

# CIRCUIT DE FILTRARE A SEMNALELOR DIGITALE

Structura Sistemelor de Calcul

Ilovan Mara Gabriela

Grupa 30232

Technical University of Cluj Napoca

# Cuprins

<b>1. Introducere .....</b>	<b>3</b>
1.1 Context .....	3
1.2 Specificatii .....	3
1.2 Obiective .....	3
<b>2. Studiu bibliografic .....</b>	<b>4</b>
<b>3. Analiza.....</b>	<b>6</b>
<b>4. Proiectare.....</b>	<b>11</b>
<b>5. Implementare .....</b>	<b>14</b>
<b>6. Testare/ Experimente .....</b>	<b>16</b>
<b>7. Concluzii.....</b>	<b>17</b>

# 1. Introducere

## 1.1 Context

Scopul acestui proiect este implementarea unui circuit de filtrare a semnalelor digitale. Astfel, circuitul trebuie să poată face transformarea unei secvențe de valori de intrare pe baza unei formule matematice.

În procesarea semnalelor, funcția unui filtru este aceea de a elimina părțile nedorite din semnal, cum ar fi zgomotul aleatoriu sau extragerea unor părți utile ale semnalului cum ar fi componentele care se afla într-un anumit interval. Filtrele digitale operează cu semnale digitale reprezentate sub forma de numere binare.

Există o multitudine de tipuri de filtre care pot fi utilizate pe scară largă în aplicații cum ar fi prelucrarea semnalelor audio, procesarea imaginilor, prelucrarea semnalelor biomedicale și multe altele.

## 1.2 Specificații

Proiectul va fi implementat pe o plăcuță FPGA Basys3 cu arhitectura Xilinx Artix-7 cu ajutorul mediului de dezvoltare Vivado în limbajul de descriere hardware VHDL.

Astfel, este dorită proiectarea unui circuit de filtrare a semnalelor digitale. Secvența de valori de intrare, date citite dintr-un fișier, se va transforma pe baza unei formule de genul:

$$Y(k) = X(k) \cdot a_1 + x(k-1) \cdot a_2 + X(k-2) \cdot a_3$$

Datele de intrare vor fi preluate dintr-un fișier de intrare, iar rezultatele se vor scrie în niște fișiere de ieșire. Vom presupune că datele provin de la niște senzori de temperatură, dar le avem deja stocate în fișiere.

## 2.3. Obiective

Obiectivul proiectului este crearea unei aplicații funcționale care prelucrează niște date dintr-un fișier, reprezentând semnale în timp și folosește un filtru pentru a da un output într-un alt fișier de ieșire. Am ales implementarea unui filtru digital Moving Average- tehnica de prelucrare

a semnalelor pentru a elimina zgomotele din datele secvențiale, principiul acestuia fiind acela de a calcula media unui număr de puncte de date adiacente, filtru prag care va bloca semnalele care trec de o valoare data (prag), filtru care umple datele lipsa dintr-un fișier folosind interpolarea Lagrange. Datele de intrare vor fi mai multe temperaturi, măsurate zilnic. Implementarea proiectului va cuprinde folosirea protocolului Axi 4 Stream pentru transmiterea datelor in pipeline, datele sunt preluate si procesate in mod continuu. Acest protocol ne va ajuta sa avem o comunicare eficienta intre modulele hardware, contribuind la performanta si optimizare proiectului.

## 2. Studiu bibliografic

Filtrarea digitală e una dintre cele mai importante tool-uri din DSP (Digital Signal Processing). Acestea vin cu multe avantaje precum eliminarea erorilor din filtru asociate cu fluctuațiile componente în timp, prelucrarea datelor digitale în diferite moduri sau pentru evidențierea anumitor caracteristici ale semnalului. Eliminarea zgomotului din semnale este foarte importantă pentru obținerea unei informații mai clare și mai precise.

Filtrul digital ia o secvență de date și produce o nouă secvență de date. Ele sunt reprezentate doar de o formulă matematică care ne permite să transformăm un semnal digital în altul.

Această diagramă bloc ilustrează ideea de bază a unui filtru:



Fig 1. Funcționarea unui filtru oarecare

Filtrele digitale permit o reproducere exactă a semnalului filtrat deoarece operația se bazează pe un calcul matematic, oferind astfel și consistența rezultatelor filtrării. Acestea pot fi adaptate cu ușurință la schimbările de mediu sau la variațiile semnalului fără necesitatea unor schimbări majore în designul hardware. Acestea pot fi implementate pe dispozitive FPGA, implementarea acestora necesitând utilizarea unor algoritmi matematici în funcție de cerințele proiectului.

Există două mari tipuri de filtre:

- FIR – Finite Impulse Response. Raspunsul lor la un semnal de intrare este determinat doar de o cantitate finita de date de intrare. Sunt usor de implementat, putand avea precizie. Preferate in unele aplicatii datorita predictibilitatii lor.
- IIR – Infinite Impulse Response. Raspunsul lor la semnal depinde de o secventa infinita de date de intrare si iesire. Sunt mai complexe in proiectare, dar pot fi mult mai eficiente. Structura acestora permite proiectarea filtrelor mai complexe comparativ cu FIR, deoarece sunt recursive acestea au o eficienta mult mai mare in numarul operatiilor necesare pentru implementare.

Totodată exista si filtrele trece sus si trece jos, care permit trecerea semnalelor cu o frecventa mai mica/ mai mare decât o anumita frecventa prag.

Filtrele digitale pot fi implementate intr-o varietate mare de platforme hardware, inclusiv FPGA-uri.

Moving Average Filter este un răspuns finit la impuls folosit pentru a netezi semnalul de la fluctuații zgomotoase si ajuta la păstrarea reprezentării adevărate a semnalului sau la păstrarea răspunsului in trepte ascuțite. Astfel, acest tip de filtru calculează media a unui set de valori anterioare si actuale pentru a genera un rezultat filtrat.

Filtrul Prag este un filtru care permite sau blocheaza trecerea valorilor care depasesc sau sunt egale cu o anumita limita, reprezentand pragul, blocand toate valorile sub acest prag deoarece sunt considerate invalide. Aceste filtre au utilizări multiple in aplicatii pentru a prelucra datele in diverse moduri, depinzand de obiectivul prelucarii datelor.

Interpolarea Lagrange este o metoda matematica pentru a găsi un polinom care trece prin punctele de date cunoscute, cu ajutorul acestui polinom pot fii estimate puncte intermediare dintre valorile deja cunoscute. Acest filtru are numeroase utilizări in inginerie precum completarea datelor in situații in care sunt disponibile doar puncte discrete.

Deoarece avem nevoie de o comunicare eficienta si simpla intre modulele hardware, care facilitează transferul continuu de date vom avea nevoie de folosirea protocolului AXI4-STREAM. Utilizând acest protocol, datele se vor procesa in pipeline. Astfel, Axi4Stream utilizează date seriale pentru a transmite informația, acest lucru reducând numărul de pini necesari pentru conexiunea intre modulele hardware. Cu ajutorul acestui protocol vom transmite datele filtrului.

Cu ajutorul Axi4-Stream vom avea un transfer de date simplu si eficient:

- Flux continuu de date- facilitând transferul de date flux continuu, este ideal pentru aplicațiile care implica un flux constant de date, precum implementarea acestor filtre care primesc mai multe date dintr-un fisier.
- Fără informații de adresa- nu necesita informații de adresa pentru fiecare set de date transmis, făcând comunicațiile mult mai eficiente.

In cazul filtrelor, acest protocol permite conectarea acestor module, într-un mod mai usor, facilitând interacțiunea între ele.

Modul de funcționare al acestui protocol este descris în următoarea schema:

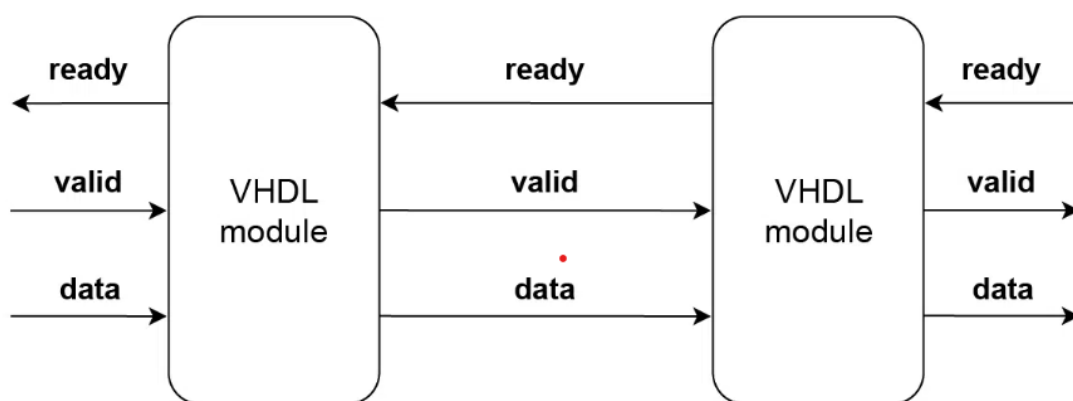


Fig. 2 Conectarea a doua module VHDL

Transferul de date se realizeaza doar daca semnalele ready si valid sunt ambele active pe 1 in timpul aceliasi ciclu de ceas.

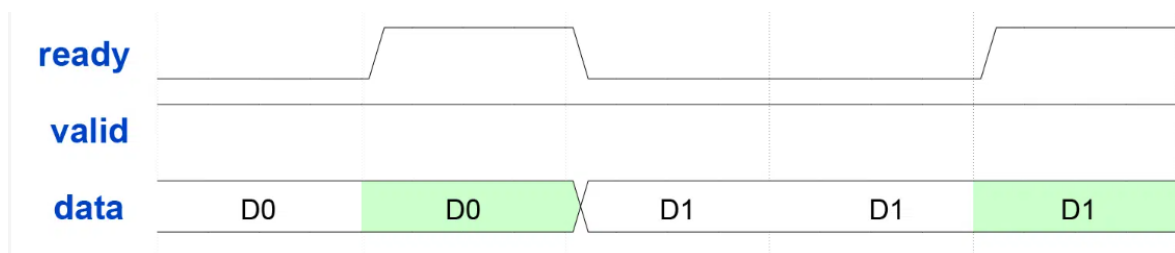


Fig. 3 Transferul de date

### 3. Analiza

Proiectul ales reprezintă un circuit de filtrare a circuitelor digitale. Pentru acest proiect am ales proiectarea și implementarea a 3 filtre: filtrul de medie mobilă, interpolarea Lagrange și filtru prag folosind protocolul Axi4-Stream.

Principala funcționalitate a proiectului este prelucrarea datelor de temperatura: filtrarea acestora utilizând filtrul de medie mobila pentru a elimina zgomotul din datele de temperatura, interpolarea datelor de temperatura prin aplicarea interpolării Lagrange pentru a estima valorile de temperatura lipsa dar si filtrarea selectiva a temperaturii.

Pentru realizarea proiectului ar trebui ca prima data sa citim datele despre temperatura din fișier folosind operațiile de intrare ieșire.

Pentru extragerea datelor trebuie sa le parsam. In cazul filtrului implementat cu interpolarea Lagrange ar trebui sa găsim datele lipsa din fișier.

Analiza proiectării filtrelor alese:

Pentru primul filtru Average Moving Filter as avea nevoie de un Shift Register in care pot retine o serie de valori anterioare. După care as avea nevoie de un mecanism de calcul al mediei. Astfel, ar fi nevoie de un sumator, care aduna valorile, după care un împărțitor, care sa împartă suma la numărul elementelor însumate.

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

Pentru al doilea filtru Interpolare Lagrange avem nevoie de niște registre pentru stocarea datelor cunoscute. Totodată pentru calculele din formula avem nevoie de un înmulțitor si un sumator.

$$f_{n-1}(x) = \sum_{i=1}^n L_i(x) f(x_i) \quad \text{where} \quad L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}$$

$f(x_i)$  reprezintă temperaturile cunoscute, in timp ce  $x_i$  reprezintă timpul corespunzător in care a fost măsurată temperatura sau valorile indicilor.

Pentru filtrul de prag este necesar un comparator pentru a compara fiecare valoare cu pragul specificat. Acesta evaluează daca o temperatura depășește sau nu valoarea de prag si va emite un semnal corespunzător.

La toate aceste filtre trebuie conectat la modul AXI4-Stream pentru a permite transmiterea si primirea datelor de temperatura folosind acest protocol. Pentru asta trebuie sa definim interfețele Axi4-Stream. Va trebui sa realizam conexiunile intre filtre si aceste interfețe, astfel trebuie asociate intrările si ieșirile la interfețele AXI4-Stram. Semnalele de date (TData),

semnalul de validare (Tvalid) si semnalul de pregătire (TReady) trebuie sa fie conectate in cadrul fiecărui modul, acest lucru fiind necesar pentru a transmite si recepționa datele corect.

Astfel, datele de intrare vor fi conectate la semnalul TData, semnalul TValid va verifica valabilitatea datelor, iar semnalul TReady va indica daca modulul e pregătīt sa primească datele.

### Inmultire: Shift and Add

Pentru operație de înmulțire am ales sa utilizez metoda: Shift and Add Multiplication.

Înmulțirea consta in testarea succesiva a biților înmulțitorului, începând cu bitul cel mai semnificativ. Daca bitul e 1, produsul parțial corespunzător este copia de înmulțitului, ian in caz contrar e 0. Produsele parțiale sun însumate. Fiecare rezultat parțial va fi deplasat la dreapta.

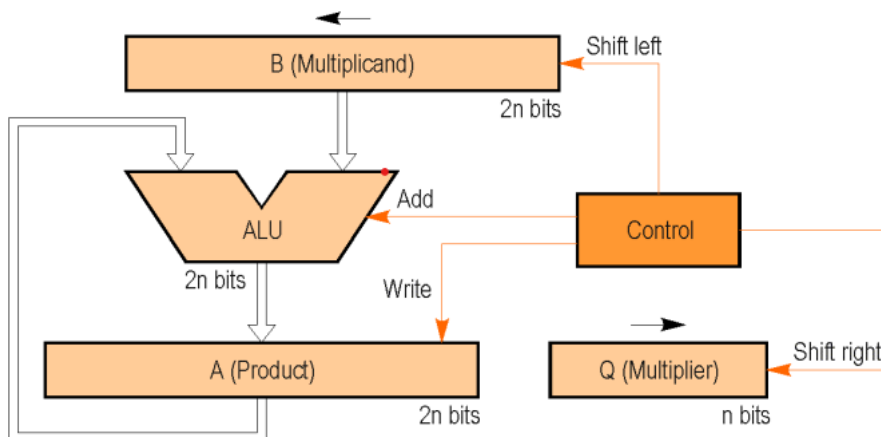


Fig. 4 Inmputior

Registrul A este utilizat ca si acumulator pentru păstrarea produsului parțial. La sfarsitul operatiei in A vom regăsi partea mai semnificativa a produsului. Acest registru trebuie sa aiba posibilitatea de deplasare la dreapta. In registrul B vom pastra deinmultitul, continutul acestuia transferadu-se la una dintre intrările sumatorului

Algoritmul fiind reprezentat de figura urmatoare:



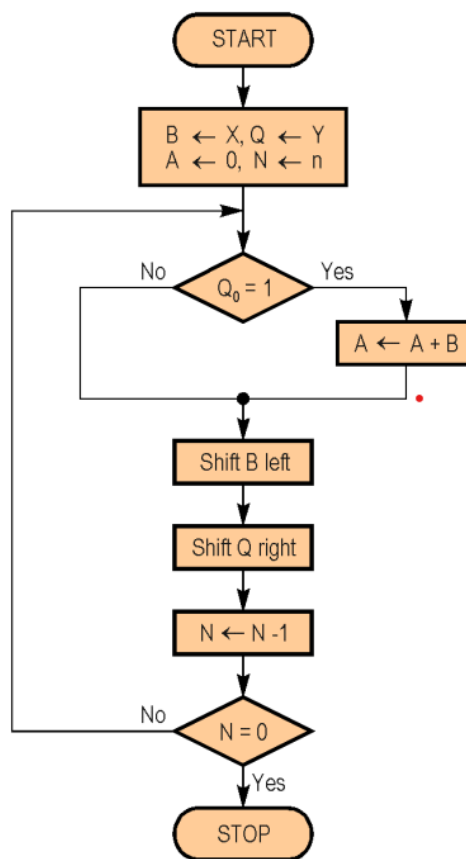


Fig. 5 Functionarea unui inmultitor

### Impartire: Restoring Division

Impartirea binara: o serie de scaderi ale impartitorului din restul partial, care se efectueaza doar daca restul partial e mai mare decat impartitorul, caz in care cifra catului e 1, altfel, cifra corespunzătoare catului e 0.

Algoritmul e bazat pe adunări si scăderi repetate pentru a obține rezultatul. Schema e la fel cu cea a înmulțitorului. Fiecare etapa a operației începe cu o deplasare la stânga a restului parțial. Duca care se efectuează scăderea împărțitorului din restul parțial, obținându-se, astfel, noul rest parțial. Dacă se obtine un numar pozitiv, cifra catului e 1, altfel e 0. Înmulțitorul e adunat la restul parțial pentru a-l reface. Operațiile se vor repete de n ori.

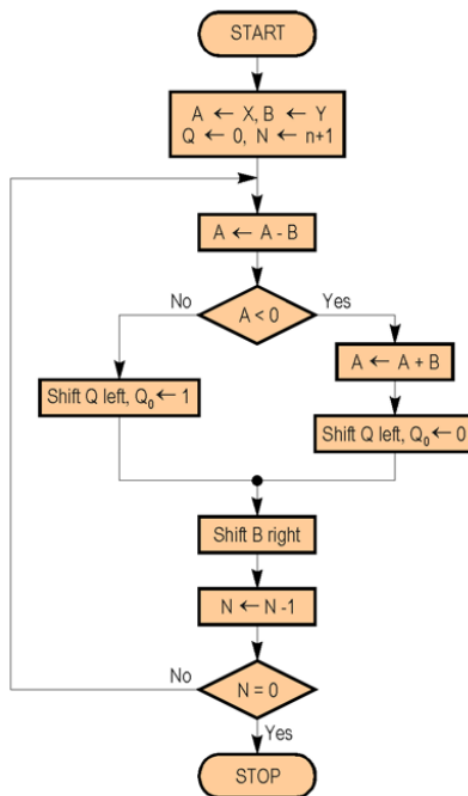


Fig. 6 Functionarea unui impartitor

Pentru ca acesteasa funtioneze si pe numere negative a fost necesara adaugarea unor conditii in plus.

Fiecare modul care foloseste Axi4-Stream pentru transmiterea datelor are nevoie de 3 semnale: tdata, tready, tdata:

- Tdata – Acest semnal e folosit pentru treasmiterea datelor intre modulele hardware, deoarece datele cu care lucrez au 16 biti, tdata va fi mereu 16 biti.
- Tvalid – Indica daca datele sunt valide
- Tready – Indica faptul ca modulul destinatar e pregatit sa primeasca datele

Fiecare dintre datele de intrare trebuie sa aiba semnale de tvalid, tready, tdata. Astfel modulul receptor monitorizeaza semnalele tvalid si tdata pentru a primi datele, iar semnalul tready e setat atunci cand modulul receptor este pregatit sa primeasca datele, modulul receptor incepand prelucrarea datelor primite conform logicii sale specifice. Deoarece modulele fuctioneaza la diferite cicluri de ceas este nevoie de adaugarea unui FIFO intre componente. Astfel intre toate comonentele e necesar un FIFO pentru a stoca datele intre module, asigurand astfel sincronizarea datelor. Dupa ce datele au fost preluate din FIFO, acesta este pregatit pentru urmatoarele date.

În cazul modulelor Axi4-Stream care efectuează operații simple într-un singur ciclu de ceas, comportamentul intern poate fi reprezentat sintetic ca o mașină de stări finite (FSM) cu două stări. Într-o stare, modulul acceptă datele și efectuează operațiile necesare, în timp ce cealaltă stare furnizează rezultatul de ieșire.

Astfel avem nevoie de două stări READ OPERANDS – modulul este pregătit să preia valorile primite la intrare și efectuează operația reală, apare când toate semnalele tvalid sunt 1 și WRITE RESULT – modulul furnizează rezultatul la ieșire și indică că ieșirea este validă, apare când toate semnalele tready sunt 1.

În schimb pentru modulele care au nevoie de mai multe cicluri de ceas pentru a se efectua vom avea nevoie de un FSM cu mai multe stări. Avem nevoie de Axi4-Stream întrucât proiectul implică procesarea de semnale, semnale provenite de la niște senzori, iar pentru asta este necesară prelucrarea datelor în flux.

Citirea datelor din fișier și scrierea acestora se face folosind biblioteca STD.TEXTIO. Datele se citesc pe rând utilizând funcția readline, folosit pe un fișier deschis în modul de citire. Datele se citesc din linie folosind funcția read. După deschiderea fișierelor este necesară și închiderea acestora.

## 4. Design:

Componentele necesare pentru implementarea acestui proiect:

1. Adder: Folositor pentru filtrul de medie și pentru interpolarea Lagrange.
2. Înmulțitor- Shift and Add Multiplier: Folositor pentru filtrul care aproximează temperaturi lipsă folosind interpolarea Lagrange.
3. Comparator: Folositor pentru compararea datelor pentru filtru prag. Acesta conține un scăzător, la final valoarea diferenței este comparată cu 0.
4. Împartitor: Folositor pentru filtru de medie și pentru filtrul care aproximează valori de temperaturi lipsă folosind interpolarea Lagrange.
5. Filtrele: Acestea sunt alcătuite din componentele precizate mai sus.
6. O componentă pentru selectarea filtrului cu care dorim să filtrăm datele de intrare
7. FIFO: Aceasta componentă nu o voi implementa eu, o voi lua din IP Core. Aceasta componentă este folosită pentru separarea modulelor 2hardware.

Toate componentele trebuie sa aiba semnalele specifice protocolului Axi4-Stream tvalid, tready si tdata.

Aceasata componenta o voi folosi intre toate modulele vhd pentru o sincronizare mai buna a datelor intrucat fiecare operatie are un numar diferit de cicluri de ceas ca durata, asa ca FIFO-ul ne va ajuta sa retinem rezultatele in ordinea desfasurarii, iar de fiecare data vom lua primul din coada pentru a continua calculul.

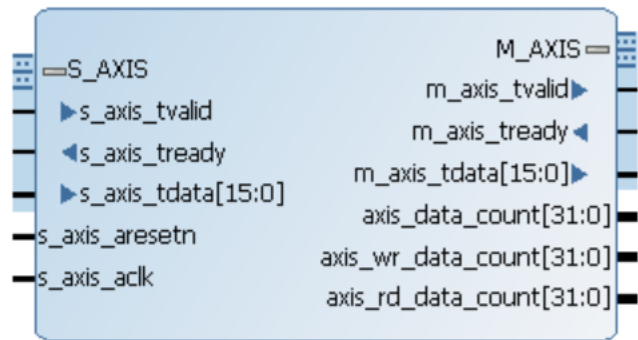


Fig. 8 FIFO

Pentru filtru de medie am ales sa fac media a 4 valori consecutive din fisier. Astfel, as avea nevoie de 3 sumatoare, si un impartitor care imparte rezultatul adunarii finale la numarul de valori. Fiecare componenta va avea semnalele specifice Axi4Stream, pentru fiecare semnal sei intrare avem in tdata si tvalid si out tready, in timp ce pentru semnalele out vom avea in tready si out tdata si tvalid.

Formula utilizata pentru design:  $(x1+x2+x3+x4)/4$

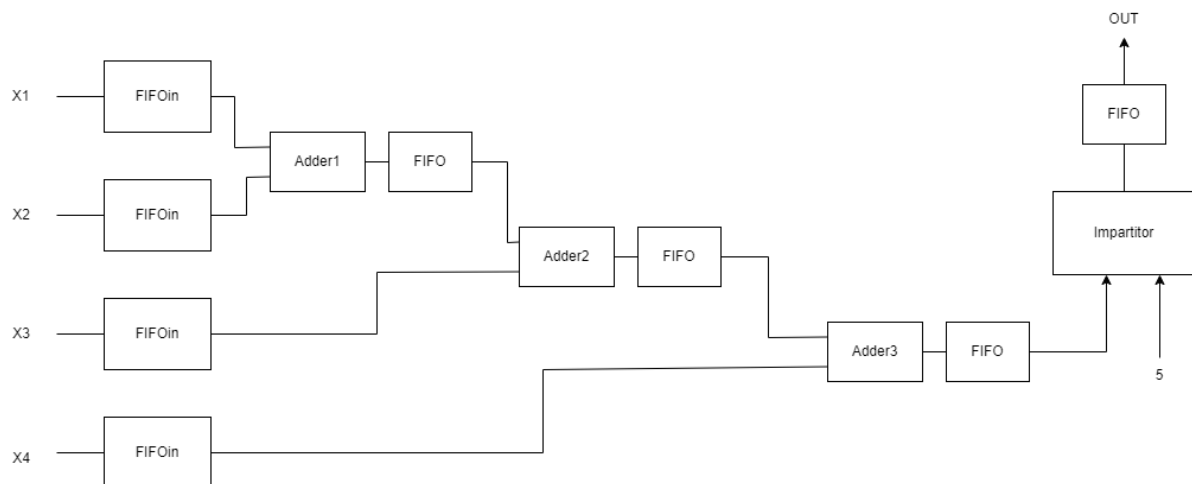


Fig. 9 Diagrama filtrului de medie

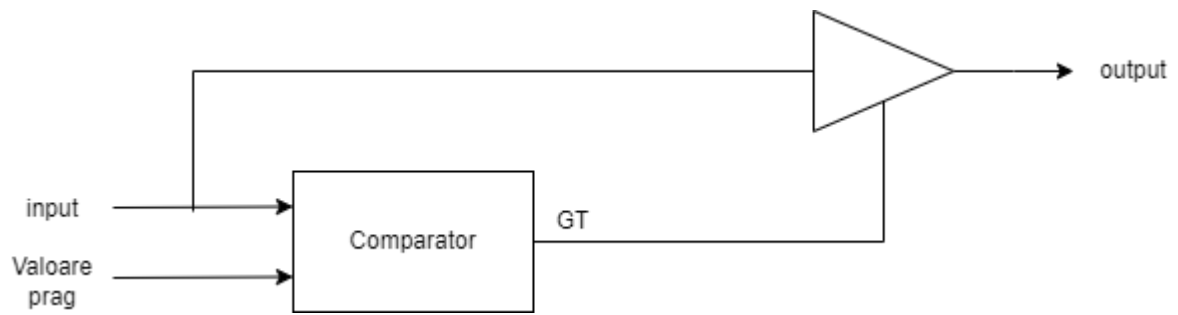


Fig. 10 Diagrama filtrului pag

Formula utilizata:  $(x_{\text{missing}} - x_1)/(x_0 - x_1) * y_0 + (x_{\text{missing}} - x_0)/(x_1 - x_0) * y_1$

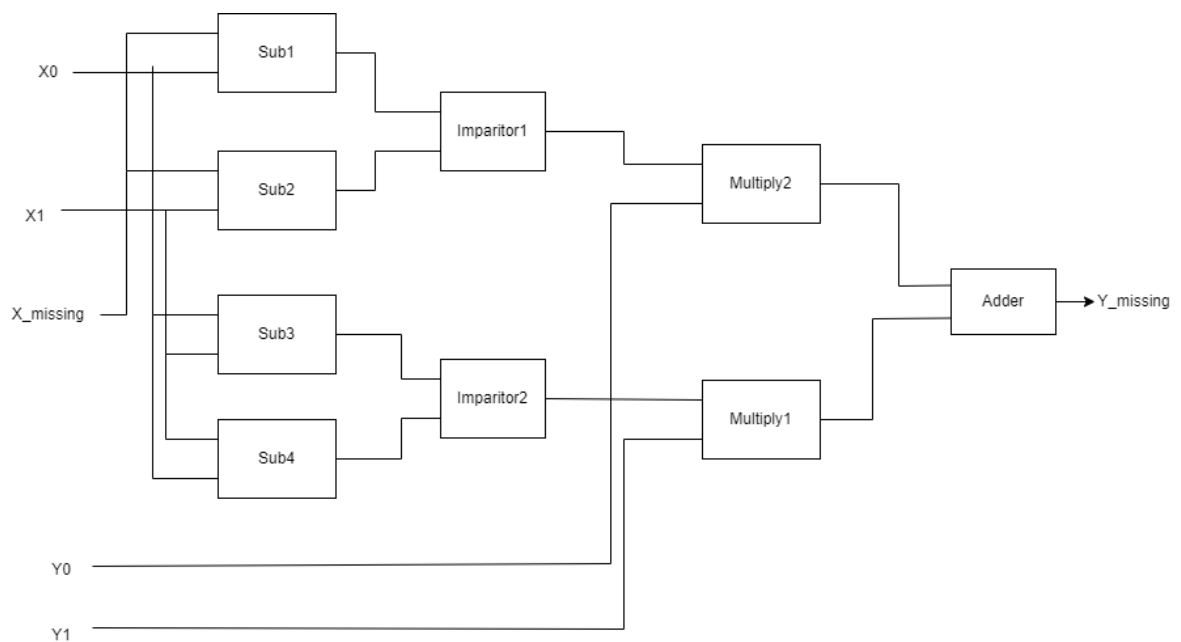


Fig. 11 Diagrama filtrului pentru aproximarea unei temperaturi fara FIFO-uri adaugate

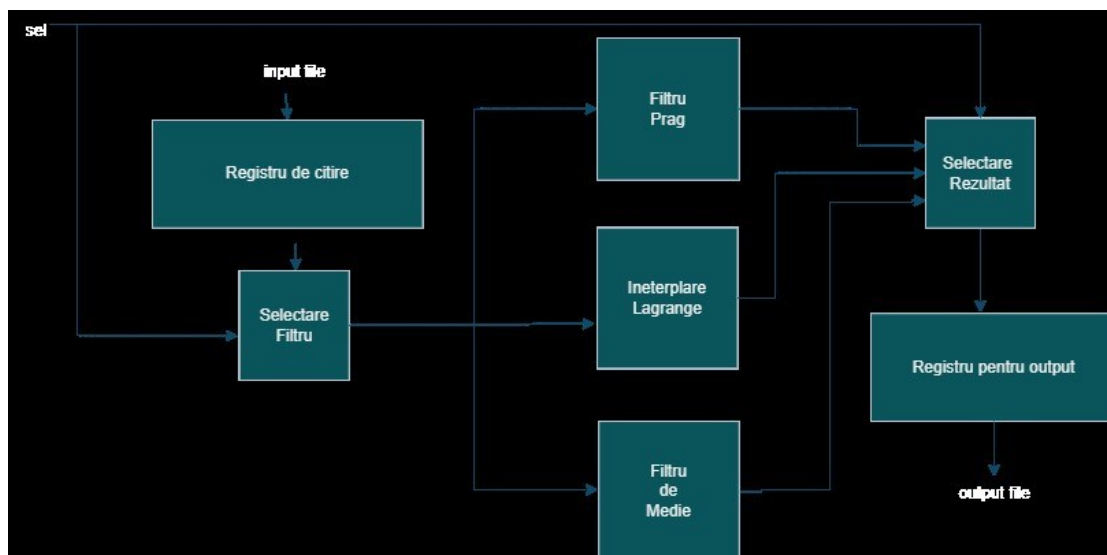
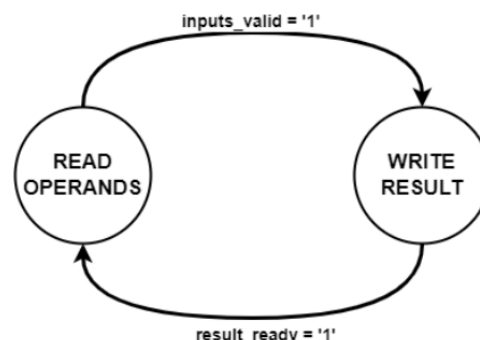


Fig. 12 Diagrama intregului circuit

## 5. Implementare:

### 1. Filtru prag:

Acesta este un filtru care compara un input cu o valoare prag, output-ul va fi valoarea de input doar daca acesta nu depaseste valoarea prag. Pentru implementare am urmat organigrama:



Cand suntem in starea

READ\_OPERANDS: comparam valorile primite ca inputuri, doar daca acestea sunt gata si valide, tready = '1' si tvalid = '1'. Daca valoarea temperaturii e mai mica ca vlaoarea prag vom putea intra in starea de WRITE, altfel vom ramane in starea de READ.

Pentru implemnetare am avut nevoie de un case pentru stari si de niste if-uri pentru verificarea semnalelor de validare si verificare daca sunt gata, logica propriu-zisa a fitrului fiind implementata in READ\_OPERANDS.

### 2. Filtru de Medie:

Pentru acest filtru am implementat un împărțitor, construit după organigrama de la Fig. 6. Astfel, am creat 3 stări: IDLE – pentru inițializare, DIVIDE – realizarea algoritmului propriu-zis, STOP – pentru validarea rezultatului si resetarea componentei, daca este cazul.

A fost nevoie sa adaug semnalele de tvalid si tready pentru fiecare output si input, iar o data cu adăugarea lor, sa adaug si condiții in plus care sa verifice daca datele sunt pregătite si valide pentru a face operația, iar la final sa validez rezultatul. Si sumatorul a avut nevoie de adăugare de astfel de semnale (tvalid, tdata).

In componenta finala am legat 3 sumatoare si un divizor, intre fiecare doua componente am pus un fifo generat din IP Catalog. Astfel, am avut nevoie de 4 fifo-uri in care sa pun inputurile (x1, x2, x3, x4). Pentru prima suma am făcut un port map al sumatorului care aduna x1 cu x2, in continuare rezultatul acestei sume va fi adunat cu x3, iar la final mai avem nevoie de o componenta sumator care aduna a doua suma cu x4. Rezultatul tuturor sumelor va merge in divider, doar după ce va fi scos din fifo-ul adăugat intre acestea.

### 3. Filtru pentru aproximarea datelor lipsa:

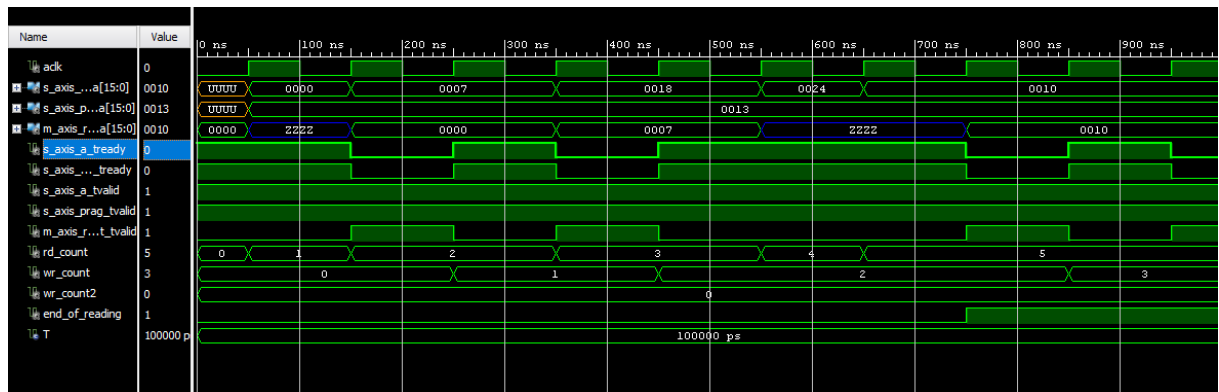
Pentru acest filtru m-am folosit de componentele create anterior, in plus a fost nevoie sa mai implementez un scăzător si un înmulțitor. Înmulțitorul l-am implementat după organigrama din Fig. 5. Astfel, am ales ca algoritmul sa aibă 3 stări: IDLE – inițializare, MULTIPLY – realizarea înmulțirii propriu-zise, STOP – validare rezultat si resetare daca este cazul.

In componenta finala am conectat componentele intre ele conform Fig. 11, adăugând semnalele tvalid, tready si tdata, pentru fiecare input si output al componentelor. Intre fiecare componente am pus un fifo, pentru ca datele sa fie sincronizate intre module. Totodată, a fost necesar si cate un fifo pentru datele de intrare si pentru rezultatul final.

Pentru citirea datelor din fișier am implementat codul intr-un fișier de tip simulare.

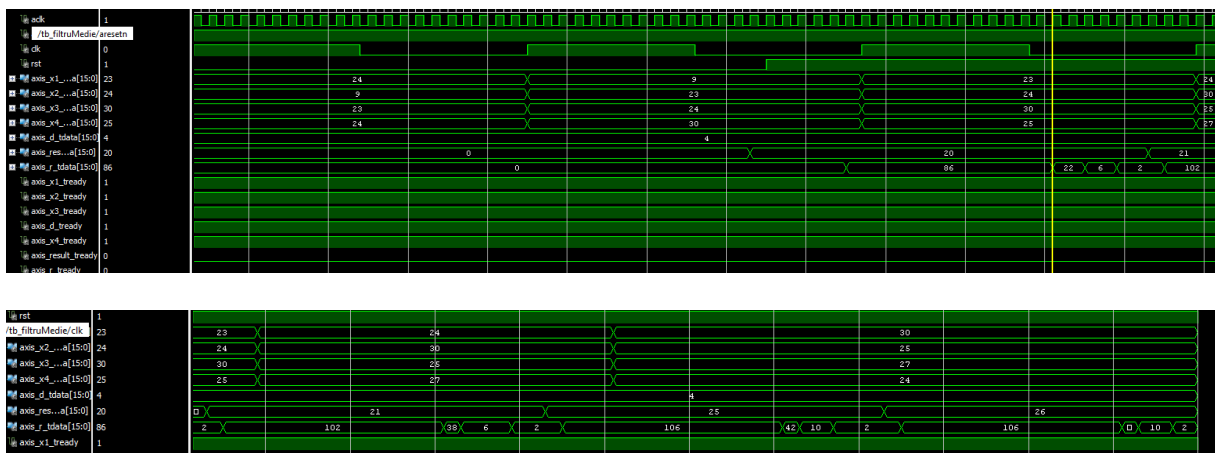
## 6. Testare:

### Filtru prag



Dupa cum observam pe iesire vom avea „ZZZZ” pentru valori mai mari ca 13 (in hexa) sau chiar valoarea, daca valorile sunt mai mici. Scrierea in fisier se face doar daca rezultatul nu e „ZZZZ”.

### Filtru de Medie



Am datele:

$$0000000000011000 = 24$$

$$0000000000001001 = 9$$

$$0000000000010111 = 23$$

$$0000000000011000 = 24$$

$$0000000000011110 = 30$$

$$0000000000011001 = 25$$



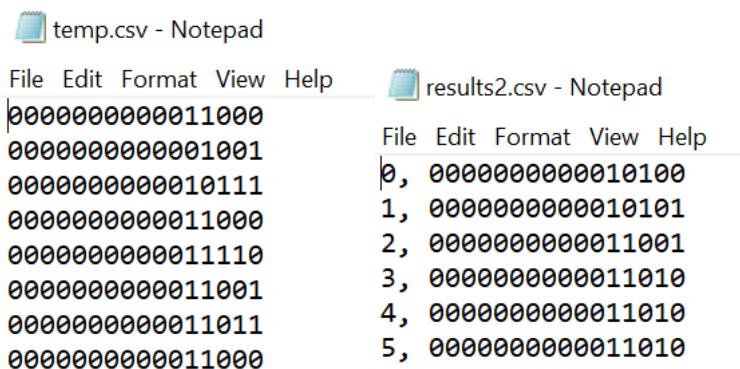
00000000000011011 = 27

00000000000011000 = 24

După cum observam simularea funcționează întrucât:

1.  $(24 + 9 + 23 + 25)/4 = 20$
2.  $(9 + 23 + 24 + 30)/4 = 21$
3.  $(23 + 24 + 30 + 25)/4 = 25$
4.  $(24 + 30 + 25 + 27)/4 = 26$

La simularea componentei care lega firele vom pune in simulare o selecție de 3 biti, fiecare bit fiind corespunzător pentru un filtru. Astfel, bitul 0 ii corespunde filtrului prag, bitul 1 ii corespunde filtrului de medie, iar bitul 2 filtrului care aproximează datele lipsa.



```
temp.csv - Notepad
File Edit Format View Help
00000000000011000
0000000000001001
00000000000010111
00000000000011000
00000000000011110
00000000000011001
00000000000011011
00000000000011000

results2.csv - Notepad
File Edit Format View Help
0, 00000000000010100
1, 00000000000010101
2, 00000000000011001
3, 00000000000011010
4, 00000000000011010
5, 00000000000011010
```

Acesta este un exemplu pentru filtrul de medie, in simulare am pus selectia pe „010”, iar după ce am rulat pentru numărul de nanosecunde potrivit vom vedea in fișierul result datele corecte.

## 7. Concluzii

Acest proiect s-a axat pe implementarea unor filtre digitale specifice intr-un mediu FPGA. Protocolul AXI4-Stream e considerat esențial pentru comunicarea eficienta intre modulele hardware. Au fost necesara implementarea unor algoritmi precum înmulțirea si împărțirea pentru realizarea operațiilor corespunzătoare formulei alese.

Proiectul este o aplicație pentru prelucrarea semnalelor digitale provenite de la senzori de temperatura. Pe viitor acest proiect ar putea fi extins. Astfel, fiecare filtru ar putea avea mai multe semnale de intrare pentru o funcționare mai buna, iar formulele ar putea fi realizate pe numere reale, pentru ca filtrele sa se potrivească mai bine pe datele din lumea reala.

## Bibliografie

1. How the AXI-style ready/valid handshake works – VHDL whiz  
<https://vhdlwhiz.com/how-the-axi-style-ready-valid-handshake-works/>
2. Introduction to Digital Filters – University of California, Davis  
[https://123.physics.ucdavis.edu/week\\_5\\_files/filters/digital\\_filter.pdf](https://123.physics.ucdavis.edu/week_5_files/filters/digital_filter.pdf)
3. Section 6 – Digital Filters – Analog Devices  
[https://www.analog.com/media/en/training-seminars/design-handbooks/mixedsignal\\_sect6.pdf](https://www.analog.com/media/en/training-seminars/design-handbooks/mixedsignal_sect6.pdf)
4. AXI4-Stream Interface- Advanced Micro Devices  
<https://docs.xilinx.com/r/en-US/pg256-sdfec-integrated-block/AXI4-Stream-Interface>
5. Shift and Add Multiplication - Dr. Baruch Zoltan Francisc  
[https://users.utcluj.ro/~baruch/book\\_ssce/SSCE-Shift-Mult.pdf](https://users.utcluj.ro/~baruch/book_ssce/SSCE-Shift-Mult.pdf)
6. Using Files in VHDL - NandLand  
<https://nandland.com/file-input-output/>
7. <https://surf-vhdl.com/read-from-file-in-vhdl-using-textio-library/>

