Macchine elementari (Rel. 2.0)

Antonino Mazzeo

29 ottobre 2018

Corso di Studi in Ingegneria Informatica

(Attenzione tale capitolo è in bozza e possono esserci degli errori)

0.1 Comparatori

L'operazione di comparazione consiste, in generale, nel confronto fra due dati A e B di un pre assegnato tipo T; il risultato di tale confronto è espresso mediante valori booleani che codificano il risultato della comparazione stessa secondo uno dei criteri d'ordinamento associati a T (ad esempio, $A > B$, $A = B$, $A < B$) associati al tipo T.

L'operazione di comparazione in hardware è effettuata da dispositivi detti, per l'appunto comparatori che, nel caso di verifica della sola condizione di uguaglianza, prendono il nome di *comparatori di uguaglianza*. Da un punto di vista funzionale un comparatore è un dispositivo che ha due ingressi che codificano i dati di tipo T, e una o più uscite booleane, associate alla verifica di una delle condizioni di cui sopra. In fig.0.1 è riportato lo schema di un comparatore d'uguglianza.

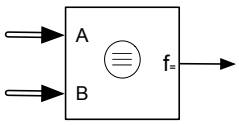


Figura 0.1: Schema di un generico dispositivo comparatore

Il comparatore più elementare è quello in cui i dati A e B sono di tipo booleano e rappresentati con segnali logici in ingresso che chiameremo x e y, così come indicato in fig.0.2. In tab.0.1 è riportata la tabella di verità relativa alle tre funzioni booleane di uguaglianza, maggiore e minore, indicate rispettivamente con $f_=$, $f_>$, e $f_<$.

La $f_=$ è data dalla nota funzione di uguaglianza $x \equiv y$; la $f_>$è data dal prodotto $x\overline{y}$; la $f_<$ dal prodotto $\overline{x}y$. Il caso $x \neq y$ è dato dalla verifica di una delle due condizioni $f_>$ e $f_<$, ovvero dalla negata della condizione di equivalenza che equivale allo Or-esclusivo $\overline{x \equiv y} = x \oplus y$.

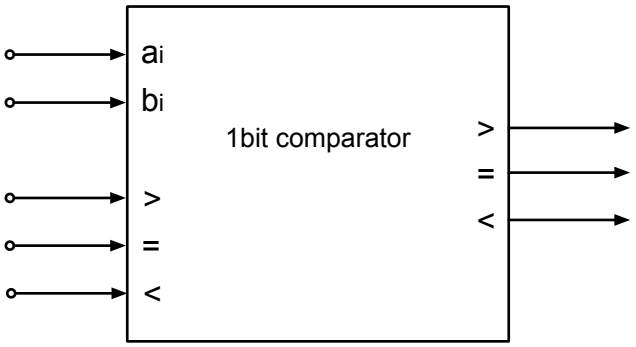


Figura 0.2: Comparatore binario

x	y	f<	f=	f>
0	0		1	
0	1	1		
1	0			1
1	1		1	

$f_{=} = x \equiv y = \overline{xy} + xy;\ f_{<} = \overline{x}y;\ f_{>} = x\overline{y};\ f_{\neq} = x \oplus y = \overline{x}y + x\overline{y}$

Tabella 0.1: Tabella di verità di un comparatore binario

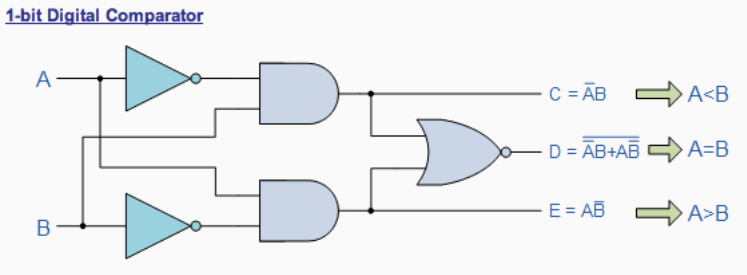


Figura 0.3: Realizzazione circuitale del comparatore a 1-bit

Lo schema del comparatore a 1-bit basato sull'uso di una porta NXOR è mostrato in fig.0.3.

In generale i tipi di appartenenza dei dati A e B sono codificati con una stringa di n bit. Indicando con a_i e b_i il generico bit di ciascuna delle due stringhe, A e B sono rappresentati con due vettori di n elementi binari: $A[a_{n-1}...a_0]$ e $B[b_{n-1}...b_0]$.

La condizione di uguaglianza di A e B è, quindi, data dal verificarsi dell'uguaglianza di ciascuna delle n coppie di bit a_i e b_i:

$$A \equiv B \Rightarrow (a_{n-1} \equiv b_{n-1}) \cdot (a_{n-2} \equiv b_{n-2}) \cdots (a_0 \equiv b_0) = \tag{0.1}$$
$$= \overline{(a_{n-1} \oplus b_{n-1})} \cdot \overline{(a_{n-2} \oplus b_{n-2})} \cdots \overline{(a_0 \oplus b_0)}$$

Dalla 0.1 deriva l'architettura del comparatore d'uguaglianza di cui alla fig.0.4.

La 0.1 può essere riscritta in forma iterativa osservando che la verifica della condizione di uguaglianza per una generica coppia di bit in posizione i-esima implica, ai fini della determinazione della verifica della condizione di uguaglianza di A e B, la verifica della condizione di uguaglianza per tutte le precedenti coppie di bit da i-1 a 0. Il confronto i-esimo può pertanto essere scritto come:

$$CO_i = (a_i \oplus b_i) \cdot CI_i \qquad con\ CI_i = CO_{i-1} = (a_{i-1} \oplus b_{i-1}) \cdot CI_{i-1} e CO_0 = (a_0 \oplus b_0) \cdot 1 \tag{0.2}$$

Da tale espressione iterativa deriva l'architettura del comparatore con propagazione dei risultati delle comparazioni precedenti riportata in fig.0.5b e la cui cella i-esima è riportata in fig.0.5a.

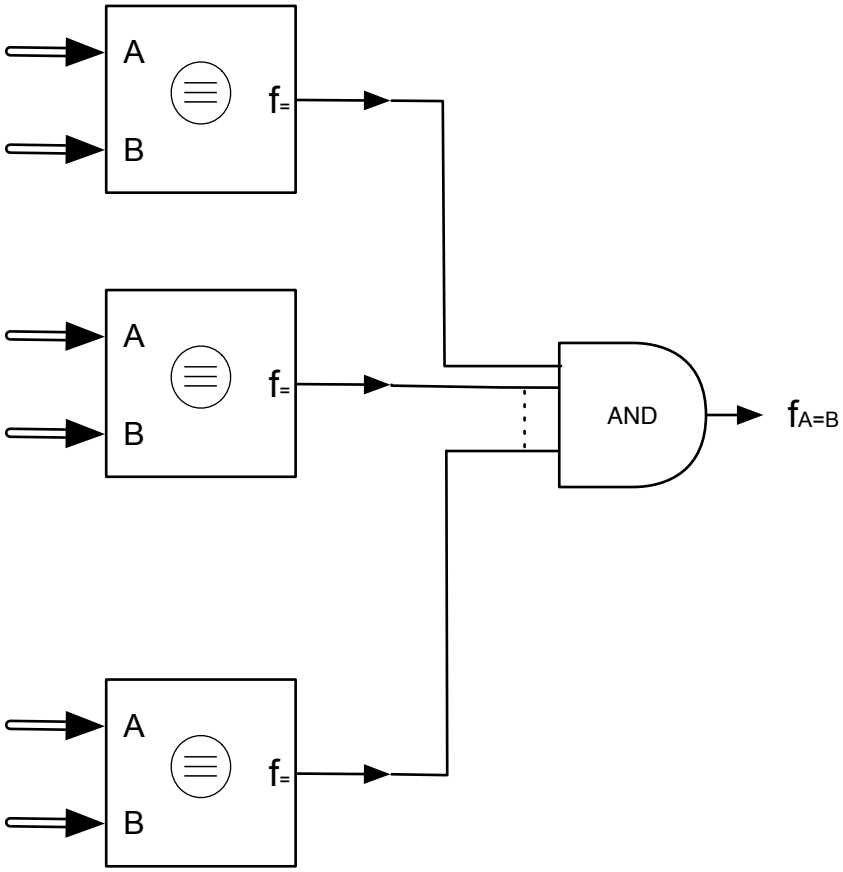
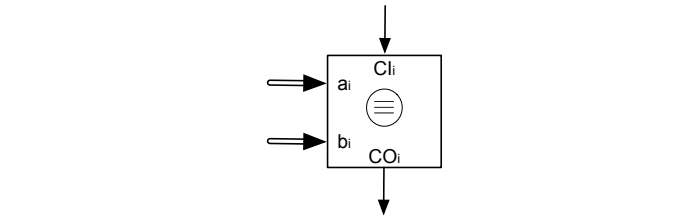


Figura 0.4: Comparatore d'uguaglianza realizzato con l'uso di n operatori di uguaglianza e una porta and a n ingressi

La cella di fig.0.5a, basata sulla propagazione del solo risultato delle comparazione di uguaglianze dalla posizione i=0 all'ultimo stadio i=n-1, può essere generalizzata con l'aggiunta di due ulteriori linee per la propagazione della coondizione di maggioranza e minoranza. E' ovvio che dai risultati della comparazione di uguaglianza e maggioranza (o minoranza) si può derivare quella di minoranza o di maggioranza, infatti, disponendo dei due risultati booleani della comparazione f= e $f_>$, il caso $f_<$ è dato dall'espressione logica $f_< = \overline{f_= + f_>}$ (se si dispone di f<, in modo analogo si avrebbe $f_> = \overline{f_= + f_<}$). Basterebbero pertanto due sole linee di propagazione per determinare tutti i casi di una comparazione $(<, >, =, \neq)$.

Come detto all'inizio del paragrafo, la comparazione fra due dati dipende, in generale, dalla codifica adottata per rappresentare il tipo a cui essi appartengono. Le soluzioni sopra discusse si riferiscono esclusivamente alla comparazione di stringhe di bit che rappresentano interi senza segno e in cui il confronto è effettuato comparando tutte le coppie di bit a partire dalla posizione i=n-1 a quella i=0, avendo i bit più significativi un peso maggiore ai fini della comparazione (il confronto fra i bit in posizione più significativa



(a) Cella base

(b) Comp. A=B

Figura 0.5: Comparatore con propagazione dei risultati delle comparazioni precedenti

N_{10}	7	6	5	4	3	2	1	**0**
$R_2(N)$	0111	0110	0101	0100	0011	0010	0001	**0000**
$C_1[R_2(N)]$								
$C_2[R_2(N)]$								

N_{10}	-1	-2	-3	-4	-5	-6	-7	-8
$R_2(N)$	1111	1110	1101	1100	1011	1010	1001	1000
$C_1[R_2(N)]$	0000	0001	0010	0011	0100	0101	0110	0111
$C_2[R_2(N)]$	0001	0010	0011	0100	0101	0110	0111	1000

Tabella 0.2: Rappresentazioni degli interi di 4 bit

rispetto a quelli che seguono, determina il risultato del confronto se non vale la condizione di uguaglianza. Se essi risultano uguali, occorre procedere con il confronto della coppia di bit meno significativa.

Se ad esempio si operasse con dati interi relativi rappresentati in complemento a 2, la comparazione dovrebbe tenere conto del segno dei due operandi A e B. Si ricorda che nella rappresentazione modulo e segno con numeri N di n bit, gli interi positivi sono rappresentati da stringhe di bit aventi il bit $b_{n-1}=0$ mentre quelli negativi hanno tale bit a 1. Inoltre, essendo asimmetrico l'intervallo numerico di rappresentazione dei numeri positivi e negativi (2^{n-1} numeri positivi e $2^{n-1}+1$ numeri negativi), con lo zero rappresentato con tutti i bit posti a 0, si conviene di assegnare all'intervallo negativo l'elemento con rappresentazione avente tutti i bit a 1, a cui corrisponde il primo valore dell'intervallo dei negativi -1. In tab.0.2, ad esempio, è riportata di seguito la rappresentazione su di un registro di 4 bit dei $2^4=16$ numeri interi relativi (7 positivi e 8 negativi). Per i numeri negativi da -1 a -8 è anche indicato il valore assegnato interpretando la stringa come rappresentazione in complementi alla base diminuita (complementi a 1), indicata come $C_1[R_2(N)]$, e come complementi a 2, indicata come $C_2[R_2(N)]$ e ottenuta, come è noto,

sommando 1 alla corrispondenza in complemento a 1.

In tab.0.3 sono riportate, infine, le rappresentazioni in complementi alla base, in complementi diminuiti e per eccesso dei 16 interi relativi rappresentabili con 4 bit.

N10	$R_2(N_2)$	$R_1(N_2)$	Ecc($+8_{10}$=1000_2)
-8	1000	n.r.	0000
-7	1001	1000	0001
-6	1010	1001	0010
-5	1011	1010	0011
-4	1100	1011	0100
-3	1101	1100	0101
-2	1110	1101	0110
-1	1111	1110	0111
-0	0000	1111	1000
+0	0000	0000	1000
1	0001	0001	1001
2	0010	0010	1010
3	0011	0011	1011
4	0100	0100	1100
5	0101	0101	1101
6	0110	0110	1110
7	0111	0111	1111

Tabella 0.3: Rappresentazioni in complemento a 2, in complemento a 1 e per eccessi degli interi relativi codificati con 4 bit

Supponendo di operare in complemento a 2, invece di effettuare una comparazione fra A e B ricorrendo a un confronto bit a bit come sopra mostrato, si può effettuare il confronto mediante un'operazione di sottrazione di A da B seguita da un test del risultato con zero come di seguito indicato:
$C = (A - B)$

1. $IF(C > 0)\, than\, A > B$
2. $IF(C < 0)\, than\, A < B$
3. $IF(C = 0)\, than\, A = B$

Tale soluzione richiede, pertanto, l'uso di un adder/subtractor e il confronto in complemento a 2 di C con 0. Tale confronto è facilmente realizzabile, infatti, stante la rappresentazione in complemento a 2 di C data da $C[c_{n-1}...c_0]$, in corrispondenza di ciascuno dei tre casi di cui sopra si ha:

1. C>0 se c_{n-1}= 0 and for i = 0..n-2 esista almeno un c_i=1, ovvero se $\overline{c}_{n-1} \cdot (c_{n-2} + c_{n-3} + ... + c_0) = 1$

2. C<0 se c_{n-1}=1

3. C=0 se c_i= 0 for i=0..n-1, ovvero se $\overline{c}_{n-1} \cdot \overline{c}_{n-2} \cdot \ldots \cdot \overline{c}_0 = \overline{c_{n-1} + c_{n-2} + \ldots + c_0} = 1$

In fig.0.6 è riportato il circuito di un tale comparatore.

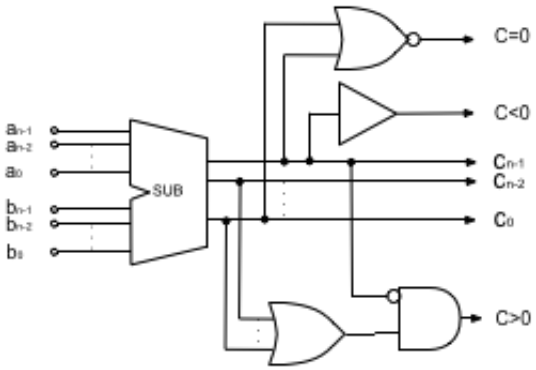


Figura 0.6: Comparatore realizzato con l'uso di un sottrattore

---

0.2 Laboratorio: simulazione e sintesi di un comparatore di stringhe di n bit

Descrivere in VHDL, sia in modo comportamentale sia data flow, un comparatore di stringhe a 1 bit, farne la verifica di correttezza mediante simulazione nell'ambiente Xilinx ISE e valutarne i ritardi. Ai fini della verifica, progettare un test-banch in grado di testare il componente in esame garantendo un'adeguata copertura.

Progettare, quindi, un magnitude comparator per confrontare stringhe di 4 bit, analogo a quello presente nella libreria TTL 7485 e mostrato in fig.0.8e, a partire dal magnitude

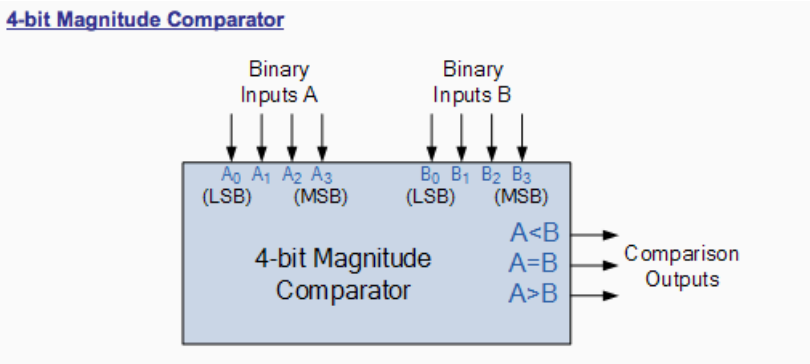


Figura 0.7: Magnitude Comparator a 4 bit

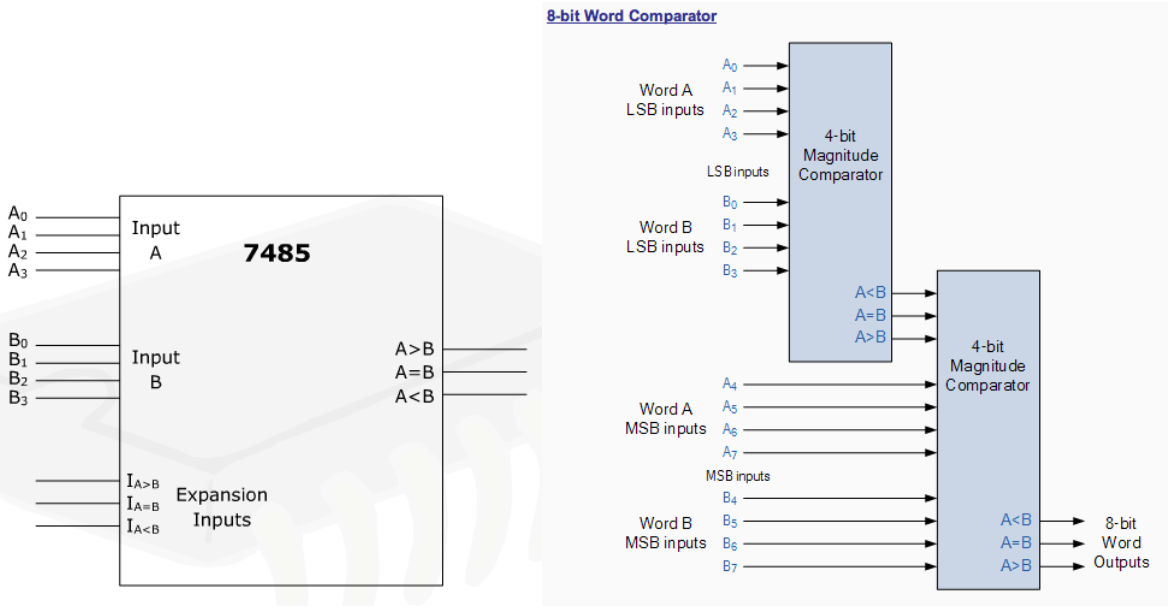


Figura 0.8: TTL 7485 8bit-comparator

comparator a 4 bit, considerato come un componente di una libreria VHDL, progettarne uno a 8 bit, così come mostrato in fig.0.9.

Creare, infine, il banco di prova (test bed) per verificare la correttezza di tutte le funzionalità (si veda fig.0.9).

0.2.1 Spunti per il progetto

Il componente elementare *1-bit cascadable comparator* (*1-bcc*) è un comparatore elementare a 1 bit, analogo funzionalmente al componente delle famiglie TTL/CMOS *4 bit magnitude comparator 7485*.

Tale componente, una volta descritto in VHDL e verificato nel funzionamento, dovrà essere instanziato n volte per realizzare il comparatore di stringhe di n bit richiesto. Il blocco funzionale che descrive le interfacce di ingresso e uscita di *1-bcc* è mostrato in fig.0.10.

In tab.0.4 è mostrata la tabella di verità con 5 ingressi e tre uscite di 1-bcc. Le equazioni booleane delle tre uscite sono riportate in fig.0.11.

Se, a partire da *1-bcc* si vuole costruire, ad esempio, un comparatore a due bit basta collegare due *1-bcc* in cascata ponendo:

$$(A_{i-1} > B_{i-1}, \quad A_{i-1} = B_{i-1}, \quad A_{i-1} < B_{i-1}) = (1,\ 0,\ 1)$$

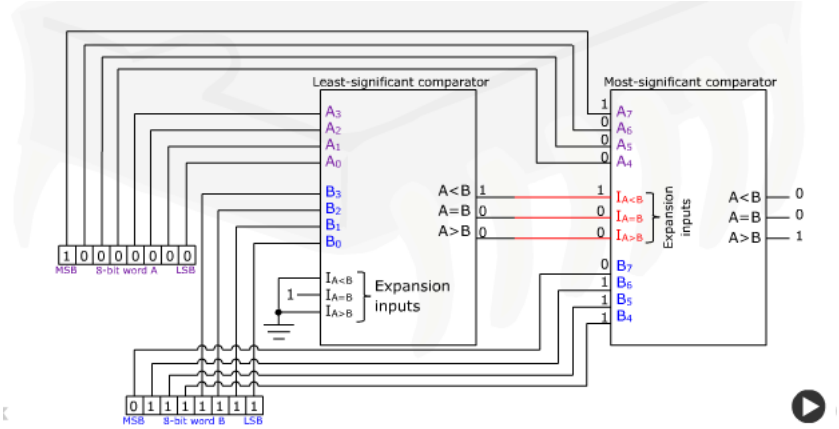


Figura 0.9: Registri che alimentano il TTL 7485

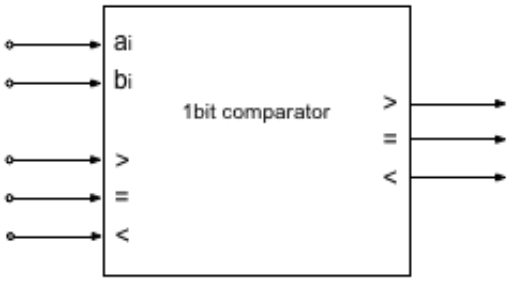


Figura 0.10: Schema funzionale del comparatore a un bit 1-bcc

a_i	b_i	$a_{i-1} > b_{i-1}$	$a_{i-1} = b_{i-1}$	$a_{i-1} < b_{i-1}$	$f_>$	$f_=$	$f_<$
-	-	1	0	0	1	0	0
-	-	0	0	1	0	0	1
0	0	0	1	0	0	1	0
0	1	0	1	0	0	0	1
1	0	0	1	0	1	0	0
1	1	0	1	0	0	1	0

Tabella 0.4: Tabella verità di *1-bcc*

$$f_> = GT + (A_i \cdot \overline{B})$$
$$(f_= = (EQ \cdot A_i \cdot \ B_i) + (\bar{A}_i \cdot \bar{B}_i \cdot EQ)$$
$$f_< = LT + (B_i \cdot \overline{A_i})$$

Figura 0.11: Equazioni booleane delle tre uscite di *1-bcc*

Si noti ancora che la funzione $(A_i < B_i)$ si può ricondurre al valore di $(A_i > B_i)$ negato: questa considerazione potrebbe ridurre il numero di porte necessarie al circuito, ma aggiungerebbero un livello di logica.

0.3 Codice VHDL

Sono riportati di seguito degli esempi di codici VHDL per descrivere il comparatore a 1 bit e quello a 4 bit sopra specificati.

0.3.1 Bit Comparator

Listing 1: Definizione del componente Bit Comparator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bit_comparator is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           aEQ_INb : in  STD_LOGIC;
           aLT_INb : in  STD_LOGIC;
           aGT_INb : in  STD_LOGIC;
           aEQb : out  STD_LOGIC;
           aLTb : out  STD_LOGIC;
           aGTb : out  STD_LOGIC);
end bit_comparator;

architecture Dataflow of bit_comparator is
begin
aEQb<=(aEQ_INb and a and b) or (not(a) and not(b) and aEQ_INb);
aGTb <= aGT_INb or (a and not b);
aLTb <= aLT_INb or (not a and b);
end Dataflow;
```

0.3.2 Bit String Comparator

Il componente Top Module è ottenuto con costrutto *Generate* e le connessioni sono fatte sfruttando tre array monodimensionali (aGT, aLT, aEQ).

I valori MSB di questi array sono posti a (0,0,1) rispettivamente per garantire che sui bit più significativi della stringa in ingresso sia fatta una comparazione assoluta, e non relativa.

Listing 2: Definizione del componente Bit String Comparator Generic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bit_string_comparator is
```

```
GENERIC(N: Integer:=4);
    Port ( a : in  STD_LOGIC_VECTOR (N-1 downto 0);
           b : in  STD_LOGIC_VECTOR (N-1 downto 0);
           aGTb : out  STD_LOGIC;
           aEQb : out  STD_LOGIC;
           aLTb : out  STD_LOGIC);
end bit_string_comparator;

architecture Structural of bit_string_comparator is

component bit_comparator_comp
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           aEQ_INb : in  STD_LOGIC;
           aLT_INb : in  STD_LOGIC;
           aGT_INb : in  STD_LOGIC;
           aEQb : out  STD_LOGIC;
           aLTb : out  STD_LOGIC;
           aGTb : out  STD_LOGIC);
end component bit_comparator_comp;

for all: bit_comparator_comp use entity WORK.bit_comparator(Dataflow);

--Array di GT, LT ed EQ per la propagazione
signal aGT, aLT, aEQ: STD_LOGIC_VECTOR(N downto 0);
begin
aGT(N) <='0';
aLT(N) <='0';
aEQ(N) <='1'; --Altrimenti non vengono riconosciute mai come uguali!!!

aGTb<=aGT(0);
aEQb<=aEQ(0);
aLTb<=aLT(0);

one_bcc: for i in N-1 downto 0 GENERATE begin
        bit_comparator: bit_comparator_comp port map(a(i), b(i), aEQ(i+1),
                                            aLT(i+1), aGT(i+1),aEQ(i), aLT(i),
                                                aGT(i));
end GENERATE one_bcc;

end Structural;
```

0.3.3 Area e Tempi di propagazione

In base ai risultati della simulazione effettuata nell'ambiente ISE di XILINX (fig.0.12), si sono verificati i valori temporali che caratterizzano tali componenti. In tab.0.5 sono indicati i valori calcolati post synthesis con parametri *xc3s1200e-4fg320* e calcolati Pad to Pad.

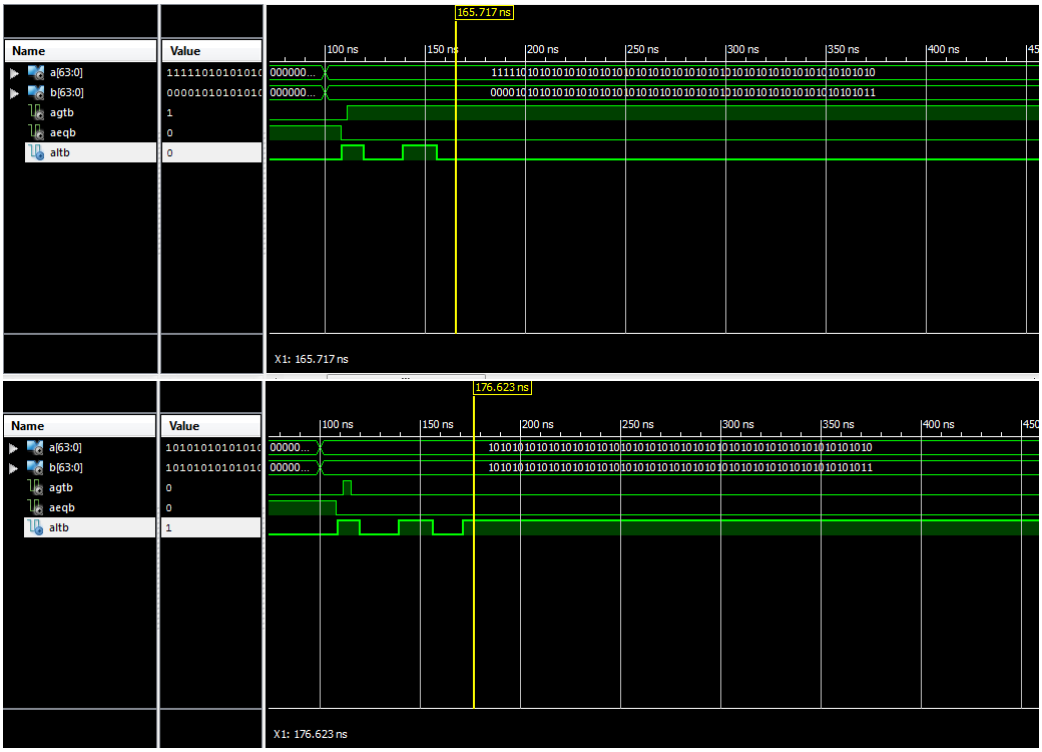


Figura 0.12: Screenshots di Simulazione del Comparator a 64 bit

Numero di Bit	Tempo Min	Tempo Max	Area
4	6.717 ns	9.496 ns	9 LUTs, 5 Slices, 11 IOBs
8	7.018 ns	13.477 ns	19 LUTs, 11 Slices, 19 IOBs
16	7.373 ns	19.616 ns	38 LUTs, 19 Slices, 35 IOBs
32	7.692 ns	39.879 ns	86 LUTs, 46 Slices, 67 IOBs
64	7.695 ns	77.692 ns	178 LUTs, 99 Slices, 131 IOBs

Tabella 0.5: Valori temporali e di occupazione d'area

A	B	GT	LT	EQ
0000	0000	0	0	1
1010	0101	1	0	0
1101	1111	0	1	0

Tabella 0.6: Test di verifica del comparatore

0.3.4 Test

In tab. sono elencati alcuni casi di test eseguiti su operandi a 4 bit ciascuno.