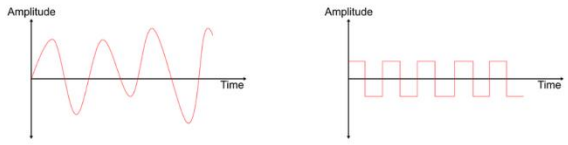
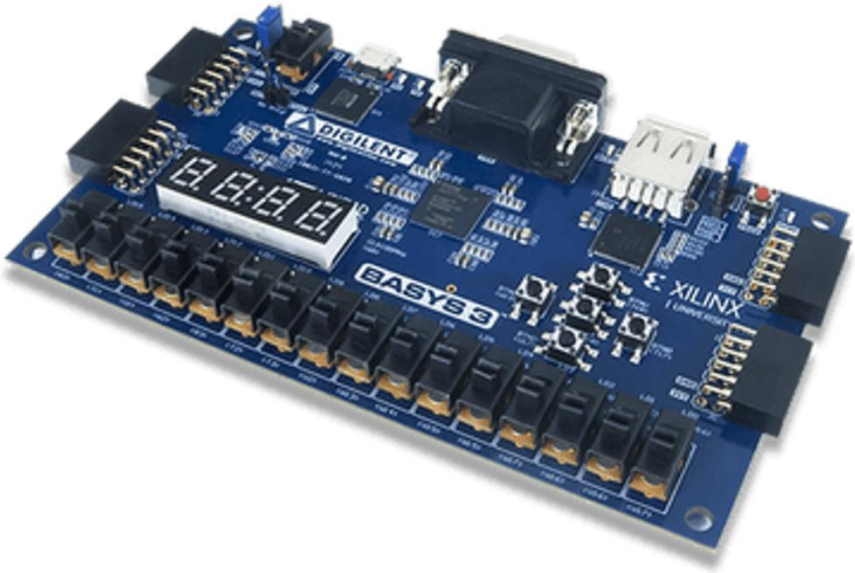


Analog Signal Digital Signal

PROIECTAREA UNUI CIRCUIT DE FILTRARE A SEMNALELOR DIGITALE



MOLDOVAN MARIA
GRUPA 30234, 2023-2024

CUPRINS

1. Introducere

Proiectul are în vedere realizarea unui circuit care permite utilizatorului să proceseze o secvență de valori de intrare X(i) conform următoarei formule:

$$Y(k) = X(k) * a_1 + X(k-1) * a_2 + X(k-2) * a_3$$

Circuitul convertește semnalele primite secvențial în semnale cu valori diferite, pe baza unei formule matematice utilizând constante și semnalele anterioare. Acest program poate fi util într-o varietate de aplicații, spre exemplu procesarea semnalelor, filtrarea datelor sau analiza numerică.

Semnalele sunt reprezentate ca numere întregi de 8 biți. Intrarea va fi recepționată prin setarea pe switch-uri a reprezentării binare a numărului, iar când numărul este pregătit, un buton va fi apăsat, semnalul va fi filtrat, iar rezultatul va fi afișat pe afișor. Sistemul va avea și o opțiune de resetare, de asemenea asignată unui buton, care va șterge conținutul valorilor introduse anterior.

Proiectul a fost dezvoltat în limbajul VHDL, oferind astfel portabilitate și flexibilitate în implementarea pe dispozitivele Xilinx, testat pe plăcuța FPGA Basys 3.

2. Studiu bibliografic

Toate semnalele de intrare vor fi reprezentate intern ca numere binare de 8 biți. Constantele vor fi reprezentate ca numere de 4 biți pentru a ușura calculul. Semnalele de intrare, adică $X(k)$, $X(k-1)$, $X(k-2)$, își vor citi valorile de pe placa de dezvoltare, în timp ce valorile constantelor vor fi declarate în program fără posibilitatea de a le modifica de pe placa de dezvoltare. Valoarea ieșirii, adică Y(k), va fi calculată pe baza formulei.

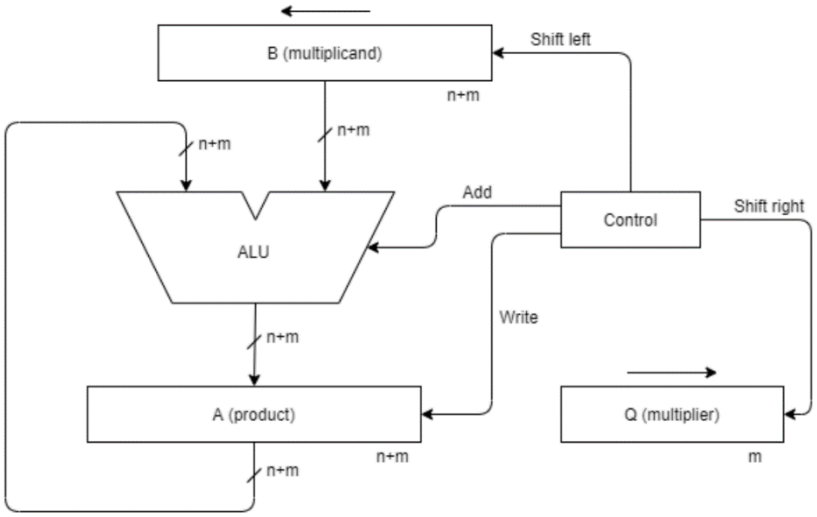
Singurele două operații care vor fi utilizate sunt adunarea și înmulțirea. Pentru adunare, se va folosi un sumator generic de n biți. Pentru înmulțire, o operație puțin mai complexă, metoda aleasă pentru implementare este tehnica de Shift-and-Add, descrisă în documentele de suport ale laboratorului. Algoritmul se bazează pe luarea fiecărei cifre al multiplicandului pe rând și înmulțirea acestuia cu o singură cifră a multiplicatorului. Fiecare produs intermediar este plasat în pozițiile corespunzătoare, la stânga rezultatelor anterioare. La final, toate produsele intermediare sunt adunate pentru a obține rezultatul final.

3. Analiză

Operația analizată și care este realizată în acest proiect este înmulțirea. Componenta care implementează înmulțirea este scrisă ca fiind generică, dar în scopul explicației, se vor folosi numere de 4 biți.

Din multiplele modalități de calcul a produsului a două numere, a fost aleasă tehnica de Shift-and-Add. Metoda este similară cu înmulțirea efectuată pe hârtie. Această metodă adaugă multiplicandul X la el însuși de Y ori, unde Y indică multiplicatorul. Pentru a înmulți două numere cu hârtie și creion, algoritmul este de a lua cifrele multiplicatorului câte una de la dreapta la stânga, înmulțind multiplicandul cu un singur cifru al multiplicatorului și plasând produsul intermediar în pozițiile corespunzătoare, la stânga rezultatelor anterioare.

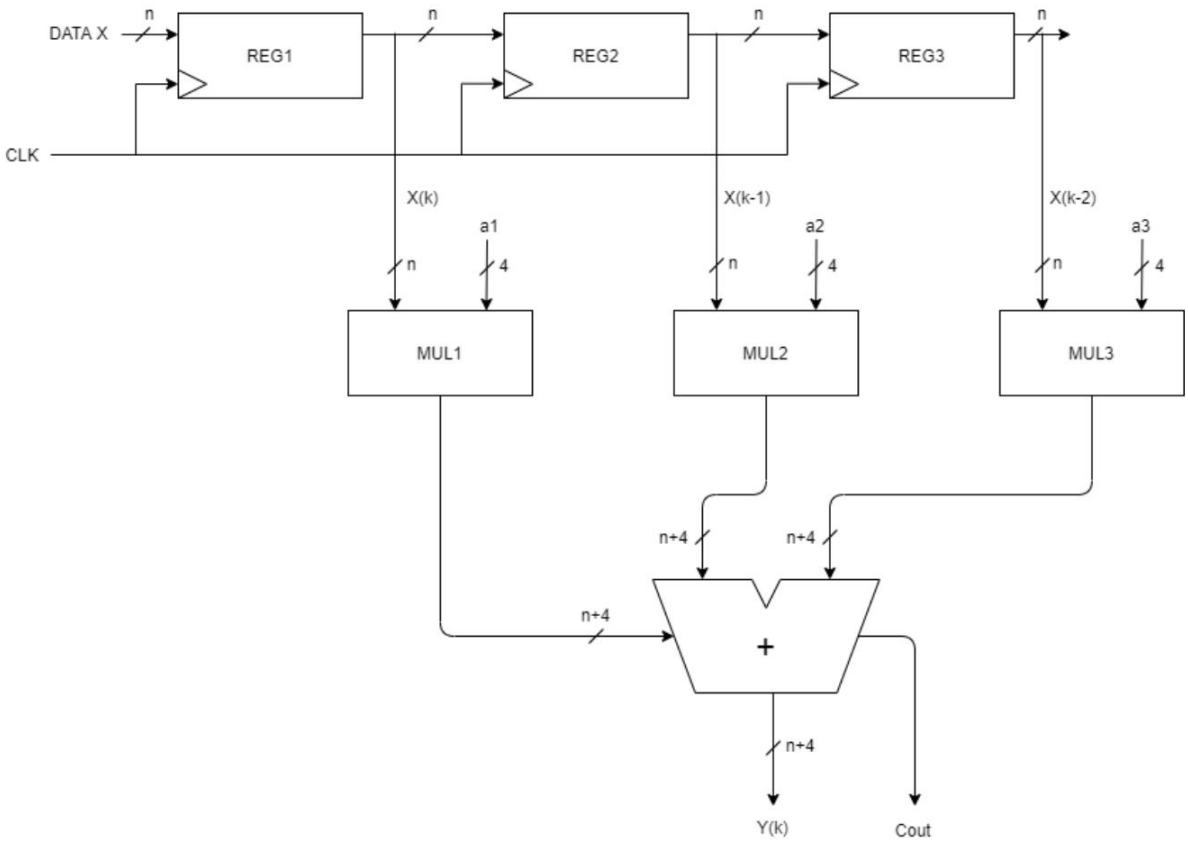
```
1001*
1010
0000
1001
0000
1010
1100010
```



Registrul de produs de n+m biți este inițializat cu 0. Deoarece algoritmul de bază deplasează înregistrul multiplicandului la stânga cu o poziție la fiecare pas pentru a alinia multiplicandul cu suma acumulată în registrul de produs, se folosește un registrul de multiplicand de n+m biți, plasând în el valoarea extinsă a multiplicandului.

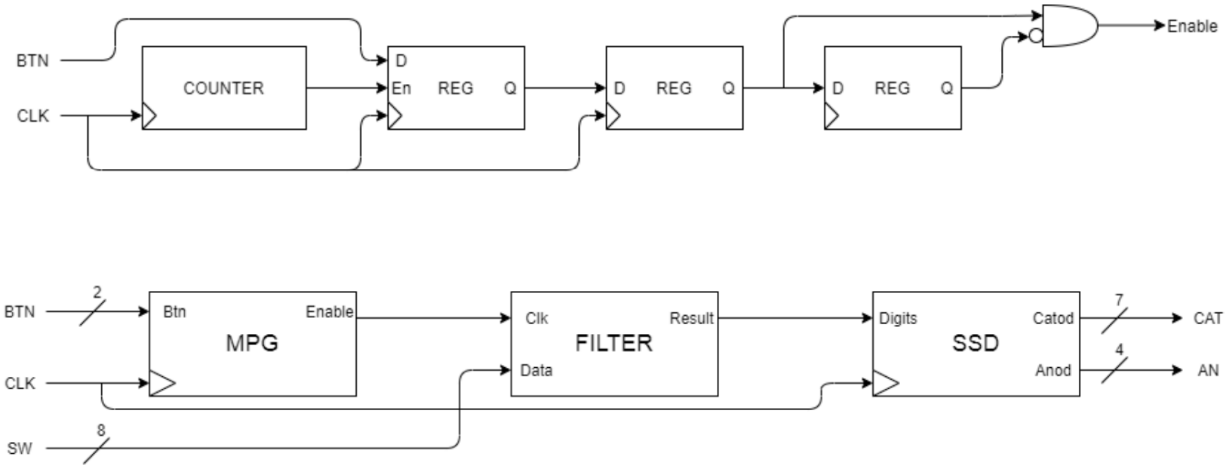
4. Proiectare

Structura generală a circuitului este vizibilă în figura următoare.



Circuitul constă în trei multiplicatoare de tip shift-and-add prezentate în capitolul anterior, un sumator și trei registre de date.
Pe portul de intrare DATA, fluxul de date intră secvențial la fiecare ciclu de ceas. În cele trei registre, ultimele trei valori sunt stocate și sunt disponibile pentru calcularea rezultatului. Conținutul registrelor este deplasat la dreapta la fiecare ciclu de ceas de la un registru la următorul, iar după trei perioade de ceas, valoarea este pierdută. Fiecare dintre cele trei numere introduse ultimele, adică X(k), X(k-1), X(k-2), este înmulțit cu o constantă, adică a1, a2, a3, iar rezultatele sunt adunate în adunător. Rezultatul final este suma, adică Y(k), împreună cu carry out, adică C_out (în cod).
Ceea ce a fost prezentat până acum reprezintă strict partea computațională a proiectului, dar funcționalitatea acestuia va fi testată pe placa de dezvoltare, așa că au fost adăugate încă două componente pentru implementarea citirii datelor de intrare și afișarea rezultatelor.

Cele două componente noi sunt un generator mono-puls și afișajul cu șapte segmente, iar structura proiectului final este vizibilă în figura următoare.



5. Implementare

Sistemul a fost proiectat structural, prin urmare, proiectul va conține componente care implementează funcționalități diferite: componenta pentru sumator, componenta pentru multiplicator și componenta pentru întregul sistem în care cele două componente anterioare sunt mapate. Deoarece proiectul va fi testat pe placa de dezvoltare, au fost necesare două componente suplimentare: un generator mono-puls și un afișaj cu șapte segmente.

Unele componente sunt scrise într-un mod generic astfel încât să poată fi reutilizate în cazul unor modificări viitoare care vor schimba lățimea căii de date.

1. Sumator

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_signed.all;
4  use ieee.numeric_std.all;
5
6  entity adder is
7      generic (
8          n : integer := 8
9      );
10     port (
11         -- in
12         A: in std_logic_vector(n - 1 downto 0);
13         B: in std_logic_vector(n - 1 downto 0);
14         C: in std_logic_vector(n - 1 downto 0);
15         -- out
16         S: out std_logic_vector(n - 1 downto 0);
17         C_out: out std_logic
18     );
19 end adder;
20
21 architecture Behavioral of adder is
22 -- Signals
23     signal result: std_logic_vector(n downto 0) := (others => '0');
24 begin
25     sum: process(A, B, C)
26     begin
27         result <= std_logic_vector(resize(signed(A + B + C), result'length));
28     end process sum;
29     C_out <= result(n);
30     S <= result(n - 1 downto 0);
31 end Behavioral;
```

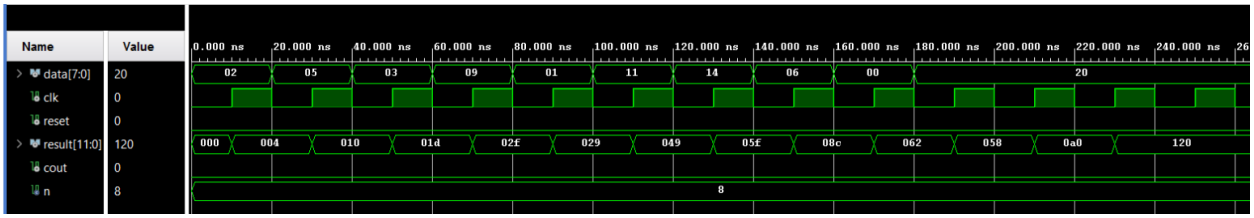
2. Înmulțitor

```
architecture Behavioral of multiplier is
begin
    process(X, Y)
        variable A: std_logic_vector(n + m - 1 downto 0); -- accumulator
        variable B: std_logic_vector(n + m - 1 downto 0); -- multiplicand
        variable Q: std_logic_vector(m - 1 downto 0);     -- multiplier
        variable i: integer;
    begin
        A := (others => '0');
        B := std_logic_vector(resize(signed(X), B'length));
        Q := std_logic_vector(resize(signed(Y), Q'length));
        i := m;

        while (i /= 0) loop -- p != 0
            if Q(0) = '1' then
                A := A + B;
            end if;
            B := B((B'length - 2) downto 0) & '0'; -- shift right
            Q := '0' & Q(m - 1 downto 1); -- shift right
            i := i - 1;
        end loop;

        result <= A;
    end process;
```

3. Filtru

```
process (clk, reset)
begin
    if reset = '1' then
        x_k <= (others => '0');
        x_k_1 <= (others => '0');
        x_k_2 <= (others => '0');
    elsif (rising_edge(clk)) then
        x_k <= data;
        x_k_1 <= x_k;
        x_k_2 <= x_k_1;
    end if;
end process;

-- X(k) * a1
prod_1: multiplier generic map (n => n, m => a1'length)
                            port map (X => x_k, Y => a1, result => mul_1);

-- X(k - 1) * a2
prod_2: multiplier generic map (n => n, m => a2'length)
                            port map (X => x_k_1, Y => a2, result => mul_2);

-- X(k - 2) * a3
prod_3: multiplier generic map (n => n, m => a3'length)
                           port map (X => x_k_2, Y => a3, result => mul_3);

-- X(k) + X(k - 1) * a2 + X(k - 2) * a3
add: adder generic map (n => n + m) -- m = pe cati biti ii constanta
                port map (A => mul_1, B => mul_2, C => mul_3, S => result, C_out => C_out);
```

6. Testare și validare



Y(k) = X(k) * a1 + X(k-1) * a2 + X(k-2) * a3.
a1 = 2; a2 = 3; a3 = 4 -> constante

Trasarea algoritmului, cu rezultate în hexadecimal:

Step 1:
X(k) = 2; X(k-1) = 0; X(k-2) = 0
Y(k) = 2 * 2 + 0 * 3 + 0 * 4 = 4
Step 2:
X(k) = 5; X(k-1) = 2; X(k-2) = 0
Y(k) = 5 * 2 + 2 * 3 + 0 * 4 = 10
Step 3:
X(k) = 3; X(k-1) = 5; X(k-2) = 2
Y(k) = 3 * 2 + 5 * 3 + 2 * 4 = 1d
Step 4:
X(k) = 9; X(k-1) = 3; X(k-2) = 5
Y(k) = 9 * 2 + 3 * 3 + 5 * 4 = 2f
Step 5:
X(k) = 1; X(k-1) = 9; X(k-2) = 3
Y(k) = 1 * 2 + 9 * 3 + 3 * 4 = 29
Step 6:
X(k) = 11; X(k-1) = 1; X(k-2) = 9
Y(k) = 11 * 2 + 1 * 3 + 9 * 4 = 49
Step 7:
X(k) = 14; X(k-1) = 11; X(k-2) = 1
Y(k) = 14 * 2 + 11 * 3 + 1 * 4 = 5f
Step 8:
X(k) = 6; X(k-1) = 14; X(k-2) = 11
Y(k) = 6 * 2 + 14 * 3 + 11 * 4 = 8c
Step 9:
X(k) = 0; X(k-1) = 6; X(k-2) = 14
Y(k) = 0 * 2 + 6 * 3 + 14 * 4 = 62
Step 10:
X(k) = 20; X(k-1) = 0; X(k-2) = 6
Y(k) = 20 * 2 + 0 * 3 + 6 * 4 = 58

7. Concluzii

În concluzie, proiectul funcționează conform așteptărilor noastre, fapt ce reiese din partea de validare și testare.

În ceea ce privește îmbunătățirile viitoare, se poate lua în considerare extinderea lățimii căii de date. Codul poate fi adaptat ușor, deoarece majoritatea componentelor au fost proiectate într-un mod generic. Din păcate, acest lucru poate fi realizat doar utilizând o placă de dezvoltare diferită care are mai multe afișaje cu șapte segmente, deoarece rezultatele vor fi considerabil mai mari și afișarea ar fi imposibilă pe o placă Basys 3. O altă modificare posibilă este ajustarea formulei matematice după care datele de intrare sunt filtrate.

8. Bibliografie

1. https://users.utcluj.ro/~vcristian/AC.html

2. Structura Sistemelor de Calcul, 2023-2024, laboratoare

3. stackoverflow.com

4. support.xilinx.com