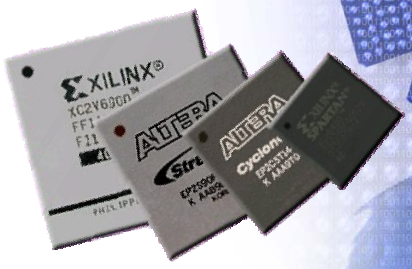
www.delucagiovanni.com

Corsi, Forum, Docs – Robotics and A.I.

Progetto di una CPU – SAP1
Simple as Computer step-by-step



Giovanni De Luca

Sito personale e forum

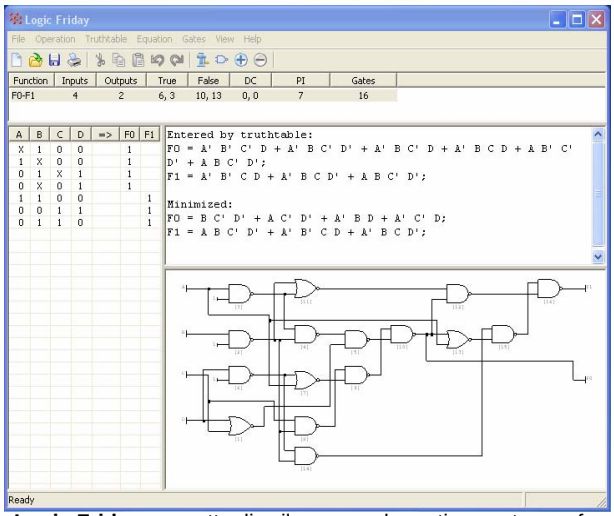
…….iniziamo con le installazioni

- Installazione dei seguenti software:

- **Logic Friday** (free)
- **Logic Works 5** (free)
- **Quartus II v9** (student version)

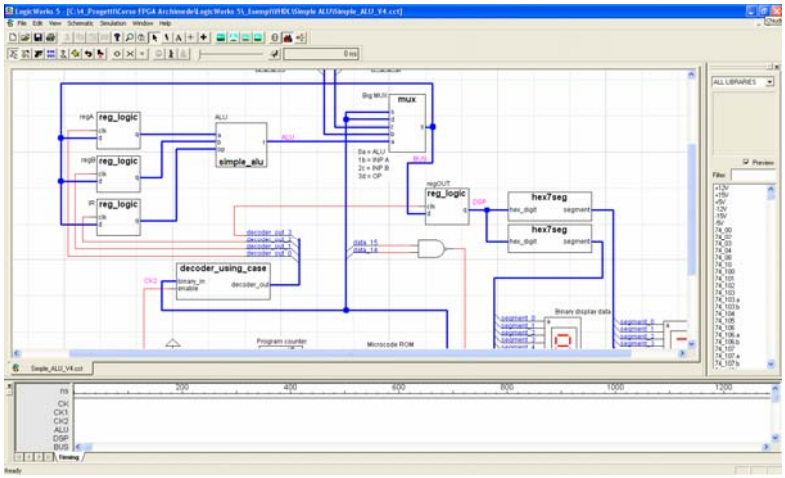
I software si trovano sulla cartella
“SW_Corso_FPGA”

Logic Friday



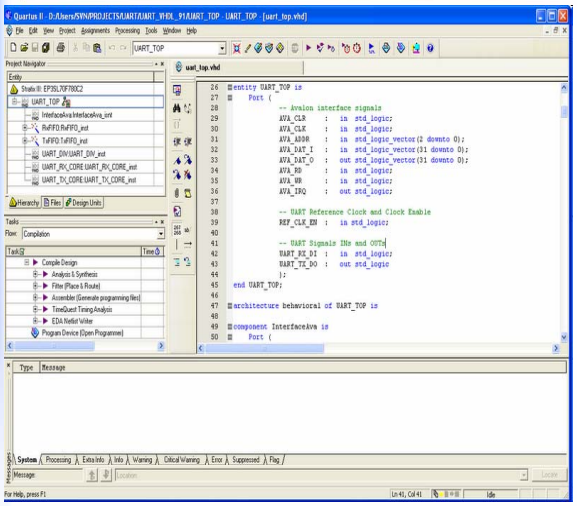
Logic Friday permette di sviluppare schematicamente una funzione logica combinatoria a partire dalla tabella della verità di n. variabili ingresso/uscita. La funzione può essere quindi minimizzata e convertita in schema Logico con la possibilità di scegliere il tipo di porte logiche da utilizzare nello schema finale.

LogicWorks



LogicWorks è un tool per lo sviluppo di circuiti elettronici interattivi a logica digitale.
Il pacchetto dà la potenza, la velocità e la flessibilità per creare e testare innumerevoli circuiti digitali direttamente sullo schermo del vostro computer.
Questo significa che si possono studiare concetti molto complessi in modo chiaro usando la simulazione senza spendere soldi, tempo e soprattutto senza danneggiare componenti.

Quartus II – Tool per FPGA



QUARTUS è un tool utilizzabile per effettuare, nell'ambito della progettazione di circuiti digitali:

- Sintesi logica
- Simulazione digitale
- Place and Route
- Analisi delle prestazioni

In questo corso verranno illustrati e commentati i vari passi che sarà necessario compiere nell'ambito del flusso di progetto di circuiti digitali tramite linguaggio VHDL.
QUARTUS è composto da diversi tools (Compiler, Simulator, Text Editor, etc.) ognuno dei quali serve per una fase specifica del flusso di progetto.

Finalità del corso

Apprendere le nozioni di base per la progettazione realizzazione di una “minimalized” CPU 8bit ed implementarla su una FPGA generica.

SAP-1

Simple as Computer 8bit -1

Schema a blocchi di una SAP

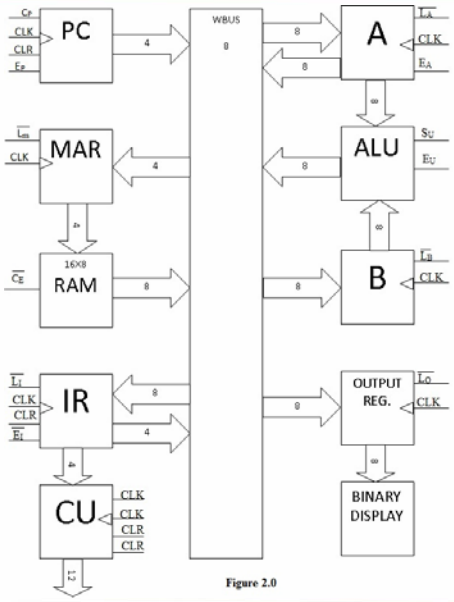


Figure 2.0

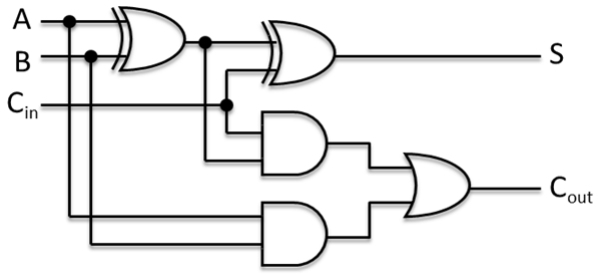
1. Program Counter (PC)
2. Memory Address Registers (MAR)
3. Random Access Memory (RAM)
4. Instruction Register (IR)
5. Controller Sequencer (CU)
6. Accumulator (A)
7. Adder-Subtracter (ALU)
8. B-Registers (B)
9. Output Register
10. Binary Display

Il Full Adder – il cuore di tutto !

Nell'architettura degli elaboratori i circuiti di tipo sommatore (e sottrattore) rivestono un ruolo di primissimo piano, dal momento che molte operazioni (come ad esempio la moltiplicazione) sono ricondotte all'esecuzione ripetuta di operazioni di addizione fra bit. Pertanto, visto che l'operazione di somma è richiesta con una frequenza elevata, è necessario che il circuito sommatore sia il più efficiente possibile, cioè veloce nell'eseguire la computazione, il più robusto possibile ai disturbi esterni e soprattutto non eccessivamente costoso.

Il più semplice circuito realizzabile per ottenere la somma fra due numeri ad bit si ottiene connettendo in cascata Full-Adder, in modo che l'uscita di un Full-Adder insista sull'ingresso del Full-Adder successivo (Ripple Carry).

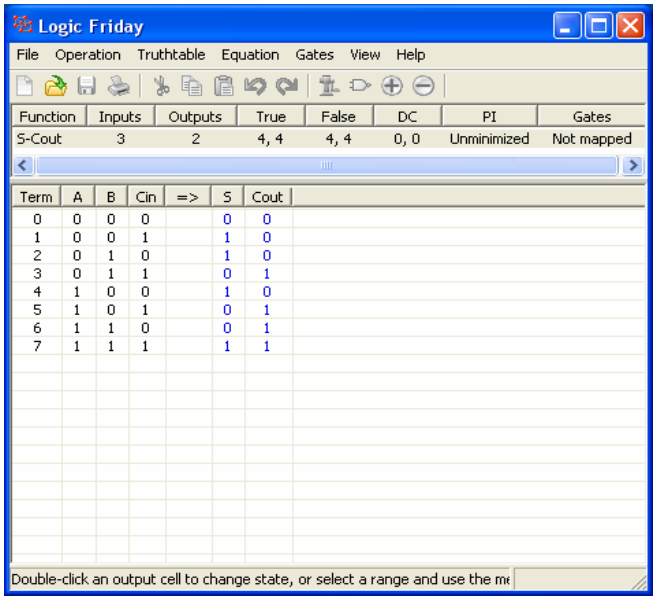
…….iniziamo



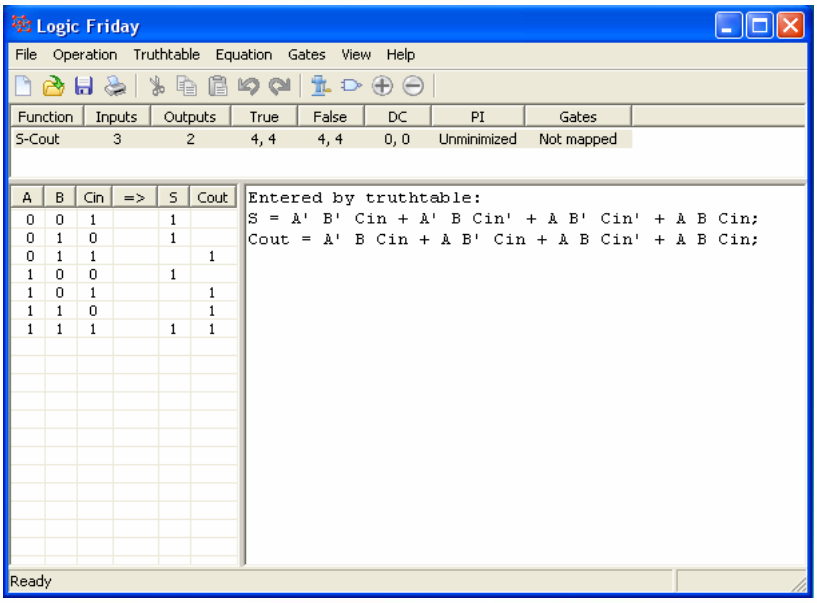
A_n	B_n	C_{n-1}	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabella della verità del full adder

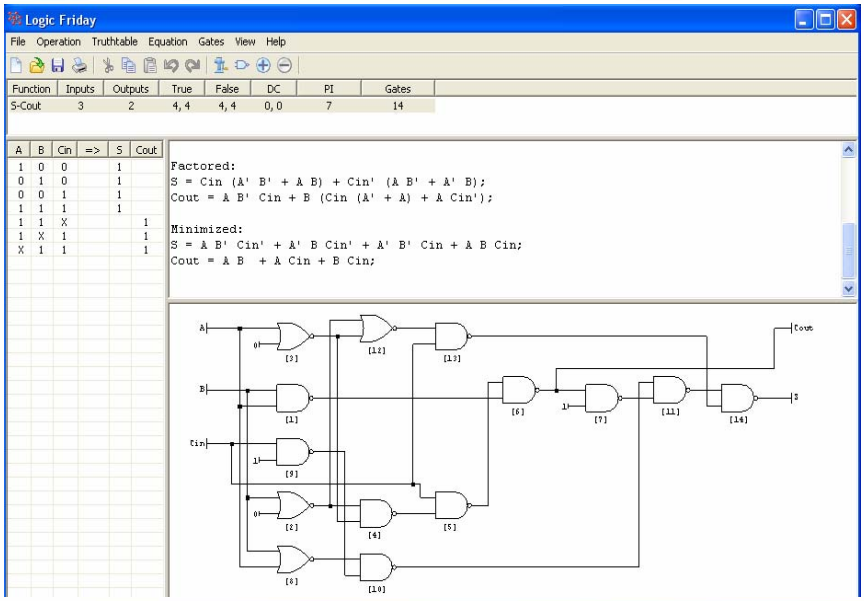
Logic Friday – tavola della verità



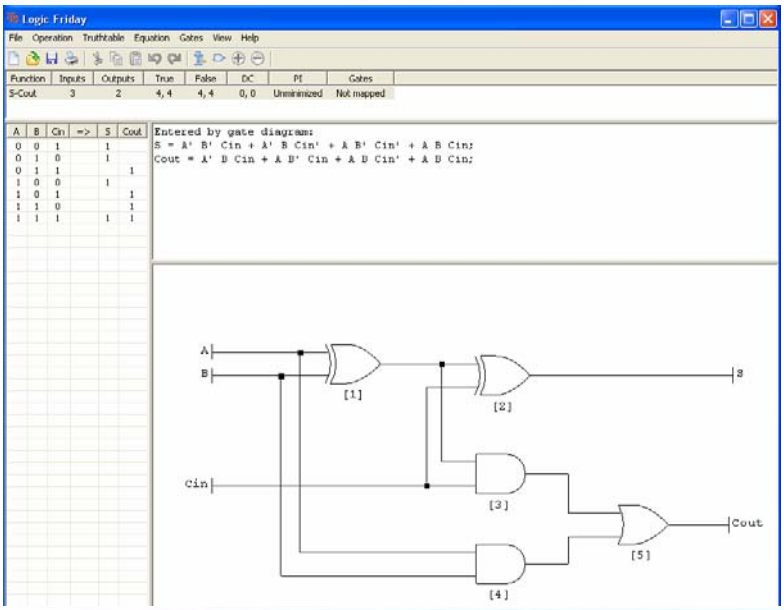
Logic Friday - minimizza



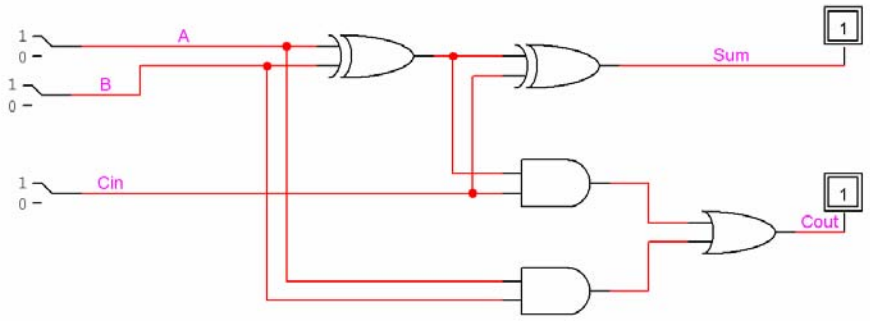
Schema elettrico – complesso !!



Dopo vari passaggi …….

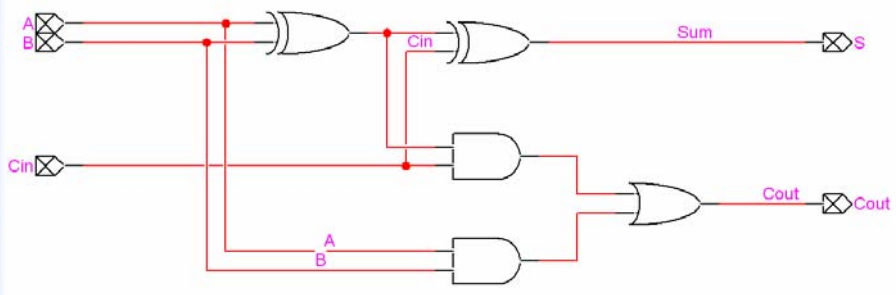


Usiamo LogicWorks



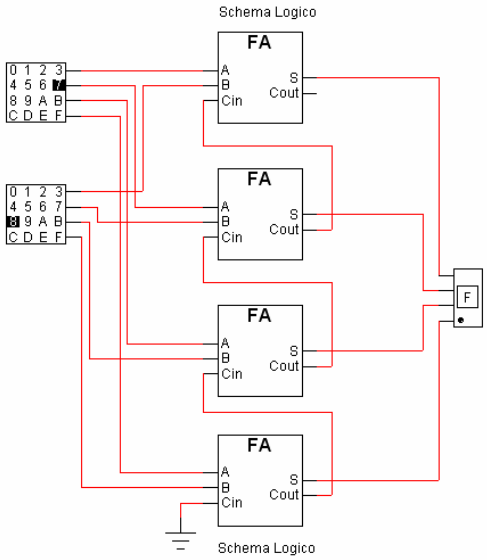
Circuito 1

Inseriamo i pin I/O per creare un IC



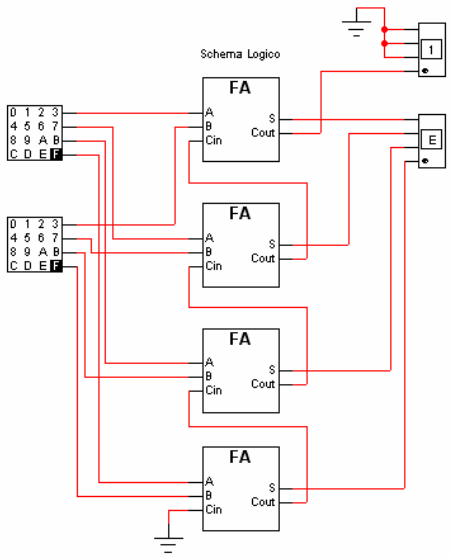
Circuito 2

Dopo aver creato il FA 4bit (cosa manca?)



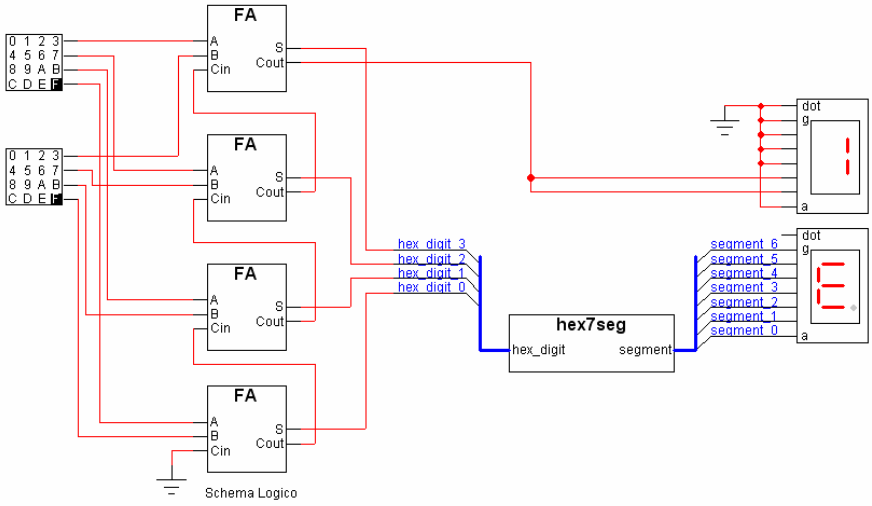
Circuito 3

Completo ……



Circuito 4

Aggiungiamo i display (hex7seg)

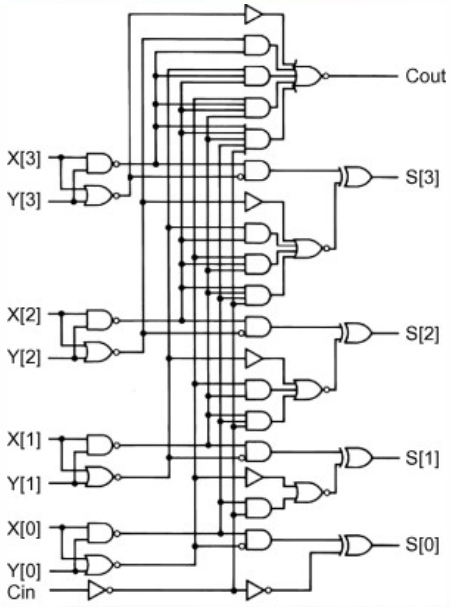


Circuito 5

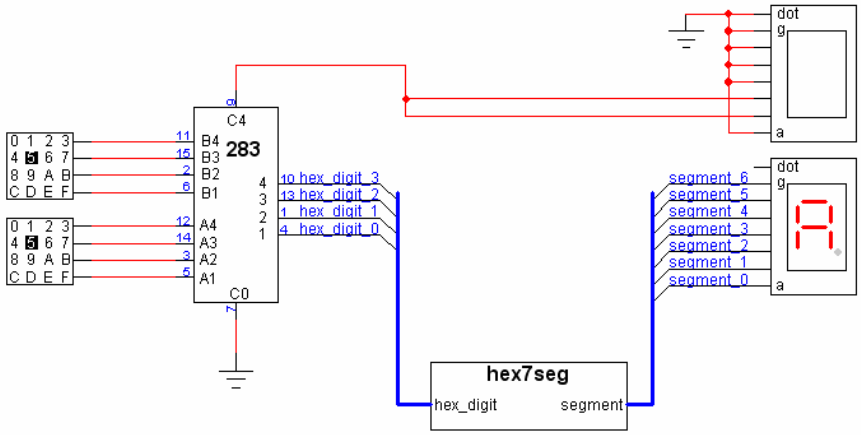
Hex7seg.dwv (VHDL code)

```
-- Hexadecimal to 7 Segment Decoder for LED Display

ENTITY hex7seg IS
    PORT(hex_digit : IN  BIT_VECTOR(3 DOWNTO 0);
         segment   : OUT BIT_VECTOR(0 TO 6));
END hex7seg;

ARCHITECTURE behavioral OF hex7seg IS

BEGIN
    WITH hex_digit SELECT
        -- HEX to 7 Segment Decoder for LED Display
        -- Hex-digit is the four bit binary value to display in hexadecimal
      segment <= "0000001" WHEN "0000",
                 "1001111" WHEN "0001",
                 "0010010" WHEN "0010",
                 "0000110" WHEN "0011",
                 "1001100" WHEN "0100",
                 "0100100" WHEN "0101",
                 "0100000" WHEN "0110",
                 "0001111" WHEN "0111",
                 "0000000" WHEN "1000",
                 "0000100" WHEN "1001",
                 "0001000" WHEN "1010",
                 "1100000" WHEN "1011",
                 "0110001" WHEN "1100",
                 "1000010" WHEN "1101",
                 "0110000" WHEN "1110",
                 "0111000" WHEN "1111";
    END behavioral;
```

Full Adder con porte logiche ?

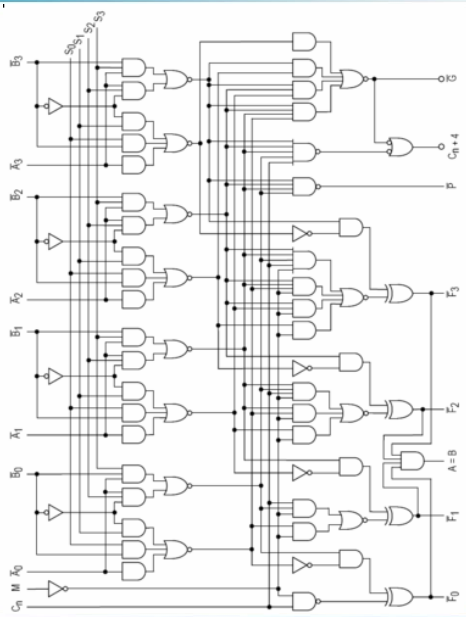


Full Adder con 74283

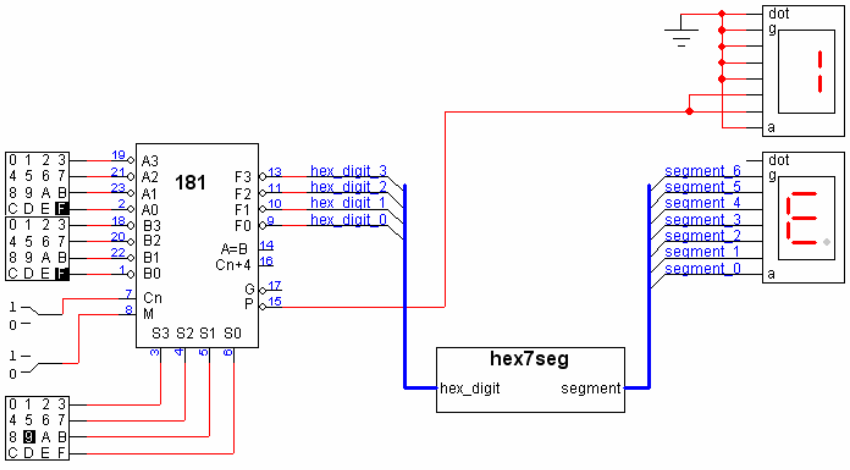


Circuito 6

4bit ALU con ……. ?



4bit ALU con 74181

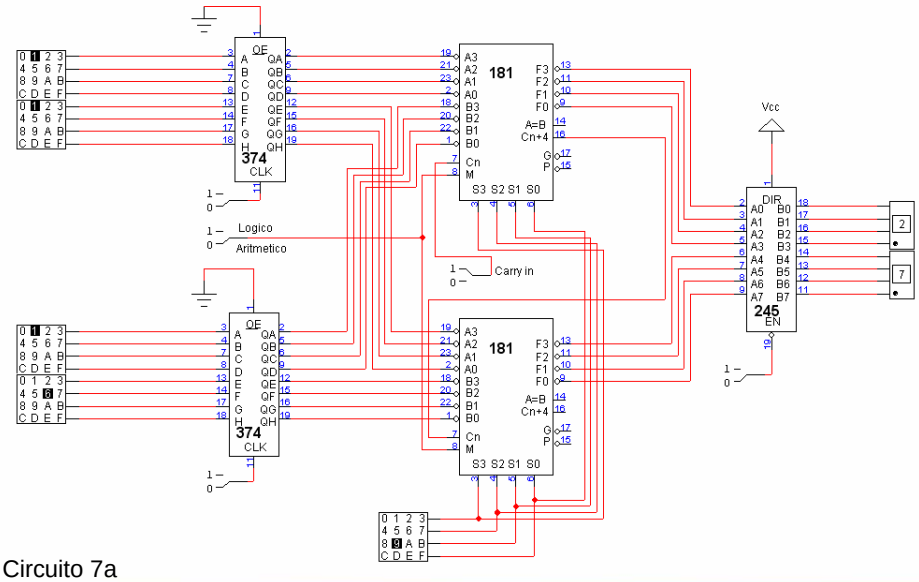


Circuito 7

Funzioni del 74181

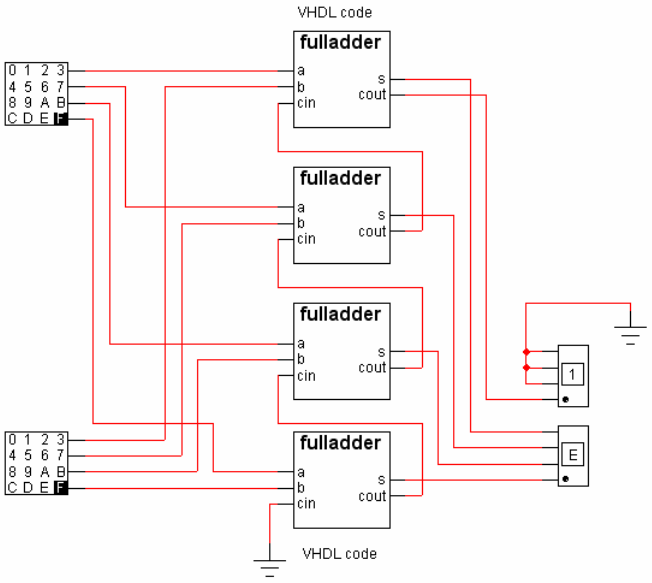
Codice della Funzione				Funzioni Logiche	Funzioni aritmetiche M = 0	
S3	S2	S1	S0	M = 1	Cn = 1	Cn = 0
0	0	0	0	$\overline{A}$	A	A + 1
0	0	0	1	$\overline{A+B}$	A + B	(A + B) + 1
0	0	1	0	$\overline{A}B$	A + $\overline{B}$	(A + $\overline{B}$) + 1
0	0	1	1	0	-1	0
0	1	0	0	$\overline{AB}$	A + A$\overline{B}$	A + A$\overline{B}$ + 1
0	1	0	1	$\overline{B}$	(A + B) + A$\overline{B}$	(A + B) + A$\overline{B}$ + 1
0	1	1	0	A $\oplus$ B	A - B - 1	A - B
0	1	1	1	A$\overline{B}$	A$\overline{B}$ - 1	A$\overline{B}$
1	0	0	0	$\overline{A}$ + B	A + AB	A + AB + 1
1	0	0	1	$\overline{A \oplus B}$	A + B	A + B + 1
1	0	1	0	B	(A + $\overline{B}$) + AB	(A + $\overline{B}$) + AB + 1
1	0	1	1	AB	AB - 1	AB
1	1	0	0	1	A + A	A + A + 1
1	1	0	1	A + $\overline{B}$	(A + B) + A	(A + B) + A + 1
1	1	1	0	A + B	(A + $\overline{B}$) + A	(A + $\overline{B}$) + A + 1
1	1	1	1	A	A - 1	A

Introduciamo i registri 8bit



Circuito 7a

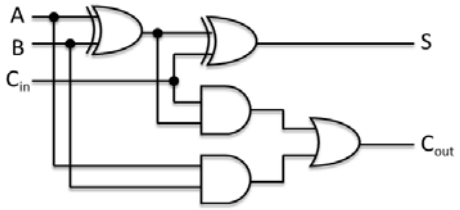
Full Adder 4bit in VHDL (data flow)



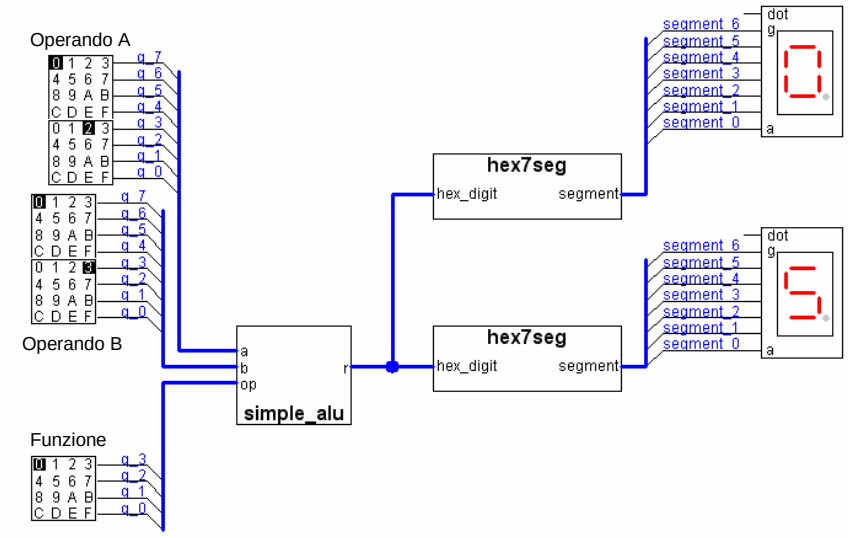
Circuito 5

Full Adder in VHDL (data flow)

```
library ieee;
use ieee.std_logic_1164.all;

entity fullAdder is
   port( A, B, Cin: in std_logic;
         S, Cout: out std_logic);
end fullAdder;

architecture faArch of fullAdder is
begin
    S <= A xor B xor Cin;
    Cout <= (A and B) or ((A xor B) and Cin);
end faArch;
```



Progettiamo una ALU 8bit



Circuito 8

ALU, Quali funzioni ?

```
-- 00 somma
-- 01 sottrazione
-- 02 incr
-- 03 decr

-- 04 NOT R1
-- 05 AND R1 R2
-- 06 OR  R1 R2
-- 07 XOR gate

-- 08 shift left logic
-- 09 shift right logic
-- 0A rotate left logic
-- 0B rotate right logic

-- 0C reg3 = reg1
-- 0D reg3 = reg2

-- other null zero
```

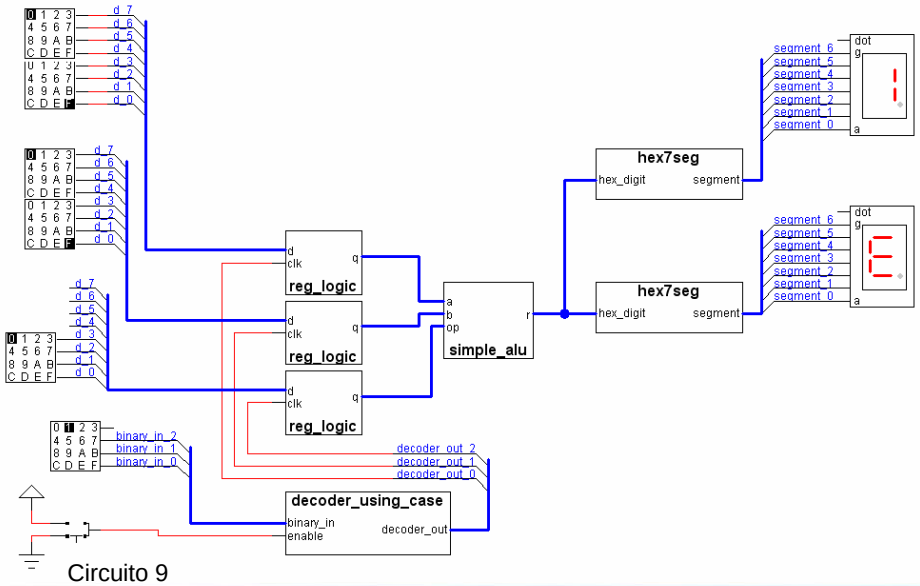
Descriviamo una ALU in VHDL (1)

```
-- simple_alu.dwv --
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity simple_alu is
port(A,B : in  signed  (7 downto 0);    -- input operands
      Op : in  unsigned(3 downto 0);    -- Operation to be performed
       R : out signed  (7 downto 0)     -- output of ALU
       );
end simple_alu;
-------------------------------------------------------
architecture Behavioral of simple_alu is

--temporary signal declaration.
  signal Reg1 : signed(7 downto 0) := (others => '0');
  signal Reg2 : signed(7 downto 0) := (others => '0');
  signal Reg3 : signed(7 downto 0) := (others => '0');

begin

    Reg1 <= A;
    Reg2 <= B;
    R <= Reg3;
```


ALU in VHDL (2)

```
 alu:process(op, Reg1, Reg2)
 begin
    case Op is
      when "0000" => Reg3 <= Reg1 + Reg2;  -- 00 somma
      when "0001" => Reg3 <= Reg1 - Reg2;  -- 01 sottrazione
      when "0010" => Reg3 <= Reg1 + 1;     -- 02 incr
      when "0011" => Reg3 <= Reg1 - 1;     -- 03 decr

      when "0100" => Reg3 <= not Reg1;     -- 04 NOT R1
      when "0101" => Reg3 <= Reg1 and Reg2;-- 05 AND R1 R2
      when "0110" => Reg3 <= Reg1 or Reg2; -- 06 OR  R1 R2
      when "0111" => Reg3 <= Reg1 xor Reg2;-- 07 XOR gate

      when "1000" => Reg3 <= Reg1 sll 1;   -- 08 shift left logic
      when "1001" => Reg3 <= Reg1 srl 1;   -- 09 shift right logic
      when "1010" => Reg3 <= Reg1 rol 1;   -- 0A rotate left logic
      when "1011" => Reg3 <= Reg1 ror 1;   -- 0B rotate right logic

      when "1100" => Reg3 <= Reg1;         -- 0C reg3 = reg1
      when "1101" => Reg3 <= Reg2;         -- 0D reg3 = reg2

      when others => Reg3 <="00000000";    -- null
     end case;
 end process;

 end Behavioral;
```

ALU con registri e display

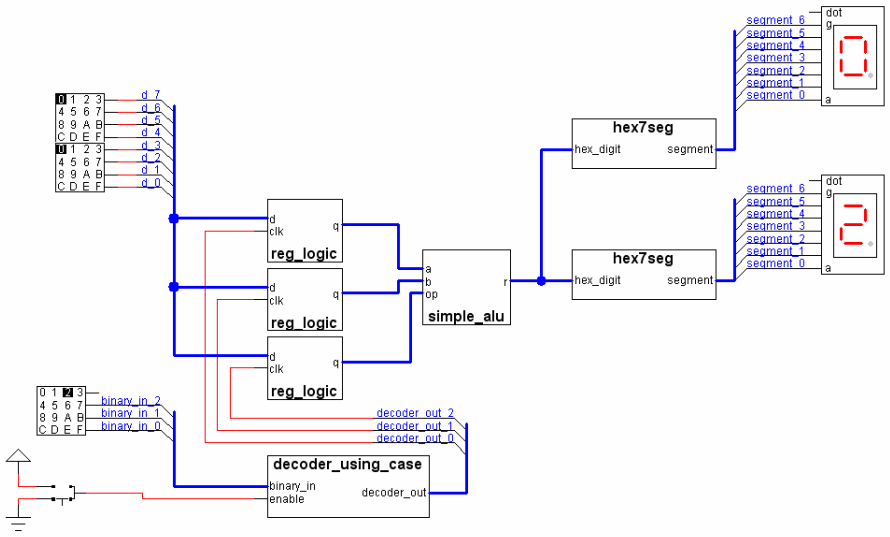


Circuito 9

Registri in VHDL

```
-- register8.dwv --

library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------
entity reg_logic is
port ( d   : in std_logic_vector(7 downto 0);
       clk : in std_logic;
       q   : out std_logic_vector(7 downto 0)
     );
end reg_logic;

architecture r_example of reg_logic is
begin
process (clk) begin
        if (clk'event and clk = '1') then
            q <= d;
        end if;
end process;
end r_example;
```

Aggiungiamo il Decoder

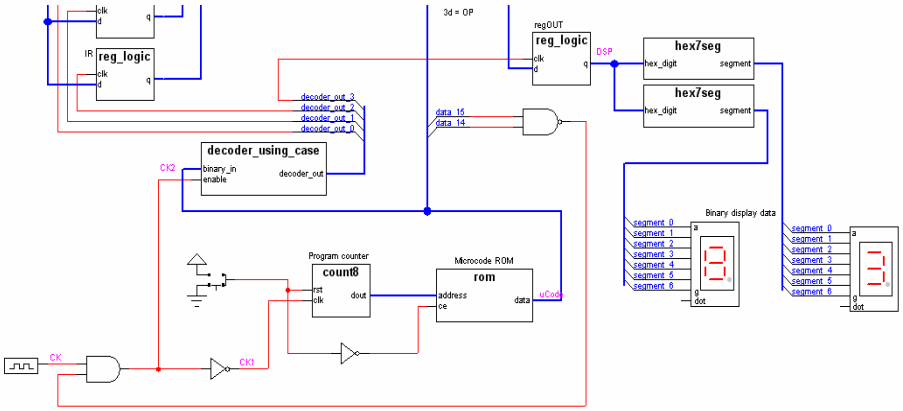


Circuito 10

Decoder in VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_using_case is
    port (
         enable      :in  std_logic;                     --   Enable for the decoder
         binary_in   :in  std_logic_vector (2 downto 0); --   4-bit Input
         decoder_out :out std_logic_vector (7 downto 0)  --  16-bit Output
         );
end entity;
-------------------------------------------------------
architecture bh of decoder_using_case is
begin
-------------------------------------------------------
process (enable, binary_in)
begin
   decoder_out <= X"00";
        if (enable = '1') then
            case (binary_in) is
                when "000"   => decoder_out <= X"01";
                when "001"   => decoder_out <= X"02";
                when "010"   => decoder_out <= X"04";
                when "011"   => decoder_out <= X"08";
                when "100"   => decoder_out <= X"10";
                when "101"   => decoder_out <= X"20";
                when "110"   => decoder_out <= X"40";
                when "111"   => decoder_out <= X"80";
                when others => decoder_out <= X"00";
            end case;
        end if;
    end process;
end bh;
```

Program Counter e Micro-codice



Circuito 11

PC - in VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
-----------------------------------------------------------------
entity count8 is
 port( CLK  : in  std_logic;
       RST  : in  std_logic;
       DOUT : out std_logic_vector(7 downto 0)
     );
end count8;
-----------------------------------------------------------------
architecture arch1 of count8 is
begin
clk_proc : process(CLK,RST)

    variable count : unsigned(7 downto 0) := unsigned("00000000");

begin

    if (rst = '1') then
        count := "00000000";
    elsif clk'event AND CLK = '1' then
        count := count + 1;
    end if;

    DOUT <= std_logic_vector(count);
end process clk_proc;
end arch1;
```

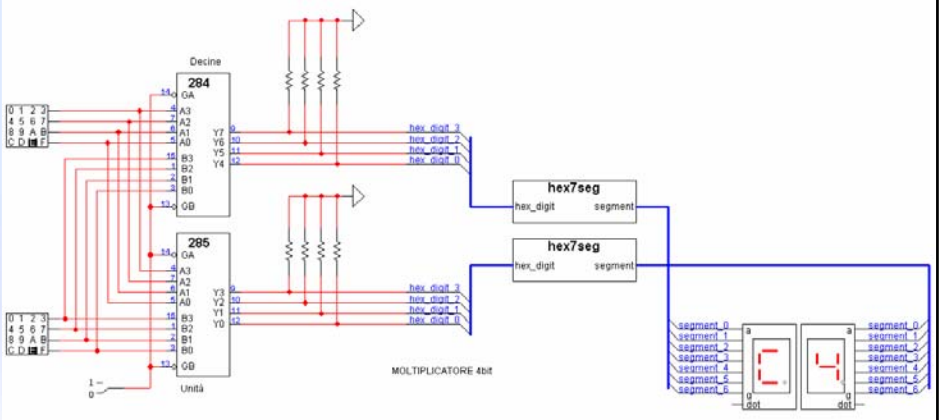
Micro-codice (ROM) in VHDL (1)

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_arith.all;
 -- use ieee.std_logic_unsigned.all;

entity rom is
    port ( ce      :in  std_logic;                       -- Chip Enable
           address :in  std_logic_vector (3 downto 0); -- Address input
           data    :out std_logic_vector (15 downto 0) -- Data output
         );
end entity;
architecture behavior of rom is
```


Microcodice (ROM) in VHDL (2)

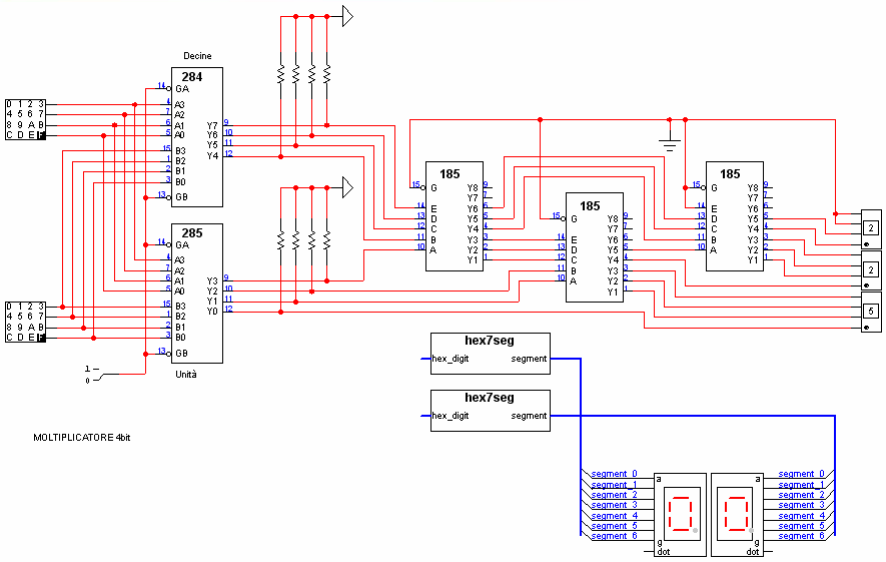
```
begin
process (ce, address) begin
if (ce = '1') then
    case (address) is

    ------------------------------------------------------------------------------------------------
    ------------------        SelMx    ------------------------------------------------------------
    ------------------        21010    ------------------------------------------------------------
    when x"0"   => data <= "00000011" & x"00";      -- NOP           ; 0300 NOP load zero dalla rom
    when x"1"   => data <= "00001111" & x"00";      -- MOV out,0     ; 0F00 NOP
    when x"2"   => data <= "00000011" & x"00";      -- LDI regA,0    ; 0300 Load immediate zero su regA
    when x"3"   => data <= "00000111" & x"00";      -- LDI regB,0    ; 0700 Load immediate zero su regB
    when x"4"   => data <= "00001011" & x"00";      -- ADD regA,regB ; 0B00 Somma regA con regB
    when x"5"   => data <= "00000001" & x"00";      -- LDR regA,inpA ; 0100 Load regA con inpA
    when x"6"   => data <= "00000110" & x"00";      -- LDR regB,inpB ; 0600 load regB con inpB
    when x"7"   => data <= "00001100" & x"00";      -- MOV out,Acc   ; 0C00 Muovi ACC su display
    when x"8"   => data <= "00000000" & x"00";      -- LDR regA,Acc  ; 0000 Load regA con ACC
    when x"9"   => data <= "00001011" & x"02";      -- INC regA      ; 0B02 Incrementa regA
    when x"A"   => data <= "00000000" & x"00";      -- LDR regA,Acc  ; 0000 Load regA con ACC
    when x"B"   => data <= "00001011" & x"02";      -- INC regA      ; 0B02 Incrementa regA
    when x"C"   => data <= "00000000" & x"00";      -- LDI regA,Acc  ; 0000 Load regA con ACC
    when x"D"   => data <= "00001011" & x"0B";      -- ROR regA      ; 0B0B Ruota a destra ACC
    when x"E"   => data <= "00001100" & x"00";      -- MOV out,Acc   ; 0C00 Muovi ACC su display
    when x"F"   => data <= "11000000" & x"00";      -- END           , C000 Fine del programma
    when others => data <= x"C000";

  end case;
else
    data<=x"C000";
end if;
end process;
end architecture;
```

Moltiplicatore Hardware 4bit



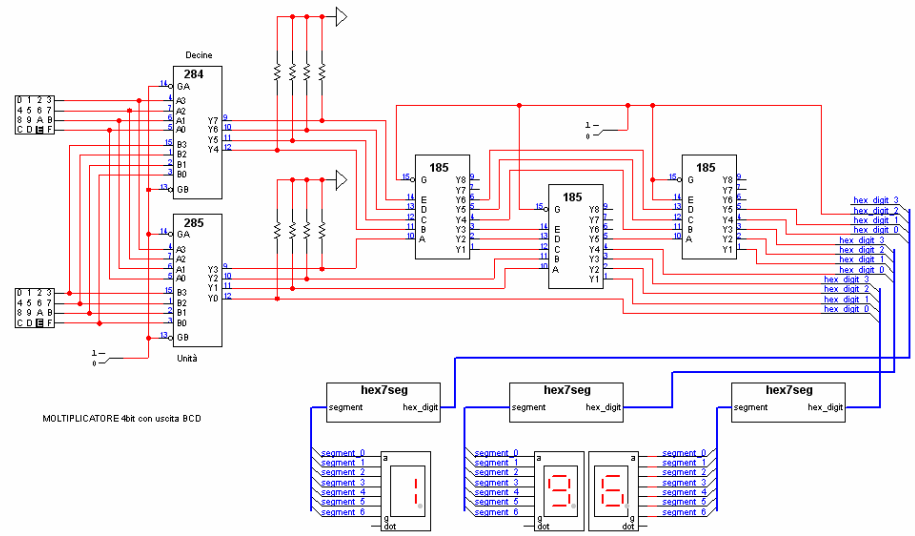
Circuito 12

Moltiplicatore con uscita BCD



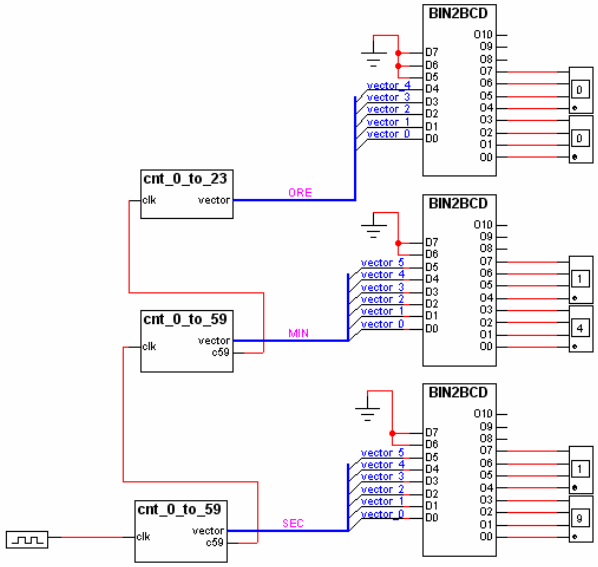
Circuito 13

Moltiplicatore completo BCD



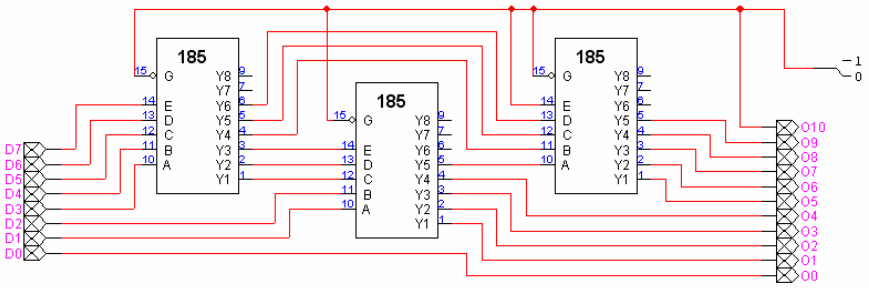
Circuito 14

Orologio BCD



Orologio

Convertitore Binario - BCD



Contatore 0..23

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity cnt_0_to_23 is
    port( clk:in std_logic;
          vector:out std_logic_vector(4 downto 0));
end cnt_0_to_23;

architecture cnt_behavior of cnt_0_to_23 is
begin
process(clk)
    variable cnt : unsigned(4 downto 0) := unsigned("00000");
begin
    if(clk'event and clk = '1') then
        if(cnt = "10111")then
            cnt := "00000";
        else
            cnt := cnt + 1;
        end if;
    end if;
    vector <= STD_LOGIC_VECTOR(cnt);
end process;
end cnt_behavior;
```

Contatore 0..59

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity cnt_0_to_59 is
    port( clk:in std_logic;
          c59:out std_logic;
          vector:out std_logic_vector(5 downto 0));
end cnt_0_to_59;

architecture cnt_behavior of cnt_0_to_59 is
begin
process(clk)
    variable cnt : unsigned(5 downto 0) := unsigned("000000");
begin
    if(clk'event and clk = '1') then
        if(cnt = "111011")then
            cnt := "000000";
            c59<='1';
        else
            cnt := cnt + 1;
            c59<='0';
        end if;
    end if;
    vector <= STD_LOGIC_VECTOR(cnt);
end process;
end cnt_behavior;
```


Altre entità ed architetture

Counter up/down

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity up_down_counter is
  port (
    cout    :out std_logic_vector (7 downto 0);-- Output of the counter
    up_down :in  std_logic;                    -- up_down control for counter
    clk     :in  std_logic;                    -- Input clock
    reset   :in  std_logic                     -- Input reset
  );
end entity;

architecture rtl of up_down_counter is
    signal count :std_logic_vector (7 downto 0);
begin
    process (clk, reset) begin
        if (reset = '1') then
            count <= (others=>'0');
        else
            wait until (clk'event and clk = '1');
            if (up_down = '1') then
                count <= count + 1;
            else
                count <= count - 1;
            end if;
        end if;
    end process;
    cout <= count;
end architecture;
```


Encoder priority

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

entity ENC7 is
    port(DIN : in std_logic_vector(6 downto 0);
         DOUT : out std_logic_vector(2 downto 0));
end ENC7;

architecture Arch1 of ENC7 is
begin
process (DIN)
begin
    DOUT <= "000";
        if DIN(0)='1' then DOUT<="001"; end if;
        if DIN(1)='1' then DOUT<="010"; end if;
        if DIN(2)='1' then DOUT<="011"; end if;
        if DIN(3)='1' then DOUT<="100"; end if;
        if DIN(4)='1' then DOUT<="101"; end if;
        if DIN(5)='1' then DOUT<="110"; end if;
        if DIN(6)='1' then DOUT<="111"; end if;
end process;
end Arch1;
```

Finite state machine (1)

```
library ieee ;
use ieee.std_logic_1164.all;
---------------------------------------------------
entity seq_design is
    port(a:     in std_logic;
         clock: in std_logic;
         reset: in std_logic;
         x:     out std_logic_vector(3 downto 0)
         );
end seq_design;
---------------------------------------------------
architecture FSM of seq_design is

    -- define the states of FSM model
    type state_type is (S0, S1, S2, S3);
    signal next_state, current_state: state_type;

begin

-- cocurrent process#1: state registers --
state_reg: process(clock, reset)
begin

    if (reset='1') then
            current_state <= S0;
    elsif (clock'event and clock='1') then
            current_state <= next_state;
    end if;

end process;
```


FSM (2)

```
-- cocurrent process#2: combinational logic --
comb_logic: process(current_state, a)
begin
    -- use case statement to show the state transistion --
    case current_state is
        when S0 =>  x <= "0000";
                if a='0' then
                    next_state <= S0;
                elsif a ='1' then
                    next_state <= S1;
                end if;

        when S1 =>  x <= "0001";
                if a='0' then
                    next_state <= S1;
                elsif a='1' then
                    next_state <= S2;
                end if;

        when S2 =>  x <= "0010";
                if a='0' then
                    next_state <= S2;
                elsif a='1' then
                    next_state <= S3;
                end if;

        when S3 =>  x <= "0011";
                if a='0' then
                    next_state <= S3;
                elsif a='1' then
                    next_state <= S0;
                end if;

        when others =>
                x <= "0000";
                next_state <= S0;

    end case;
end process;
end FSM;
```