UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

SISTEME CU CIRCUITE INTEGRATE DIGITALE

PROIECT: GENERAREA NUMERELOR FIBONACCI

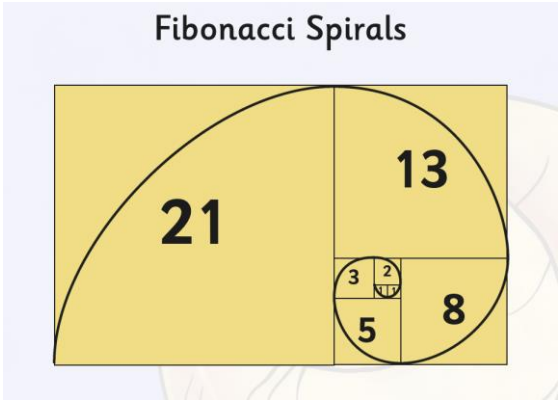
Profesor indrumator: Lupu Miruna

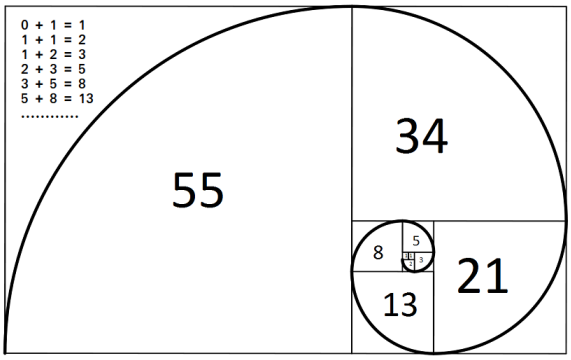
Paul Farago Egri Anna-Maria

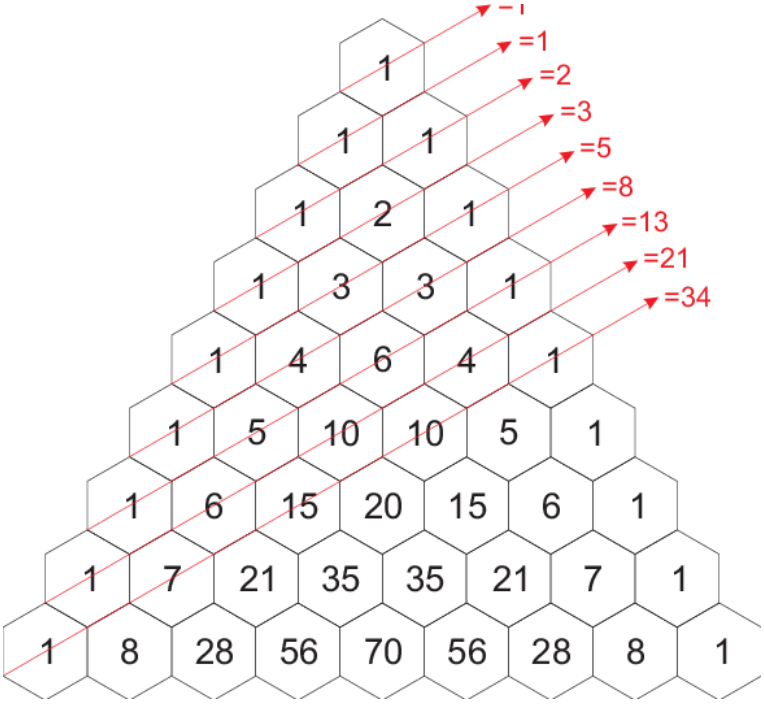
- Ce sunt numerele Fibonacci?

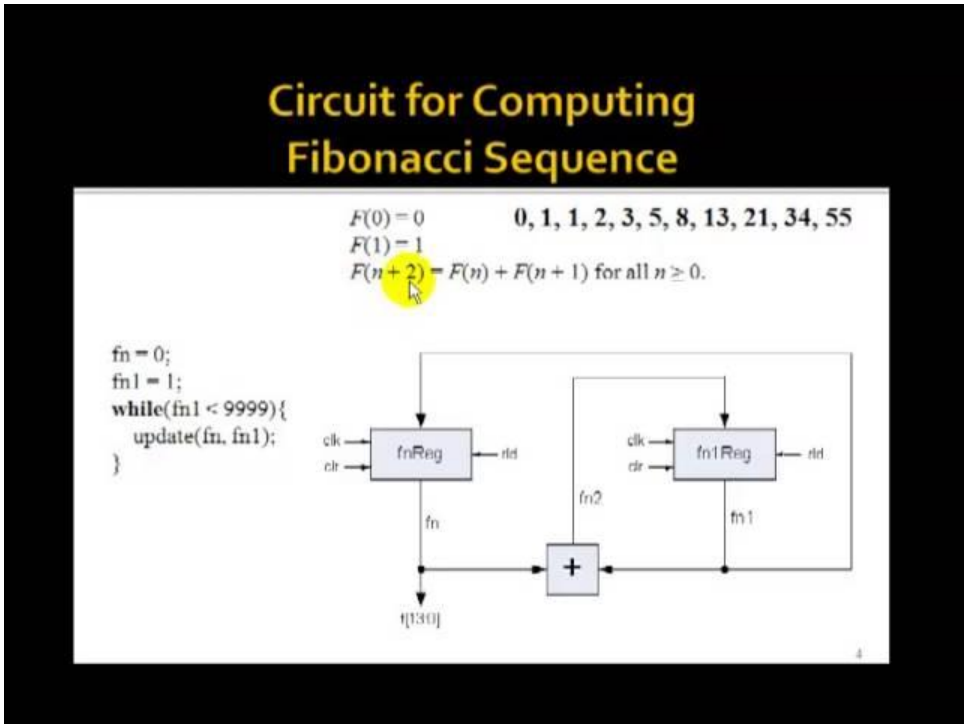
Numerele Fibonacci sunt o serie de numere în care fiecare număr este suma celor două numere anterioare. Începând cu 0 și 1, primele câteva numere din această serie sunt: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 și așa mai departe. Această serie de numere a fost introdusă de matematicianul italian Leonardo Fibonacci în secolul al XIII-lea.

- Fibonacci Spirals:









- Cerinta proiectului:

20. Generarea numerelor Fibonacci

Sa se proiecteze un circuit pentru a genera numerele Fibonacci. Numărul curent sa fie afișat pe 7 segmente. Daca șirul depășește 4 digiti atunci se reia șirul de la 0. Incrementarea numerelor sa se facă cu perioada de 1 sec.

Observație: Șirul Fibonacci se calculează folosind formula $F_0=0$, $F_1=1$, $F_n=F_{n-1} + F_{n-2}$.

- Proiectul:

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

entity Fibo is                                  --entitate
Port ( clk : in STD_LOGIC;
rst : in STD_LOGIC;
fibo_out : out STD_LOGIC_VECTOR (13 downto 0);
thousand : out integer range 0 to 9;
hundred : out integer range 0 to 9;
ten : out integer range 0 to 9;
unit : out integer range 0 to 9);
end Fibo;

architecture Behavioral of Fibo is

-- declarare semnale

signal fib1 , fib2 : STD_LOGIC_VECTOR (13 downto 0);
signal unitati ,zeci ,sute ,mii : integer range 0 to 9;
constant fibo_max: integer := 9999;



begin

p1: process (clk,rst)
begin

if rst = '1' then
```

```vhdl
if rst = '1' then
fib1 <= "00000000000000";
fib2 <= "00000000000001";
mii <= 0;
sute <= 0;
zeci <= 0;
unitati <= 0;

elsif rising_edge(clk) then
  if fib2 <fibo_max then
  fib2 <= fib1 + fib2;
  fib1 <= fib2;

  --transformare zecimal
  unitati <= to_integer(unsigned(fib2))mod 10;
  zeci <=( to_integer(unsigned(fib2))mod 100)/10;
```

```
    sute <= (to_integer(unsigned(fib2))mod 1000 )/ 100;
    mii <= (to_integer(unsigned(fib2))mod 10000)/ 1000;

    else
    fib1 <= "00000000000000";
    fib2 <= "00000000000001";
    mii <= 0;
    sute <= 0;
    zeci <= 0;
    unitati <= 0;

  end if;
  end if;
  end process;

  --asociere semnale

 --asociere semnale

 fibo_out <= fib1;
 unit <= unitati;
 ten <= zeci;
 hundred  <= sute;
 thousand <= mii;

 end Behavioral;
```

Acest cod VHDL descrie un circuit digital care generează numerele Fibonacci și afișează ultimul număr generat pe un afișaj cu 7 segmente.

Circuitul are, de asemenea, un modul de contorizare care resetează numerele la 0 atunci când numărul de cifre afișate depășește 4.

Procesul p1 este cel care generează numerele Fibonacci. Atunci când semnalul de reset (rst) este activ (valoare logică '1'), procesul resetează toate semnalele la valorile inițiale.

Dacă semnalul de reset nu este activ, atunci procesul verifică dacă ultimul număr Fibonacci generat (fib2) este mai mic decât valoarea maximă (9999).

Dacă da, procesul generează următorul număr Fibonacci și convertește valoarea acestuia în sistemul zecimal. În caz contrar, toate semnalele sunt resetate la valorile inițiale.

Pentru a afișa numărul Fibonacci pe afișajul cu 7 segmente, procesul convertește valoarea numărului în patru semnale separate pentru fiecare cifră (mii, sute, zeci și unități).

Aceste semnale de ieșire (thousand, hundred, ten și unit) sunt apoi utilizate pentru a controla afișajul cu 7 segmente pentru a afișa numărul Fibonacci curent.

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

entity bcd is
    generic(N: positive := 16);
    port(clk, reset: in std_logic;
        binary_in: in std_logic_vector(N-1 downto 0);
        bcd0, bcd1, bcd2, bcd3: out std_logic_vector(3 downto 0));
end bcd ;

architecture behaviour of bcd is
    type states is (start, shift, done);
    signal state, state_next: states;

    signal binary, binary_next: std_logic_vector(N-1 downto 0);
    signal bcds, bcds_reg, bcds_next: std_logic_vector(19 downto 0);
    -- output register keep output constant during conversion
    signal bcds_out_reg, bcds_out_reg_next: std_logic_vector(19 downto 0);
    -- need to keep track of shifts
    signal shift_counter, shift_counter_next: natural range 0 to N;
begin

    process(clk, reset)
    begin
        if reset = '1' then
            binary <= (others => '0');
            bcds <= (others => '0');
            state <= start;
            bcds_out_reg <= (others => '0');
            shift_counter <= 0;
        elsif falling_edge(clk) then
            binary <= binary_next;
            bcds <= bcds_next;
            state <= state_next;
            bcds_out_reg <= bcds_out_reg_next;
            shift_counter <= shift_counter_next;
        end if;
    end process;

    convert:
    process(state, binary, binary_in, bcds, bcds_reg, shift_counter)
    begin
        state_next <= state;
        bcds_next <= bcds;
        binary_next <= binary;
        shift_counter_next <= shift_counter;

        case state is
            when start =>
                state_next <= shift;
                binary_next <= binary_in;
                bcds_next <= (others => '0');
                shift_counter_next <= 0;
            when shift =>
                if shift_counter = N then
                    state_next <= done;
                else
                    binary_next <= binary(N-2 downto 0) & 'L';
                    bcds_next <= bcds_reg(18 downto 0) & binary(N-1);
```

```
                shift_counter_next <= shift_counter + 1;
            end if;
        when done =>
            state_next <= start;
    end case;
end process;

bcds_reg(19 downto 16) <= bcds(19 downto 16) + 3 when bcds(19 downto 16) > 4 else
                          bcds(19 downto 16);
bcds_reg(15 downto 12) <= bcds(15 downto 12) + 3 when bcds(15 downto 12) > 4 else
                          bcds(15 downto 12);
bcds_reg(11 downto 8) <= bcds(11 downto 8) + 3 when bcds(11 downto 8) > 4 else
                         bcds(11 downto 8);
bcds_reg(7 downto 4) <= bcds(7 downto 4) + 3 when bcds(7 downto 4) > 4 else
                        bcds(7 downto 4);
bcds_reg(3 downto 0) <= bcds(3 downto 0) + 3 when bcds(3 downto 0) > 4 else
                        bcds(3 downto 0);

bcds_out_reg_next <= bcds when state = done else
                     bcds_out_reg;

bcd3 <= bcds_out_reg(15 downto 12);
bcd2 <= bcds_out_reg(11 downto 8);
bcd1 <= bcds_out_reg(7 downto 4);
bcd0 <= bcds_out_reg(3 downto 0);

end behaviour;
```

Acest cod descrie un convertor binar-cod BCD (Binary Coded Decimal) într-un stil de programare comportamental. Convertorul primește intrarea sub forma unui semnal binar de lungime N și generează ieșirile codului BCD, reprezentate de semnalele bcd0, bcd1, bcd2 și bcd3.

Procesul principal este împărțit în două părți, un proces de conversie și un proces de ieșire.

Procesul de conversie începe cu o stare de start și trece apoi în starea de shift, în care se realizează conversia efectivă a semnalului binar în codul BCD. În această stare, semnalul binar este deplasat un bit la stânga, iar un bit nou este adăugat pe partea dreaptă, astfel încât să se poată determina valoarea corespunzătoare codului BCD. Acest proces se repetă de N ori, unde N este numărul de biți al semnalului binar.

După ce conversia este finalizată, procesul trece în starea de done și revine la starea de start. În timpul acestei stări, semnalul de ieșire este menținut constant, astfel încât să se poată citi valorile corespunzătoare.

Procesul de ieșire utilizează un registru pentru a menține semnalul de ieșire constant în timpul conversiei. Valorile acestui registru sunt actualizate numai în timpul stării de done. Ulterior, valorile corespunzătoare sunt atribuite semnalelor de ieșire bcd0, bcd1, bcd2 și bcd3.

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



entity fib_on_board is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           seg : out STD_LOGIC_VECTOR (0 to 6);
           dp : out STD_LOGIC;
           an : out STD_LOGIC_VECTOR (3 downto 0));
end fib_on_board;

architecture Behavioral of fib_on_board is

component Fibo is
Port (  clk : in STD_LOGIC;
        rst : in STD_LOGIC;
        fibo_out : out STD_LOGIC_VECTOR (13 downto 0);
        thousand : out integer range 0 to 9;
        hundred : out integer range 0 to 9;
        ten : out integer range 0 to 9;
        unit : out integer range 0 to 9);
end component Fibo;

signal fibo_out : STD_LOGIC_VECTOR (13 downto 0);
signal clk1Hz: STD_LOGIC;

component bcd is
    generic(N: positive := 16);
      port( clk, reset: in std_logic;
            binary_in: in std_logic_vector(N-1 downto 0);
            bcd0, bcd1, bcd2, bcd3: out std_logic_vector(3 downto 0));
end component bcd ;

signal binary_in: STD_LOGIC_VECTOR (15 downto 0);
signal reset: STD_LOGIC;
signal bcd0, bcd1, bcd2, bcd3: STD_LOGIC_VECTOR (3 downto 0);

begin
```

```
  -- divizor 1 Hz
) process(clk, rst)
  variable q : integer := 0;
  begin
)   if rst = '1' then
      q := 0;
      clk1Hz <= '0';
    elsif rising_edge(clk) then
)       if q = 10**8 - 1 then
          q := 0;
          clk1Hz <= '1';
        else
          q := q + 1;
          clk1Hz <= '0';
)     end if;
)   end if;
) end process;

) u_fib : Fibo port map ( clk => clk1Hz,
                          rst => rst,
                         fibo_out => fibo_out);

binary_in <= "00" & fibo_out;

u_bcd : bcd port map (clk => clk,
                      reset=> reset,
                      binary_in => binary_in,
                      bcd0 => bcd0,
                      bcd1 => bcd1,
                      bcd2 => bcd2,
                      bcd3 => bcd3);

end Behavioral;
```

Acesta este un cod VHDL care implementează un modul de afișare a numărului Fibonacci pe un display cu 4 cifre BCD (binary coded decimal). În esență, modulul primește semnalul de ceas și semnalul de reset, generează un semnal de ceas de 1 Hz folosind un divizor, calculează numerele Fibonacci și le transformă în format BCD pentru a fi afișate pe display.

Componentele utilizate în acest modul sunt "Fibo", care primește semnalul de ceas și semnalul de reset și generează numerele Fibonacci, și "bcd", care primește semnalul de ceas și semnalul de reset, precum și numerele generate de "Fibo", și le convertește în format BCD.

Semnalul "fibo_out" este semnalul de ieșire care stochează numerele Fibonacci generate de componenta "Fibo". Semnalul "clk1Hz" este semnalul de ieșire generat de divizorul de ceas și care este folosit pentru a alimenta componenta "Fibo".

Semnalul "binary_in" este semnalul intermediar care stochează numerele Fibonacci convertite în format binar, iar semnalele "bcd0", "bcd1", "bcd2" și "bcd3" reprezintă semnalele de ieșire pentru cele patru cifre ale display-ului BCD.

- SIMULAREA:

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

use IEEE.NUMERIC_STD.ALL;

entity fib_test is
--  Port ( );
end fib_test;

architecture Behavioral of fib_test is

component  Fibo is
Port ( clk : in STD_LOGIC;
rst : in STD_LOGIC;
fibo_out : out STD_LOGIC_VECTOR (13 downto 0);
thousand : out integer range 0 to 9;
hundred  : out integer range 0 to 9;
ten : out integer range 0 to 9;
unit : out integer range 0 to 9);

end component ;

signal clk, rst : std_logic;
signal fibo_out :std_logic_vector (13 downto 0);
signal thousand , hundred , ten ,unit : integer range 0 to 9;

begin

uut : Fibo port map (clk => clk,
                     rst =>rst,
                     fibo_out=>fibo_out ,
                     thousand =>thousand ,
                     hundred =>hundred  ,
                     ten=>ten,
                     unit=>unit);

rst <= '1' after 0 ns, '0' after 1 ns;

process
begin
clk <= '0'; wait for 5 ns;
clk <= '1'; wait for 5 ns;


end process;
end Behavioral;
```

- CONSTRAINTS:

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

##7 Segment Display
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {seg[0]}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {seg[1]}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {seg[2]}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {seg[3]}]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {seg[4]}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {seg[5]}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {seg[6]}]

set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports dp]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {an[0]}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {an[1]}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {an[2]}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {an[3]}]

##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports rst]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports btnU]
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports btnL]
#set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports btnR]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports btnD]

## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

- REZULTATUL SIMULARII: