Mini ALU

Relazione di progetto

Studenti:
Frega Umberto 239527, frgmrt04a051353d@studenti.unical.it;
Napoli Leonardo 234364, npllrd02s30d086@studenti.unical.it;

Codice Sorgente

Il progetto assegnato consiste nel progettare ed implementare una mini alu, capace di fare addizioni e sottrazioni, tramite linguaggio VHDL. Per la progettazione del sistema si è deciso di utilizzare un pattern comportamentale, andando quindi a definire il comportamento del sistema in base a determinate condizioni, oltretutto si è optato per l'utilizzo del tipo *STD_LOGIC* e quindi *STD_LOGIC_VECTOR* per una maggiore flessibilità e maggiori funzionalità.

Il primo passo della progettazione è stato definire la politica tramite la quale la mini ALU potesse cambiare tra addizione e sottrazione. A questo proposito si è deciso di mantenere un singolo adder, ma cambiare il segno del secondo operando.

1 Adder

1.1 Implementazione

La componente di base del sistema è un carry look-ahead adder, che genera quindi vari segnali generate e propagate a seconda del numero di bit degli operandi. Riportiamo di seguito il codice dell'adder con caso di default con 4 bit.

1: Codice carry look-ahead adder

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

entity generic_adder is
    generic (bit_number : INTEGER := 4);
    Port ( A_adder,B_adder: in STD_LOGIC_VECTOR (bit_number-1 downto 0);
         cin : in STD_LOGIC;
         sum : out STD_LOGIC_VECTOR (bit_number downto 0));
end generic_adder;

architecture Behavioral of generic_adder is
    signal p,g : STD_LOGIC_VECTOR (bit_number downto 0);
    signal carry : STD_LOGIC_VECTOR (bit_number+1 downto 0);
begin
    carry(0) <= cin
    p_g: for i in 0 to bit_number generate
        p_gMSB: if (i=bit_number) generate
            p(i) <= A_adder(bit_number-1) xor B_adder(bit_number-1);
            g(i) <= A_adder(bit_number-1) and B_adder(bit_number-1);
        end generate;
        p_gLSB: if i<bit_number generate
            p(i) <= A_adder(i) xor B_adder(i);
```

```
24              g(i) <= A_adder(i) and B_adder(i);
25          end generate;
26          carry(i+1) <= (g(i) or (p(i) and carry(i)));
27          sum(i) <= carry(i) xor p(i);
28      end generate;
29  end Behavioral;
```

1.2 Schematica

Il codice precedente con bit number 4, 8 e 16 ha generato in vivado le schematiche riportate rispettivamente in *Figure 1*, *Figure 2*, *Figure 3*.

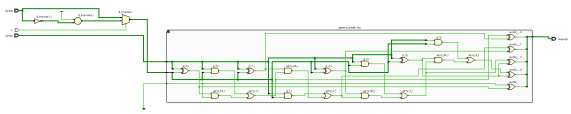

Figure 1: Adder a 4 bit

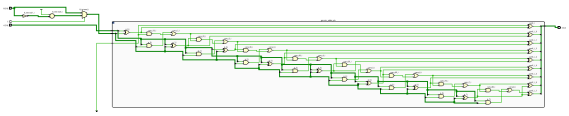

Figure 2: Adder a 8 bit

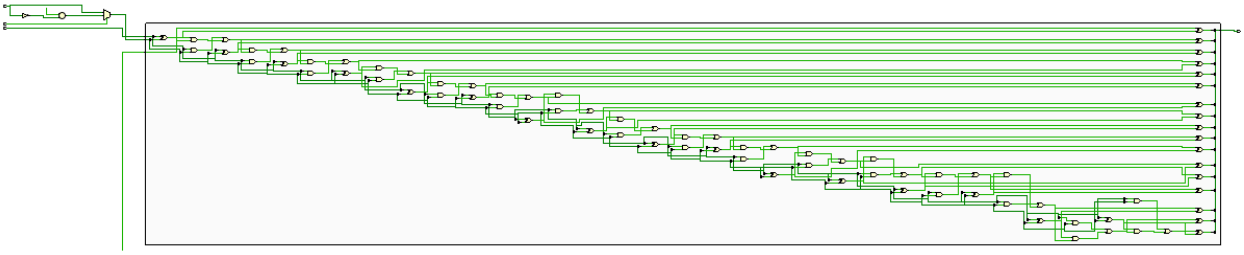

Figure 3: Adder a 16 bit

2 Mini ALU

2.1 Implementazione

La mini ALU progettata presenta al suo interno un solo adder, preceduto da un multiplexer, che in base al bit di controllo C decide se dare in output B oppure il risultato di B invertito. Nell'adder poi, oltre ad A ed al valore calcolato di B, verrà introdotto il valore di C stesso, completando il complemento a 2 in caso di necessità, non apportando cambiamenti altrimenti.

2: Codice Mini ALU

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity mini_alu is
5    generic (bit_number : INTEGER := 4);
6      Port ( A,B : in STD_LOGIC_VECTOR (bit_number-1 downto 0);
7             C : in STD_LOGIC;
8             output : out STD_LOGIC_VECTOR (bit_number downto 0));
9  end mini_alu;
10
11 architecture Behavioral of mini_alu is
12   component generic_adder is
13     generic (bit_number:INTEGER := 4);
14       Port (
15         A_adder, B_adder : in STD_LOGIC_VECTOR (bit_number-1 downto 0);
16         cin : in STD_LOGIC;
17         sum : out STD_LOGIC_VECTOR (bit_number downto 0));
18   end component;
19
20 signal B_internal: STD_LOGIC_VECTOR (bit_number-1 downto 0);
21 signal carry_in: STD_LOGIC;
22
23 begin
24
25   process(A, B, C) begin
26     case C is
27       when '0' =>
28       B_internal <= B;
29
30       when others =>
31       B_internal <= STD_LOGIC_VECTOR(not B);
32
33     end case;
34   end process;
35
36 generic_adder_alu: generic_adder
37       GENERIC MAP (bit_number => bit_number)
38       PORT MAP (
39       A_adder => A,
40       B_adder => B_internal,
41       cin => C,
42       sum => output);
43 end Behavioral;
```

Possiamo trovare la schematica risultante nella *Figure 4*

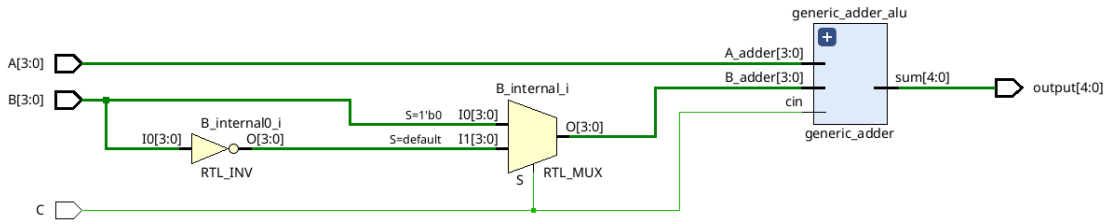

Figure 4: Circuito Logico del mini ALU

2.2 TestBench

I test sono stati svolti in tutti i casi possibili, dando un tempo di 10 ns per ogni caso, con un tempo totale in nanosecondi:

$$2^{2n+1} \times 10 \tag{1}$$

3: Codice test

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

entity mini_alu_testbench is
    generic (n: integer := 4);
end mini_alu_testbench;

architecture Behavioral of mini_alu_testbench is
    component mini_alu is
    --generic (n : INTEGER := 4);
    Port ( A,B : in STD_LOGIC_VECTOR (n-1 downto 0);
           C : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (n downto 0));
    end component;

    constant min_value : integer := -(2**(n-1));
    constant max_value : integer := (2**(n-1))-1;

    signal Ia,Ib: STD_LOGIC_VECTOR (n-1 downto 0);
    signal Ic: STD_LOGIC;
    signal Ooutput: STD_LOGIC_VECTOR(n downto 0);
begin
    CUT: mini_alu port map(Ia,Ib,Ic, Ooutput);
    process
    begin
      external: for i in min_value to max_value loop
          Ia <= (STD_LOGIC_VECTOR((TO_SIGNED(i,n))));
          internal: for j in min_value to max_value loop
            Ic <= '0';
            Ib <= (STD_LOGIC_VECTOR((TO_SIGNED(j,n))));
            wait for 10ns;
```

```
33              Ic <= '1';
34              wait for 10ns;
35            end loop internal;
36          end loop external;
37      end process;
38  end Behavioral;
```

2.3 Simulazione

Sono state effettuate simulazioni behavioural e post-implementation. Vediamole, evidenziandone le differenze.

2.4 Behavioural

Figure 5: Fine del primo input behavioural

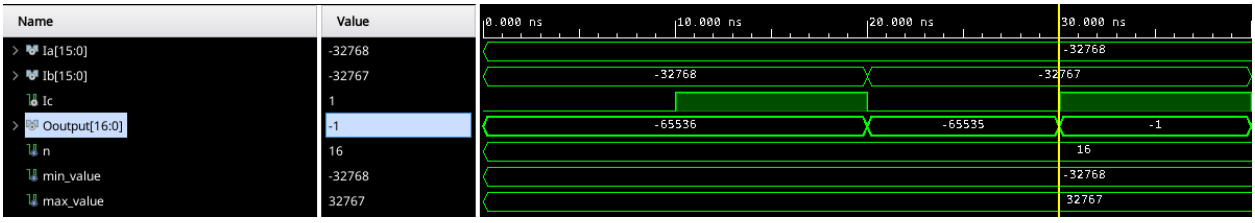

Figure 6: Inizio del secondo input behavioural