=G(3) xor (P(3) and G(2)) xor (P(3) and P(2) and G(1)) xor (P(3) and P(2) and P(1) and G(0)) xor (P(3) and P(2) and P(1) and P(0) and c\_in);

P\_out<=P;

end Dataflow;

Il componente complessivo che implementa il sommatore è il seguente:

entity sommatore\_cla is

Port (

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

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

c\_in: in std\_logic;

c\_out: out std\_logic;

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

);

end sommatore\_cla;

architecture Structural of sommatore\_cla is

component generazione\_propagazione is

Port (

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

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

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

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

);

end component;

component carry\_look\_ahead is

Port (

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

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

c\_in: in std\_logic;

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

c\_out: out std\_logic;

P\_out: out std\_logic\_vector(3 downto 0)

);

end component;

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

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

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

signal cla\_to\_xor : std\_logic\_vector(3 downto 0);

begin

ga: generazione\_propagazione

Port Map(

x =>x,

y =>y,

P =>P\_signal,

G =>G\_signal

);

cla:carry\_look\_ahead

Port Map(

P =>P\_signal,

G =>G\_signal,

c\_in =>c\_in,

carry=>carry\_signal,

c\_out=>c\_out,

P\_out=>cla\_to\_xor

);

s<=cla\_to\_xor xor carry\_signal;

end Structural;

L’unità operativa complessiva appare come segue:

entity uo\_b is

Port (

clk : in std\_logic;

rst : in std\_logic;

data\_in : std\_logic\_vector(3 downto 0);

write\_reg\_in : in std\_logic;

write\_reg\_out : in std\_logic;

data\_out : out std\_logic\_vector

);

end uo\_b;

architecture Structural of uo\_b is

component sommatore\_cla is

Port (

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

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

c\_in: in std\_logic;

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

);

end component;

component reg is

Generic(N: natural:=8);

Port ( CLK : in STD\_LOGIC;

RST : in STD\_LOGIC;

load : in STD\_LOGIC;

input : in STD\_LOGIC\_VECTOR(N-1 downto 0);

output : out STD\_LOGIC\_VECTOR(N-1 downto 0)

);

end component;

signal reg\_in\_signal : std\_logic\_vector(3 downto 0);

signal reg\_out\_signal: std\_logic\_vector(3 downto 0);

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

begin

reg\_in:reg

Generic Map(N=>4)

Port Map( CLK =>clk,

RST =>rst,

load =>write\_reg\_in,

input =>data\_in,

output =>reg\_in\_signal

);

cla:sommatore\_cla

Port Map(

x =>reg\_in\_signal,

y =>reg\_out\_signal,

c\_in=>'0',

s =>ris

);

reg\_out:reg

Generic Map(N=>4)

Port Map( CLK =>clk,

RST =>rst,

load =>write\_reg\_out,

input =>ris,

output =>reg\_out\_signal

);

data\_out<=reg\_out\_signal;

end Structural;

L’unità di controllo, invece, è la seguente:

entity uc\_b is

Port (

clk: in std\_logic;

rst: in std\_logic;

req: in std\_logic;

ack: out std\_logic;

write\_reg\_in : out std\_logic;

write\_reg\_out : out std\_logic

);

end uc\_b;

architecture Behavioral of uc\_b is

type stato is (wait\_req,send\_ack,read,somma);

signal stato\_corrente: stato:=wait\_req;

signal stato\_prossimo: stato;

begin

comb:process(stato\_corrente,req)

begin

case stato\_corrente is

when wait\_req =>

write\_reg\_in<='0';

write\_reg\_out<='0';

if(req='1') then

stato\_prossimo<=send\_ack;

else

stato\_prossimo<=wait\_req;

end if;

when send\_ack =>

ack<='1';

if(req='0') then

stato\_prossimo<=read;

else

stato\_prossimo<=send\_ack;

end if;

when read=>

ack<='0';

write\_reg\_in<='1';

stato\_prossimo<=somma;

when somma =>

write\_reg\_out<='1';

stato\_prossimo<=wait\_req;

end case;

end process;

mem:process(clk)

begin

if(rising\_edge(clk)) then

if(rst='1') then

stato\_corrente<=wait\_req;

else

stato\_corrente<=stato\_prossimo;

end if;

end if;

end process;

end Behavioral;

Il Top Module del nodo è:

entity B is

Port (

clk: in std\_logic;

rst: in std\_logic;

req: in std\_logic;

ack: out std\_logic;

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

data\_out: out std\_logic\_vector(3 downto 0)

);

end B;

architecture Structural of B is

component uo\_b is

Port (

clk : in std\_logic;

rst : in std\_logic;

data\_in : std\_logic\_vector(3 downto 0);

write\_reg\_in : in std\_logic;

write\_reg\_out : in std\_logic;

data\_out : out std\_logic\_vector

);

end component;

component uc\_b is

Port (

clk: in std\_logic;

rst: in std\_logic;

req: in std\_logic;

ack: out std\_logic;

write\_reg\_in : out std\_logic;

write\_reg\_out : out std\_logic

);

end component;

signal write\_reg\_in : std\_logic;

signal write\_reg\_out : std\_logic;

begin

uo:uo\_b

Port Map(

clk =>clk,

rst =>rst,

data\_in =>data\_in,

write\_reg\_in=>write\_reg\_in,

write\_reg\_out=>write\_reg\_out,

data\_out =>data\_out

);

uc:uc\_b

Port Map(

clk =>clk,

rst =>rst,

req =>req,

ack =>ack,

write\_reg\_in =>write\_reg\_in,

write\_reg\_out=>write\_reg\_out

);

end Structural;

##### Sistema complessivo

I due nodi collegati insieme presentano la seguente struttura:

entity system is

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

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

);

end system;

architecture Structural of system is

component A is

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

req: out std\_logic;

ack: in std\_logic;

data\_out: out std\_logic\_vector(3 downto 0)

);

end component;

component B is

Port (

clk: in std\_logic;

rst: in std\_logic;

req: in std\_logic;

ack: out std\_logic;

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

data\_out: out std\_logic\_vector(3 downto 0)

);

end component;

signal req\_signal:std\_logic;

signal ack\_signal:std\_logic;

signal a\_to\_b :std\_logic\_vector(3 downto 0);

begin

a\_unit:A

Port Map(

clk =>clk,

rst =>rst,

start =>start,

req =>req\_signal,

ack =>ack\_signal,

data\_out=>a\_to\_b

);

b\_unit:B

Port Map(

clk =>clk,

rst =>rst,

req =>req\_signal,

ack =>ack\_signal,

data\_in =>a\_to\_b,

data\_out=>output

);

end Structural;

#### Simulazione

La simulazione del sistema prevede semplicemente l’avvio dei segnali di start e reset per verificare il corretto funzionamento:

entity sys\_tb is

end sys\_tb;

architecture Behavioral of sys\_tb is

component system is

Port (

clk: in std\_logic;

rst: in std\_logic;

start: in std\_logic;

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

);

end component;

signal clock :std\_logic;

signal rst :std\_logic;

signal start :std\_logic;

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

signal period: time:= 10 ns;

begin

uut: system

Port Map(

clk =>clock,

rst =>rst,

start =>start,

output=>output

);

cl:process

begin

clock<='0';

wait for period/2;

clock<='1';

wait for period/2;

end process;

stim:process

begin

wait for period;

start<='1';

wait for period;

start<='0';

wait for 60\*period;

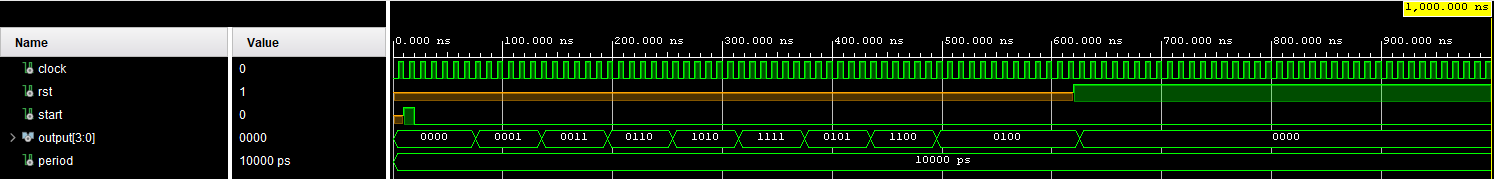
rst<='1';

wait;

end process;

end Behavioral;

È possibile osservare il corretto invio dei dati tra A e B e il corretto funzionamento del reset del sistema:



# **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

Il divisore di frequenza è un componente che, a partire da una certa frequenza di input, produce un’onda con una frequenza minore in output.

Ciò che fa nell’effettivo è avviare un conteggio, che raggiungerà il suo massimo al valore CLKIN\_freq/(CLKOUT\_freq)-1, ovvero il rapporto tra le frequenze.

È un componente molto utile per “rallentare” determinati dispositivi, e generalmente si parte dalla frequenza di clock della board per rallentare proprio il segnale di tempificazione del sistema.

Su ogni fronte di salita del clock in ingresso, il divisore di frequenza inizializza il clock\_out a zero e incrementa una variabile contatore. Non appena la variabile raggiunge il massimo, il clock\_out passa al valore logico alto e così via. In tal modo, si genera un’onda quadra alla frequenza desiderata.

### 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;

## Contatore modulo N

### Progetto e architettura

Il componente in analisi è stato realizzato seguendo un approccio **Behavioral**. Il modulo è stato realizzato mediante un parametro Generic. Il funzionamento prevede rami di if-else per effettuare verifiche sul conteggio, in modo da azzerarlo opportunamente non appena viene raggiunto il modulo.

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.

### Implementazione

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;