$_{L}OGIC, STD_{L}OGIC_{V}ECTOR, keywordstyle =$
$[1]blue, keywordstyle =$
$[2]teal, sensitive =$
$true, morecomment =$
$[l]--, morestring =$
$[b]"$
,
```
numbers=left,
numberstyle=gray,
keywordstyle=blue,
commentstyle=green!60!black,
stringstyle=orange,
breaklines=true,
showstringspaces=false,
tabsize=2,
xleftmargin=2em
```

Carry Select Adder a 16 bit

Relazione di progetto

Studenti:

Frega Umberto 239527, frgmrt04a05l353d@studenti.unical.it`frgmrt04a05l353d@studenti.unical.it`;
Napoli Leonardo 234364, npllrd02s30d086@studenti.unical.it `npllrd02s30d086@studenti.unical.it`;
https://github.com/Zi0LEO/elettronica$_{d}igitale$`Codice Sorgente`

Il progetto assegnato consiste nel progettare ed implementare un carry select adder a 16 bit tramite linguaggio VHI
Il primo passo della progettazione è stato definire la modularizzazione dell'adder tramite adder più semplici a mino
Full-Adder Implementazione Come componente di base del sistema si è optato per un semplice full-adder. A causa
Codice Full-Adderproblem-label [language=VHDL] library IEEE; use IEEE.STD$_{L}OGIC_{1}$164.ALL;
entity fulladder$_{1}bitisPort(A : inSTD_{L}OGIC; B : inSTD_{L}OGIC; carry_{i}n : inSTD_{L}OGIC; carry_{o}ut : outSTD_{L}OG$
architecture Behavioral of fulladder$_{1}bitissignalp : STD_{L}OGIC; beginp <= AxorB; carry_{o}ut <= Awhenp =' 0'elsec$
end Behavioral;

Schematica Il codice precedente ha generato in Vivado la schematica riportata in *Figure 1.* Da notare la creazione d
[h] [width=15cm]resources/fulladder.png Circuito Logico del Full Adder
Testbench Essendo presenti 3 entrate e quindi sole $2^3 = 8$ possibli combinazioni, si è deciso di testare ogni caso poss
Test Full-Adder [language=VHDL] library IEEE; use IEEE.STD$_L OGIC_1$164.ALL;
entity testbench$_f ulladderisendtestbench_f ulladder$;
architecture Behavioral of testbench$_f ulladderiscomponentfulladder_1bitisPort(A : inSTD_LOGIC; B : inSTD_LOG$
signal Ia,Ib,Icin,Ocout,Osum: STD$_L OGIC$; $beginCUT : fulladder_1bitportmap(Ia, Ib, Icin, Ocout, Osum); processbe$
–Test 2: A = 0, B = 0, carry$_i n = 1 Ia <=' 0'; Ib <=' 0'; Icin <=' 1'; waitfor10ns; assert(Osum =' 1' and Ocout ='$
–Test 3: A = 1, B = 0, carry$_i n = 0 Ia <=' 1'; Ib <=' 0'; Icin <=' 0'; waitfor10ns; assert(Osum =' 1' and Ocout ='$
–Test 4: A = 1, B = 0, carry$_i n = 1 Ia <=' 1'; Ib <=' 0'; Icin <=' 1'; waitfor10ns; assert(Osum =' 0' and Ocout ='$
–Test 5: A = 0, B = 1, carry$_i n = 0 Ia <=' 0'; Ib <=' 1'; Icin <=' 0'; waitfor10ns; assert(Osum =' 1' and Ocout ='$
–Test 6: A = 0, B = 1, carry$_i n = 1 Ia <=' 0'; Ib <=' 1'; Icin <=' 1'; waitfor10ns; assert(Osum =' 0' and Ocout ='$
–Test 7: A = 1, B = 1, carry$_i n = 0 Ia <=' 1'; Ib <=' 1'; Icin <=' 0'; waitfor10ns; assert(Osum =' 0' and Ocout ='$
–Test 8: A = 1, B = 1, carry$_i n = 1 Ia <=' 1'; Ib <=' 1'; Icin <=' 1'; waitfor10ns; assert(Osum =' 1' and Ocout ='$
end process; end Behavioral;

Simulazione Il risultato della testbench sopra indicata è illustrato nella seguente simulazione: [ht] [width=16cm]resources/fulladder$_s$$im.pngSimulazioneFullAdder$

Ripple-Carry Adder a 4 bit Implementazione L'addizionatore a propagazione del riporto conta al suo interno 4 full
Codice Ripple Carry [language=VHDL] library IEEE; use IEEE.STD$_LOGIC_1164.ALL$;
entity ripplecarry$_4 bit is Port(A_4 : in STD_LOGIC_VECTOR(3 downto 0); B_4 : in STD_LOGIC_VECTOR(3 downto 0); ca$
architecture Behavioral of ripplecarry$_4 bit is component full adder_1 bit is Port(A : in STD_LOGIC; B : in STD_LOGIC; c$
signal sum, carry: STD$_LOGIC_VECTOR(3 downto 0)$; $begin fa0 : full adder_1 bit PORT MAP(A_4(0), B_4(0), carry_in, c$
Schematica [H] [width=15cm]resources/ripplecarry.png Circuito Logico del Ripple Carry Adder
TestBench Nella fase di testing essendo impossibile testare ogni caso sono stati implementati i test soltanto di alcu
Codice Ripple Carry [language=VHDL] library IEEE; use IEEE.STD$_LOGIC_1164.ALL$;
entity testbench$_ripplecarry is end testbench_ripplecarry$;
architecture Behavioral of testbench$_ripplecarry is component ripplecarry_4 bit is Port(A_4 : in STD_LOGIC_VECTOR(3$
signal Ia,Ib,Osum: std$_logic_vector(3 downto 0)$; $signal Icin, Ocout : std_logic$;
begin CUT: ripplecarry$_4 bit port map(Ia, Ib, Icin, Ocout, Osum)$; $process begin --Test 1 : caso base, A = 0, B = 0, car$
–Test 2: A = 1, B = 1, carry = 0 Ia ¡= "0001"; Ib ¡= "0001"; Icin ¡= '0'; wait for 10ns; assert (Osum = "0010" an
–Test case 3: A = 0101, B = 0011, carry$_in = 0 Ia <= "0101"; Ib <= "0011"; Icin <=' 0'; wait for 10 ns; assert (Osu$
– Test case 4: A = 1111, B = 0001, carry$_in = 0 Ia <= "1111"; Ib <= "0001"; Icin <=' 0'; wait for 10 ns; assert (Osu$
– Test case 5: Ia = 1111, Ib = 1111, cin = 0 Ia ¡= "1111"; Ib ¡= "1111"; Icin ¡= '0'; wait for 10 ns; assert (Osum =
– Test case 6: Ia = 1010, Ib = 0101, cin = 1 Ia ¡= "1010"; Ib ¡= "0101"; Icin ¡= '1'; wait for 10 ns; assert (Osum =
– Test case 7: Ia = 1001, Ib = 1001, cin = 0 Ia ¡= "1001"; Ib ¡= "1001"; Icin ¡= '0'; wait for 10 ns; assert (Osum =
– Test case 8: Ia = 0110, Ib = 1001, cin = 1 Ia ¡= "0110"; Ib ¡= "1001"; Icin ¡= '1'; wait for 10 ns; assert (Osum =
end process; end Behavioral;
Simulazione Il risultato della testbench sopra indicata è illustrato nella seguente simulazione: [h] [width=16cm]reso

Carry-Select Adder Implementazione Nella fase iniziale vengono istanziati i vari addizionatori Ripple Carry con le l
Codice Carry Select [language=VHDL] library IEEE; use IEEE.STD$_LOGIC_1$164.ALL;
entity carry$_select_1$6$bit is Port(A : in STD_LOGIC_VECTOR(15 downto 0); B : in STD_LOGIC_VECTOR(15 downto 0); c$
architecture Behavioral of carry$_select_1$6bit
component ripplecarry$_4$$bit is Port(A_4 : in STD_LOGIC_VECTOR(3 downto 0); B_4 : in STD_LOGIC_VECTOR(3 downto$
shared variable carry$_select$: STD_LOGIC; $signal carry_start$: STD_LOGIC; $signal carry0, carry1 : STD_LOGIC_V$
begin ripplecarry0$_0$: $ripplecarry_4bit PORT MAP(A_4 => A(3 downto 0), B_4 => B(3 downto 0), carry_in => carry_in_s$
ripplecarry1$_0$: $ripplecarry_4bit PORT MAP(A_4 => A(7 downto 4), B_4 => B(7 downto 4), carry_in =>' 0', carry_out =>$
ripplecarry2$_0$: $ripplecarry_4bit PORT MAP(A_4 => A(11 downto 8), B_4 => B(11 downto 8), carry_in =>' 0', carry_out =$
ripplecarry3$_0$: $ripplecarry_4bit PORT MAP(A_4 => A(15 downto 12), B_4 => B(15 downto 12), carry_in =>' 0', carry_ou$
process(carry$_start$, $sum_0, sum_1, carry_start$) $begin case carry_start is when '0' => sum(7 downto 4) <= sum_0(3 downto 0)$
case carry$_select$ $is when '0' => sum(11 downto 8) <= sum_0(7 downto 4); carry_select := carry0(1); when others =>$
case carry$_select$ $is when '0' => sum(15 downto 12) <= sum_0(11 downto 8); carry_select := carry0(2); when others =$
entity testbench$_c$ $arryselect is end testbench_carryselect$;
architecture Behavioral of testbench$_c$ $arryselect is component carry_select_1$6$bit is Port(A : in STD_LOGIC_VECTOR(15$
process begin – Test 1: A = 0000000000000000, B = 0000000000000000, carry$_in$ $= 0 Ia <= (others =>' 0'); Ib <=$
– Test 2: A = 0000000000000000, B = 0000000000000000, carry$_in$ $= 1 Ia <= (others =>' 0'); Ib <= (others =>' 0$
– Test 3: A = 0111111111111111, B = 0000000000000001, carry$_in$ $= 0 Ia <=$ "0111111111111111"; $--$ 32767$Ib <=$
– Test 4: A = 0111111111111111, B = 0111111111111111, carry$_in$ $= 0 Ia <=$ "0111111111111111"; $--$ 32767$Ib <=$
– Test 5: A = 0000000000000001, B = 0000000000000001, carry$_in$ $= 1 Ia <=$ "0000000000000001"; $--$ $1 Ib <=$ "00
– Test 6: A = 0000000000000000, B = 1111111111111111, carry$_in$ $= 0 Ia <=$ "0000000000000000"; $Ib <=$ "1111111
– Test 7: A = 1111111111111111, B = 1111111111111111, carry$_in$ $= 1 Ia <=$ "1111111111111111"; $--$ 65535$Ib <=$
– Test 8: A = 1111111111111111, B = 0000000000000001, carry$_in$ $= 0 Ia <=$ "1111111111111111"; $--$ 65535$Ib <=$
–Test Finiti report "Tutti i test sono stati completati con successo!" severity note; end process; end Behavioral; Si