DOCUMENTAȚIE

Proiectarea unui circuit de filtrare a semnalelor digitale

Student: Flutur Florina

Grupa: 30232

An universitar: 2023/2024

Cuprins:

1. Introducere

1.1 Context

Acest proiect are ca scop proiectarea, implementarea si testarea mai multor valori ale pixelilor, cu scopul de a efectua operatii cu acestea si de a afisa rezultatul cel mai convenabil. Proiectul se concentrează pe furnizarea unei soluții versatilă și eficiente pentru colectarea și analiza valorilor pixelilor.

Filtrarea reprezintă prelucrarea unui semnal în domeniul timpului ce induce o schimbare în modul în care semnalul este alcătuit în funcție de frecvențe. Schimbarea constă în reducerea sau eliminare anumitor componente: filtrele permit anumitor frecvente să treacă si le atenuează pe restul. [1] Acestea oferă control asupra semnalului în funcție de cerințele specifice ale aplicației.

1.2 Specificatii

Pentru realizarea proiectului vom realiza codul in mediul de dezvoltare vivado, iar dispozitivul va fi implementat pe o placuta nexys4 de la Digilent. Caracteristicile esențiale pentru acest proiect includ 2 display-uri cu 4 cifre de cate sapte segmente, 5 butoane de apăsare, 16 LED-uri și comutatoare, ceas intern și componente pentru comunicarea cu calculatorul. Rezultate obtinute vor fi vizualizate pe display, iar interacțiunea cu filtrul va fi facilitată prin comutatoare disponibile, permițând utilizatorului să schimbe filtrele și să monitorizeze rezultatele. [2]

Codul necesar pentru implementare va fi redactat în limbajul de programare VHDL si vom utiliza platforma Vivado 2022.1 pentru programare, simulare și testare. Acest mediu de dezvoltare va facilita procesul de dezvoltare al proiectului, asigurându-ne că codul este eficient și funcțional pe placa Nexys4.

Simulările detaliată în Vivado vor fi efectuate pentru a verifica comportamentul corect al filtrului înainte de implementarea sa pe placa hardware.Proiectul va implica teste pe dispozitiv, evaluând performanța filtrului în condiții reale și ajustându-l pentru a asigura funcționalitatea optimă.

1.3 Obiective

Obiectivul acestui proiect constă în proiectarea și implementarea unui circuit de filtrare a semnalelor digitale, având ca scop principal furnizarea unor semnale filtrate corecte și precise. Dispozitivul dezvoltat va fi capabil să primească semnale digitale, să le filtreze conform specificațiilor predefinite și să afișeze rezultatele pe un display. Implementarea unui filtru digital eficient va implica etapele de proiectare a circuitului, alegerea și implementarea algoritmilor de filtrare, care vor fi in numar de trei, simularea și testarea proiectului. Scopul final este să obținem un circuit de filtrare precis si versatil, potrivit pentru diverse aplicații care necesită prelucrarea semnalelor digitale.

2. Studiu bibliographic

2.1 Filtre

Filtrul numeric sau filtrul digital este un sistem discret caracterizat printr-un algoritm numeric cu ajutorul căruia o secvenţă de intrare se transformă într-o secvenţă filtrată de ieşire. Algoritmul de filtrare poate să reprezinte: o filtrare trece jos, trece sus, trece bandă, opreşte bandă, sau o operaţie de diferenţiere, integrare etc. Filtrele sunt flexible prin modificarea algoritmului ales, usor de proiectat, testat, implementat, sigure in functionare, versatile in prelucrarea semnalelor.[3]

Filtrele digitale pot fi împărţite în două clase, filtre cu răspuns finit la impuls sau FIR şi filtre cu răspuns infinit la impuls sau IIR. Filtrele cu răspuns finit la impuls (FIR) sunt tipuri de filtre digitale caracterizate prin faptul că răspunsul lor la o intrare impuls este finit în durată. Cu alte cuvinte, filtrul produce un răspuns la o intrare impuls care are o durată limitată în timp. Filtrele cu răspuns infinit la impuls (IIR) sunt tipuri de filtre caracterizate prin faptul că răspunsul lor la o intrare impuls are o durată infinită în timp. Acest lucru se traduce prin prezența unui termen recurent în funcția de transfer a filtrului, care permite propagarea semnalului de ieșire înapoi în timp.

Filtrele cu raspuns finit la impuls:

- considerară valorile precedente din semnalul de la intrare
- se stabilizează rapid la zero după încheierea intrării
- folosesc operatii simple de calcul
- sunt mai simplu de analizat si implementat, motiv pentru care sunt cel mai des întâlnite în practică

Filtre cu răspuns infinit la impuls:

- considerară valorile precedente din semnalul de la intrare
- iau în considerare iesirile precedente
- reprezintă relatii de recurentă
- alcătuiesc bucle de feedback [1]

Operația de filtrare reprezintă de regulă trecerea unui semnal x(t) printr-un sistem liniar invariant în timp, a cărui funcție pondere h(t) este cunoscută.

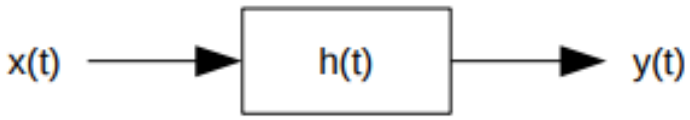


Figura 1. Operatia de filtrare

Un sistem se numește liniar, dacă la intrarea sistemului aplicăm o combinație liniară de două (sau mai multe) semnale ax1(t) + bx2(t), unde a si b sunt constante, atunci semnalul de la ieșirea sistemului poate fi scris ca fiind ay1(t) + by2(t), unde y1(t) și y2(t) reprezintă răspunsurile sistemului la semnalele x1(t) respectiv x2(t).

Dacă răspunsul sistemului este mereu același pentru un același semnal de intrare, indiferent de momentul de timp la care este aplicat semnalul respectiv la intrarea sistemului, atunci sistemul se numește invariant în timp.

Funcția ponderată h(t) asociată sistemului reprezintă răspunsul la impuls al sistemului. Aceasta indică modul în care sistemul reacționează la un impuls aplicat la intrare. Ponderarea funcției h(t) descrie cum sistemul transformă impulsul într-un semnal de ieșire. [4]

Proiectul va fi dirijat dupa formule matematice care se vor aplica asupra unei secvente de valori de intrare, asemanatoare urmatoarei formule:

$$Y(k) = X(k) * a1 + X(k-1) * a2 + X(k-2) * a3$$

Unde:

- Y(k) reprezinta semnalul final de iesire
- X(k), X(k-1), X(k-2) reprezinta semnale diferite de intrare
- a1,a2,a3 reprezinta valori constante

2.2 Protocolul de comunicare AXI4-Stream

Un protocol de comunicare este un sistem de reguli care permite două sau mai multe entități ale unui sistem de comunicații să transmită informații prin orice fel de variație a unei cantități fizice. Sistemele de comunicare folosesc formate bine definite pentru schimbul de mesaje diverse. Fiecare mesaj are o semnificație exactă menită să obțină un răspuns dintr-o serie de răspunsuri posibile predeterminate pentru acea situație particulară.[5]

Protocolul AXI4-Stream reprezintă o interfață standard în domeniul conectării componentelor care doresc să schimbe date. Interfața poate fi utilizată pentru a conecta un singur master, care generează date, la un singur slave, care primește date. Protocolul poate fi, de asemenea, folosit atunci când se conectează un număr mai mare de componente master și slave. Acest protocol suportă mai multe fluxuri de date utilizând același set de linii partajate, permițând construirea unui interconector generic care poate realiza operații de mărire, micșorare și rutare. Protocolul funcționează eficient pe baza unui mecanism de colaborare ready/valid, facilitând schimbul coordonat de date între expeditor și receptor. Cu suport pentru diverse tipuri de fluxuri, AXI4-Stream definește asocierile între transferuri și pachete, ceea ce contribuie la realizarea unor sisteme complexe și eficiente în manipularea datelor. [6]

Semnale interfetei:

- ACLK- este semnalul global de ceas, iar toate semnalele sunt esantionate la frontal ascendent al semnalului ACLK. Acesta provide de la sursa clockului
- ARESET- este semnalul global de reset si este active cand este in stare LOW. Acesta provide de la sursa resetului
- TDATA-toate datele sunt transmise prin interfata. Acesta provide de la sursa trimitatorului
- TVALID- indica faptul ca expeditorul furnizeaza date valide Acesta provide de la sursa trimitatorului
- TREADY- indica faptul ca receptorul poate accepta datele in ciclul current. Acesta provide de la sursa primitorului

Protocolul de transfer de date hardware "ready/valid" este simplu și ingenios, oferind controlul fluxului cu doar două semnale de control. Regulile sunt clare: transferul de date se întâmplă numai atunci când atât semnalul TREADY, cât și TVALID sunt '1' în aceeași perioadă de ceas.[7]

Semnalele ready/valid sunt direcționate către semnalele cu nume identic atunci când se conectează modulele expeditorului și receptorului. Acest lucru face mai ușor navigarea în proiectul VHDL.

Semnalele care conectează două module vor avea aceleași nume ca semnalele entității la care se conectează, indiferent dacă privim din perspectiva expeditorului sau a receptorului. Beneficiul acestui aspect poate fi observat în diagrama de mai jos, care arată două module interconectate cu interfețe identice:

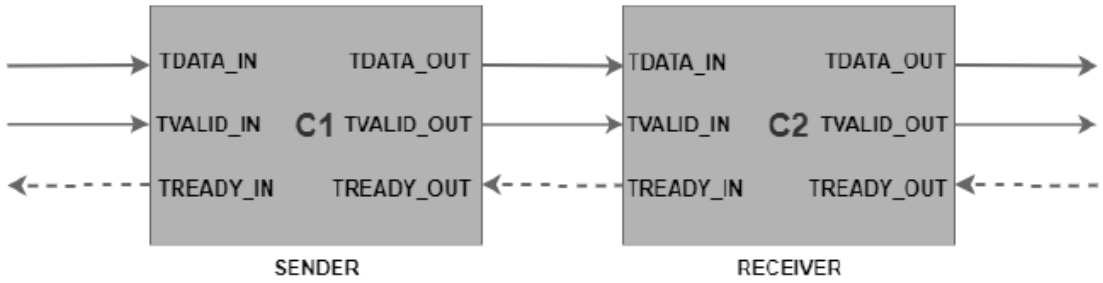


Figura 2: Interconectarea a doua componente

Reguli care trebuie respectate de componente care comunică printr-o interfață AXI4-Stream:

1. Pentru ca o transferare de date să aibă loc, atât semnalele TVALID, cât și TREADY trebuie să fie HIGH în aceeași perioadă de ceas.
2. Un master nu are voie să aștepte până când TREADY este setat la 1 înainte de a fi si TVALID setat la 1. Odată ce TVALID este setat la 1, trebuie să rămână setat la 1 până când are loc handshake-ul.
3. Un slave are voie să aștepte ca TVALID să fie setat la 1 înainte de a fi setat la 1 corespunzătorul TREADY.
4. Dacă un slave afirmă TREADY, are voie să setat la 0 TREADY înainte ca TVALID să fie setat la 1.[8]

3. Analiza

3.1 Analiza semnalelor de intrare

Ca semnale de intrare am ales sa folosesc pixelii. Acestia sunt elementele componente, de dimensiuni foarte mici, ale imaginilor grafice digitale. Imaginile, pentru a putea fi prelucrate, trebuie mai intai sa fie digitalizate, adica impartite in elemente minuscule, incat fiecare element sa aiba o singura culoare clar dominanta. Atunci fiecare element, numit pixel, poseda trei atribute care se pot exprima numeric, acestea fiind: culoare, opacitate si pozitie in matricea in care se divide imaginea.Definirea atributelor se poate face pe 1, 2, 4, 8, 16 sau și mai mulți biți pentru fiecare plan de culoare.[9]

3.2 Use-Case

Circuitul de filtrare a imaginilor ofera diferse functionalitati utile prin intermediul filtrelor alese. Un prim scenariu de utilizare implica conversia imaginilor din format RGB in tonuri de gri, oferind utilizatorului posibilitatea de a obtine efecte artistice utile in editarea imaginiilor. Al doilea scenario pe concentreaza pe adjustarea luminozitatii si a contrastului, permitand editorilor de fotografii sa inbunatateasca vizibilitatea detaliilor intr-o imagine. Iar ca ultim scenariu, inversarea tabelei de aspect, ofera posibilitatea inversarii culoriilor pentru a creea efecte vizuale unice, care pot ajuta la scoaterea in evidenta a detaliilor imaginilor.

Pentru a se bucura de toate aceste facilitatii, utilizatorul trebuie doar sa porneasca placa Nexys4 si sa o conecteze la o sursa de alimentare. Trebuie sa se asigure ca datele sunt introduse corect in fisierul din care se vor citi datele si ca sunt valide pentru citire. Dupa se poate genera bitstream-ul si configura fisierul de configurare. Dupa programarea cu success a dispozitivului, se poate incepe rularea si testarea proiectului pe placuta. Fiecare filtru lucreaza independent si si face treaba, dar in acelasi timp toate ruleaza paralel. Utilizatorul poate alege ce filtru sa se afiseze pe display-ul placutei, in functie de comutatoare. Astfel le poate vizualiza pe fiecare in parte, doar prin schimbarea comutatorului.

3.3 Filtrele alese

Am ales trei filtre care sa fie implementate in circuitul circuit de filtrare a semnalelor digitale pentru a avea rezultate optime:

1.
    a) Filtrul ce inverseaza tabele de aspect(LUT) Filtrul ce ajusteaza luminozitatea/contrastul imaginilor

       newPixelValue =255- currentPixelValue

Pentru imagini de 8 biți, valoarea fiecărui pixel este înlocuită cu 255-v. Cu LUT-urile inverse, pixelii cu valoarea zero sunt albi, iar cei cu valoarea 255 sunt negri. Valorile pixelilor existenti nu sunt modificate, ci doar modul în care imaginea este afișată pe ecran.

b) Filtrul ce ajusteaza luminozitatea/contrastul imaginilor

newPixelValue=currentPixelValue+brightness

Acest filtru este folosit pentru a modifica interactiv luminozitatea și contrastul imaginii active. În cazul imaginilor de 8 biți, luminozitatea și contrastul sunt schimbate prin actualizarea tabelului de aspect (LUT) al imaginii, astfel încât valorile pixelilor să rămână neschimbate. [10]

2. Filtrul care converteste imaginile RGB in tonuri de gri

   grayScalePixel=(r+g+b+a)/4

   Unde r este notatia pentru rosu(red), g este pentru verde(green), b este pentru albastru(blue), iar a este notatia pentru alfa. Procesul de conversie a imaginilor RGB în tonuri de gri implică transformarea unei imagini color compuse din trei canale de culoare (roșu, verde și albastru) într-o imagine în care fiecare pixel este reprezentat de o singură valoare de intensitate luminoasă.

3. Filtrul ce multiplica doi pixeli pentru a ajuta la inmultirea a doua imagini

   newPixelValue2=pixel1*pixel2

   Acest filtru este folosit pentru inmultirea a doua imagini, inmultind un pixel din prima imagine cu alt pixel din a doua imagine.[13]

4. Design

4.1 Componente

Vom detalia algoritmul folosit peentru fiecare filtru:

1.Pentru filtrul ce inverseaza tabele de aspect(LUT) vom avea nevoie de un scazator, dar putem folosi un sumator cu propagarea succesiva a transportului deoarece suporta si operatia de scadere.

Sumatorul cu propagare succesivă a transportului poate fi utilizat și pentru operații de scădere, deoarece scăderea a două numere binare se poate realiza prin adunarea complementului față de doi (C2) al scăzutului la descăzut. De exemplu, având două numere binare X = 01001 și Y = 00011, scăderea X - Y poate fi efectuată prin adunarea numărului X la complementul față de doi al numărului Y:

```
 01001 +          x
 11101            y(c2)
-----------
 00110            x-y
```

Transportul de la poziția cea mai semnificativă a rezultatului este neglijat. Același principiu se aplică și în cazul operației de scădere, unde fiecare bit al scăzătorului este scăzut din bitul corespunzător al descăzutului pentru a forma un bit de diferență. Dacă bitul descăzutului este 0 și bitul scăzătorului este 1, se ia împrumut 1 de la următoarea poziție mai semnificativă.

2.Pentru filtrul care converteste imaginile RGB in tonuri de gri vom avea nevoie atat de un sumator cu salvarea transportului extins pentru 4 numere, cat si de doua siftari.

Pentru acest filtru ne vom folosi de algoritmul “Carry save adder” care aduna de regula cate 3 numere de cate n biti. Fiecare sumator elementar genereaza cate un bit suma si un bit de transport, care formeaza un cuvant suma S de n biti, respectiv un cuvânt de transport T de n biti.

Fiecare sumator elementar funtioneaza independent unul de celalalt, iar semnalele de transport nu sunt propagate intre sumatoarele elementare. Astfel, viteza de adunare este mai mare, deoarece toti bitii suma si de transport pot fi generati in paralel. Pentru obtinerea rezultatului, suma si transportul trebuie adunate cu un sumator, numit sumator cu propagarea transportului(SPT).

Pentru realizarea mediei, o sa avem nevoie de un algoritm ce aduna cate 4 numere sau cel putin multiplu de doi, deoarece pentru a realiza media, vom avea nevoie sa shiftam la dreapta cu un multiplu de doi pentru realizarea impartirii. Procedura implică divizarea

operației de adunare în două secțiuni: una pentru cifrele de ordin înalt și alta pentru cele de ordin scăzut.

Astfel pentru adunarea a patru numere, sumatorul cu salvarea de transport genereaza mai intai o suma a primelor trei numere si un transport salvat. Apoi se aduna suma, al patrulea numar si transportul salvat, generand o noua suma si un nou transport salvat. Iar in cele din urma, se utilizeaza un sumator cu anticiparea transportului pentru a aduna noua suma si noul transport salvat, rezultand astfel rezultatul final.

Acest circuit de care avem nevoie va fi alcatuit din doua sumatoare cu salvarea transportului, fiecare fiind format dintr-o serie de sumatoare elementare. In ultima etapa, se utilzeaza un sumator cu anticiparea transportului pentru obitnerea sumei finale, prezentat in urmatoarea imagine:

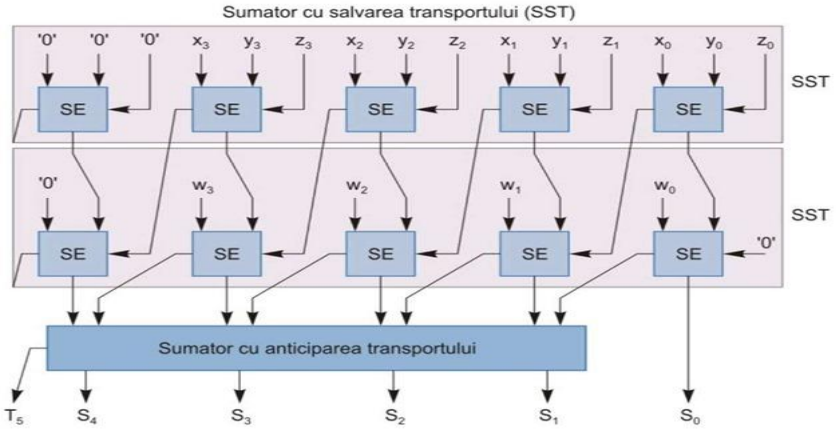


Figura 3 : Sumator cu Salvarea transportului [11]

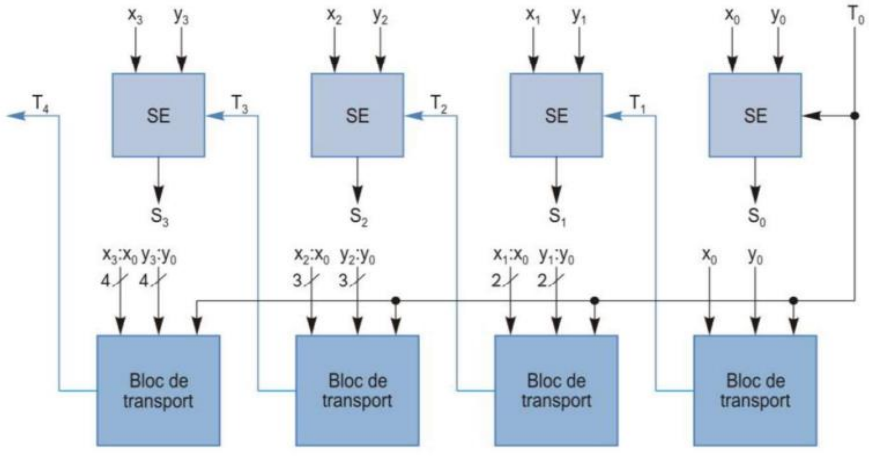


Figura 4.2. Schema bloc a unui sumator de patru biți cu anticiparea transportului.

Figura 4 : Sumator cu anticiparea transportului [11]

3. Pentru filtrul ce multiplica doi pixeli pentru a ajuta la inmultirea a doua imagini vom avea nevoie o inmultire matriciala ce se bazeaza pe un sumator cu propagarea succesiva a transportului.

Algoritmul de adunare cu propagare succesivă a transportului reprezintă unul dintre algoritmii de bază pentru operațiile de adunare, implementat simplu cu sumatoare elementare. Procesul este similar cu adunarea obișnuită.

Implementarea acestui proces implică conectarea mai multor sumatoare elementare în serie, câte un sumator pentru fiecare bit al numerelor care trebuie adunate. Ieșirea de transport a unui sumator elementar este conectată la intrarea de transport a următorului sumator elementar. Intrarea de transport a sumatorului din poziția cea mai puțin semnificativă este setată la valoarea logică 0. Prin urmare, pentru această poziție, se poate utiliza un semisumator elementar.

Sumatorul este un tip de sumator paralel, deoarece operanzii sunt aplicați la intrare în același timp. Numele acestui sumator provine de la faptul că transportul trebuie să se propage succesiv prin toate sumatoarele elementare înainte de a se cunoaște rezultatul final. Deși acest tip de sumator este simplu de realizat și are un cost redus, este unul dintre cele mai lente sumatoare. Acest lucru se datorează faptului că transportul trebuie să se propage de la poziția bitului cel mai puțin semnificativ la poziția bitului cel mai semnificativ. Este adesea folosit ca o celulă de adunare pentru construirea sumatoarelor de dimensiuni mai mari. [11]

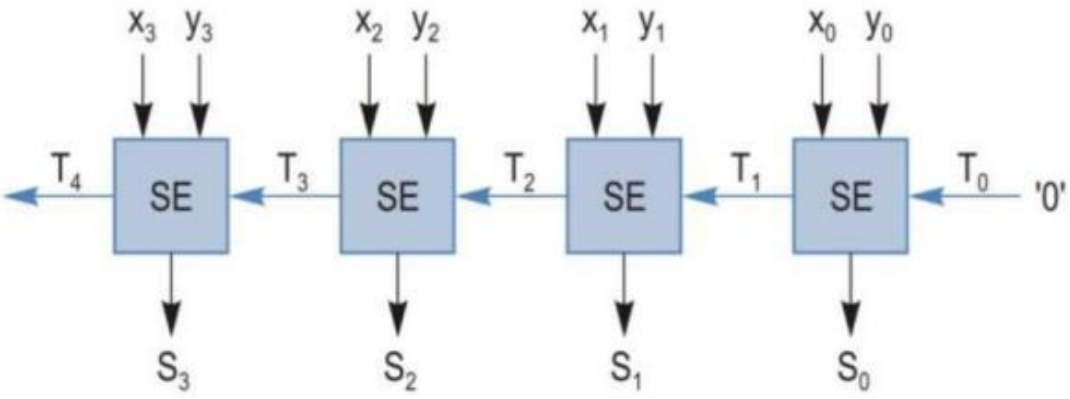


Figura 4: Schema bloc a unui sumator cu propagarea succesiva a transportului [11]

Un circuit de inmultire este format dintr-o matrice de elmente simple, fiecare din acestea implementand o operatie de adunare si deplasare pentru un bit sau pentru un numar redus de biti.

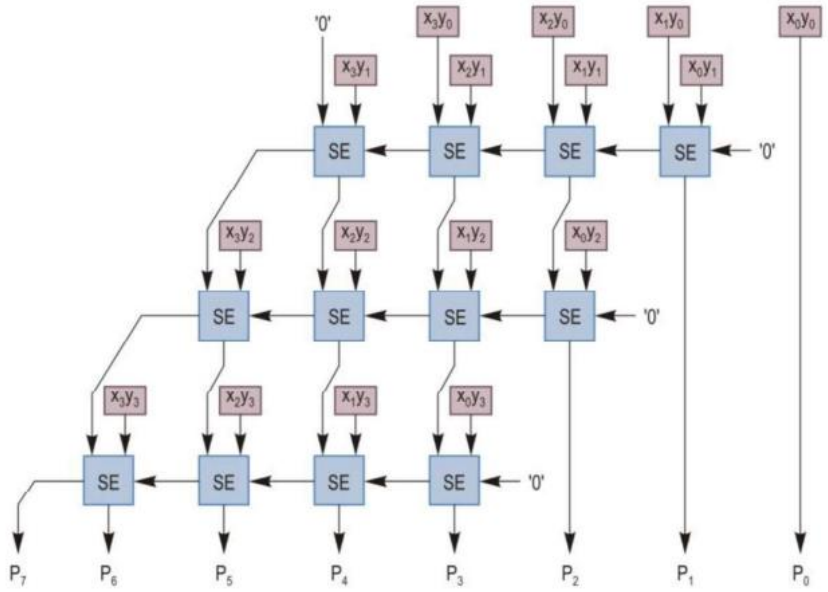


Figura 4.8. Matrice de sumatoare elementare pentru înmulțirea a două numere fără semn de câte patru biți.

Figura 4: Matrice de sumatoare elementare pentru inmultirea adoua numere fara semn de cate patru biti [11]

				x_3	x_2	x_1	x_0 *
				y_3	y_2	y_1	y_0
0	0	0	0	x_3*y_0	x_2*y_0	x_1*y_0	x_0*y_0
0	0	0	x_3*y_1	x_2*y_1	x_1*y_1	x_0*y_1	0
0	0	x_3*y_2	x_2*y_2	x_1*y_2	x_0*y_2	0	0
0	x_3*y_3	x_2*y_3	x_1*y_3	x_0*y_3	0	0	0
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

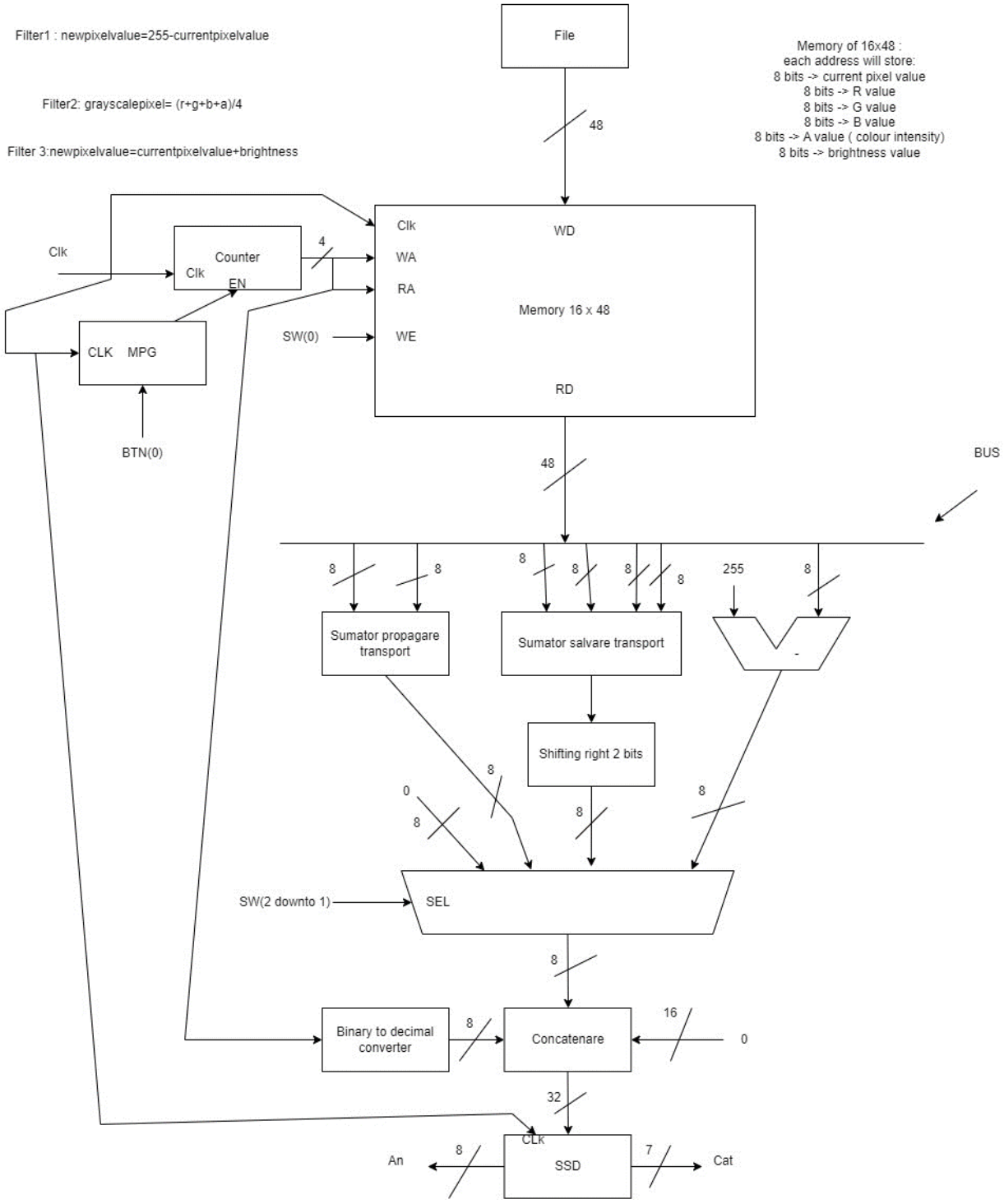
Biții produsului final au expresiile booleene din ecuațiile (4.22) – (4.28), în care produsele aritmetice x_i*y_j sunt înlocuite cu produsele logice $x_i \cdot y_j$.

$$P_0 = x_0 \cdot y_0 \quad (4.22)$$

$$P_1 = x_1 \cdot y_0 + x_0 \cdot y_1 \quad (4.23)$$

$$P_2 = x_2 \cdot y_0 + x_1 \cdot y_1 + x_0 \cdot y_2 \quad (4.24)$$

4.2 Schema bloc



În această schemă complexă, se oferă o privire de ansamblu asupra tuturor componentelor proiectului, evidențiind modul în care acestea interacționează pentru a realiza funcționalitățile dorite. Principalele etape ale schematizării includ citirea datelor dintr-un fișier folosind protocolul de comunicare AXI4-Stream, gestionarea acestor date și aplicarea unor filtre, pentru ca în final, rezultatul să fie afișat pe un display.

In prin plan vom citi datele dintr-un fisier folosindu-ne de protocolul de comunicare AXI4-Stream, cu ajutorul caruia vom citi datele de 48 de biti si le vom salva intr-o memorie. In memoria de 16X48, fiecare adresa va stoca, 8 biti pentru valoarea curenta a pixelului, 8 biti pentru valoare R, 8 biti pentru valoare G, 8 biti pentru valoare G, 8 biti pentru valoare alfa a intensitatii culorii si 8 biti pentru valoare luninozitatii.

Pe latura stângă a schemei se află un debouncer pentru butoane (denumit și MPG), menit să elimine fluctuațiile generale apărute la apăsarea butoanelor. La apăsarea butonului 0, un contor începe să numere, stabilind astfel adresa de citire/scriere pentru accesul la memorie.

ona de interfațare dintre memoria de stocare și procesarea datelor este gestionată cu atenție. Cu ajutorul comutatorului 0, datele pot fi citite de la adresa numerotată sau pot fi scrise în magistrala bus. Aici, grupurile de 8 biți corespunzătoare fiecărui filtru sunt transmise pentru a iniția procesul de filtrare.

Filtrele sunt esențiale pentru prelucrarea imaginilor . In filtrul 1 ce inverseaza tabele de aspect(LUT) folosim un scazator ce va putea fi implementat folsindu-ne de un sumator cu propagare de transport, deoarece acesta suporta si operatii de scadere, in filtrul 2 ce converteste imaginile RGB in tonuri de gri folosim atat sumatorul cu salvare a transportului, cat si siftarea la dreapta cu 2 biti, iar la filtrul 3 ce ajusteaza luminozitatea/contrastul imaginilor folosim un sumator de propagare a transportului. Toate cele trei filtre rulează în paralel, aducând complexitate și eficiență procesului.

În funcție de comutatorul ales, rezultatele filtrului selectat sunt trimise către o etapă ulterioară de concatenare. Aici, rezultatul ales este concatenat cu conversia valorii din binar în decimal, pregătindu-l pentru afișare.

In ultimul nivel al schemei, rezultatul final este transmis la un afișaj cu șapte segmente (ssd) și, în cele din urmă, este afișat pe anozii și catozii afișorului plăcii Nexys 4. Prin intermediul acestei arhitecturi complexe, proiectul realizează procesarea datelor, aplicarea filtrelor și afișarea rezultatelor într-un mod eficient.

5. Implementare

Am inceput acest proiect construind fiecare filtru in parte, dupa testandu-le.

.Prima data am implementat algoritmii de adunare, respective inmultire. Prima sursa implementata a fost algoritmul ‘sumator cu propagarea succesiva a transportului’ (SE), unde sumatorul aduna doi biti ‘x’ si ‘y’ impreuna cu bitul de transport ‘tin’ si genereaza rezultatul ‘s’ si un bit de transport de iesire’tout’. Acest sumator se bazeaza pe operatii logice de XOR si AND.

Al doilea algoritm este sumatorul cu anticiparea transportului pentru numere pe 8 biti. Acest sumator folosește semnalele x și y ca intrări pentru numerele de adunat, semnalul s ca ieșire pentru rezultatul adunării, semnalul tin pentru intrarea transportului, și semnalul tout pentru ieșirea transportului. În plus, entitatea utilizează semnale auxiliare interne, cum ar fi g, p, și t, pentru calculul intern al sumelor și transporturilor, unde rezultatul operatiei logice “and” intre biții corespunzători din x și y, o constituie g, rezultatul operatiei logice “or” intre biții corespunzători din x și y, o constituie p, iar transportul intermediar calculate din bitul anterior bazat pe g,p si t in constituie t. Este utilizata o instanta de componenta numita SE pentru a realiza adunarea individuala pe fiecare bit.

Al treilea algoritm implementat a fost sumatorul cu salvarea transportului(SST), care se foloseste de SE pentru a realiza adunarea la nivelul bitului și combina rezultatele pentru a obține un sumator ce poate aduna numere pe 8 biti. Pentru adunarea a 4 numere a fost nevoie sa schimbam SST normal ce aduna doar 3 numere, pe unul ce poate aduna 4, folosindu-se de adunarea pe 2 nivele. Asa ca am declarant semnale de transport si de sume pentru fiecare nivel. La primul nivel, pentru fiecare bit de intrare de la x(0) până la x(7), am utilizat instanțe ale componentei ‘SE’ pentru a realiza adunarea și pentru a obține semnalele de transport și sumă la nivelul 1, pentru primele 3 numere. La nivelul doi, am folosit instanțe ale componentei ‘SE’ pentru a realiza adunarea între al patrulea numar și sumele obținute la primul nivel. Apoi instantiez componenta ‘SAT4_8’ pentru a realiza adunarea finala, avand ca intrari semnalele de suma de la nivelul 2 si transportul de la nivelul 2, iar suma finala este primita de la “SAT” si se concateneaza cu primul bit de la nivelul 2.

Dupa ce am implementat algoritmii am trecut la filtre, unde primul filtru implementat a fost filtrul1, unde implementam formulele:

newPixelValue =255- currentPixelValue, daca selectia e pe ‘0’ si formula newPixelValue=currentPixelValue+brightness, daca selectia e pe ‘1’.

In arhitectura am declarat un semnal numit valoare și este inițializat cu valoarea x"FF" in hexadecimal, echivalentă cu 255 în decimal. Dupa care am create un mux, unde daca selectia e ‘0’ atunci newPixelValue este calculat negand valoarea lui currPixelValue, adaugand 1 si

adaugand valoarea 255, iar daca selectia e ‘1’ atunci pur si simplu adunam brightness si currentPixelValue.

A doua sursa implementata a fost filtru2 unde am facut media aritmetica a 4 valori ale pixelilor. In arhitectura am declarant 2 semnale, unul in care sa retinem suma(retinem) si unul in care salvam valoarea shiftata(shiftam). Apoi instantiem componenta SST4_8, care ne va face adunarea celor 4 numere si se va salva in retinem, dupa care shiftam cu doua pozitii la dreapta rezultatul, cee ace inseamna impartirea la 4 (o shiftare la dreapta este egala cu impartirea la 2), iar in cele din urma salvam rezultatul final in variabila grayscalePixel.

A treilea filtru implementat a fost filtrul3 unde am inmultit doua valori de pixeli folosindu-ne de inmultirea matriciala. In arhitectura am declarant ca prim semnal important, un array de vector de 8 biti, utilizat pentru a stoca produsele partiale, numit ‘produse’, dupa care declaram matricea produse_partiale, care este un tablou de 8 elemente, fiecare fiind un vector cu lungimea de 8 biti. Apoi am declarant semnale pentru sumele intermediare, o suma pentru fiecare linie, semnale care salveaza transportul, dupa care am declarant semnalul produs de 16 biti care va salva rezultatul final. Initial am calculat produsele partiale pentru fiecare element al matricei de intrare, dupa care am instantiat componenta SE care a calculat sumele intermediare pentru fiecare linie, adica adunam fiecare element de pe prima linie la elemental corespunzator de la a doua linie si tot asa. Apoi am concatenate toate sumele, impreuna cu primul produs partial si cu ultimul transport pentru a forma produsul final, atribuit semnalului produs, dupa care am atribuit iesirii newPixelValue ultimii 8 biti ai semnalului produs.

Pentru testarea proiectului a fost nevoie de citirea datelor dintr-un fisier pentru a confirma corectitudinea functionarii acestuia.Simularea ce aplica asupra unui fisier de intrare, unde se citesc datele intr-o fifo(coada). Se initializeaza semnale pentru ceas, pentru reset, semnale de date(t1_data), semnale de validare. Se utilizeaza o memorie de 64 pe 64 pentru stocarea datelor provenite de la fifo, iar daca datele sunt valide atunci putem citi sau scrie in memorie, dupa care daca am terminat citirea din memorie, atunci mutam switch-ul inapoi la 0. Dupa care se incarca in semnalele specific filtrelor, din RD. Apoi se instantiaza modulul fifo, unde se specifică semnalele de control s_axis_ pentru scriere și m_axis_ pentru citire și semnalele de date _tdata, se deschide un fișier text în modul de citire cu numele "filtre.csv",daca resetul este activ și nu s-a terminat citirea (end_of_reading = '0'), procesul continuă. Dacă nu s-a ajuns la finalul fișierului și modulul fifo este pregătit să accepte date noi , procesul citeste o linie din fișier. Se setează semnalele corespunzătoare modulului FIFO (t1_tvalid, t1_tdata) cu valorile citite, dupa care incrementă rd_count și se afișează un raport în consolă despre datele citite, iar dacă se termină fișierul, fișierul este închis și semnalul end_of_reading este setat la '1'.

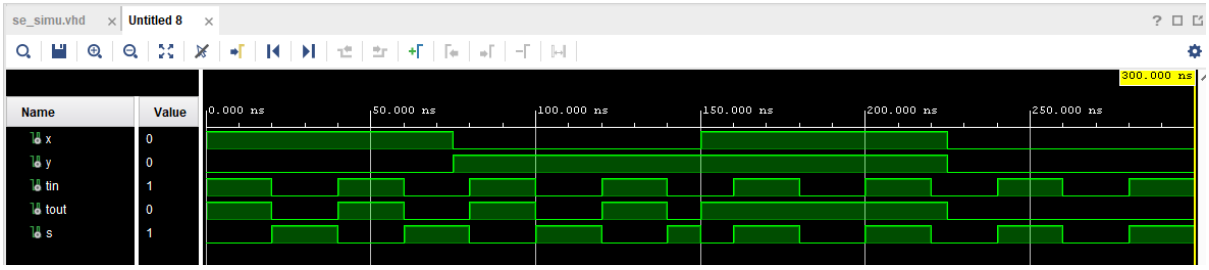
Sursa displ7seg implementeaza un afisaj cu 7 segmente pentru a afisa date pe ssd-uri. Dispozitivul are 8 cifre și poate afișa datele primite pe intrarea Data pe cele 8 cifre, utilizând un divizor de frecvență de 100 Hz. Se implementeaza un process de divizare a ceasului ce se executa la fiecare ciclu de ceas si utilizează un contor 'Num' pentru a ține evidența câtor cicluri de ceas s-au înregistrat. Scopul principal al acestui proces este de a obține un semnal care să determine selecția cifrei active pe afișaj. Apoi se selecteaza anodul activ in functie de valoarea lui "LedSel", dupa care se selecteaza cifra active, tot in functie de valoarea "LedSel", iar in final se activeaza/dezactiveaza segmentele cifrei active, in functie de valoarea "Hex".

Sursa mpg1 este denumita si debouncer ce elimina sau reduce fluctuațiile care pot apărea la apasarea butoanelor. In procesul clk, la fiecare front crescator, Q1 este actualizat cu valoarea butonului, iar in al doilea process, Q2 si Q3 sunt actualizate cu valorile lui Q1 si Q2 curente. Semnalul de ieșire en este definit ca Q2 AND (not Q3) si indica un nivel stabil al butonului btn.

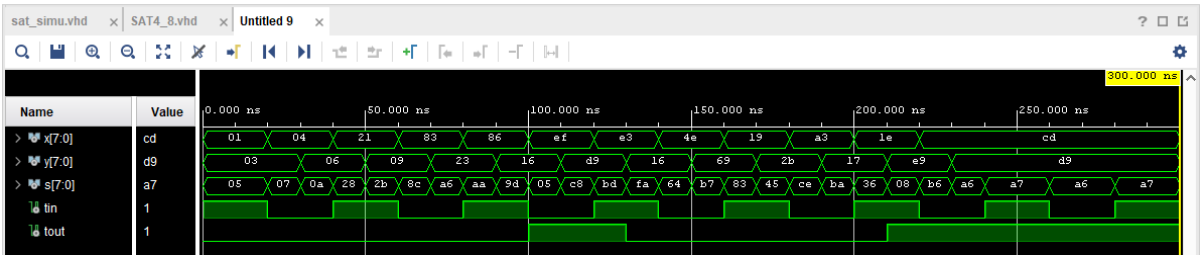
Iar in cele din urma designul principal numit nexys4 unde se face legarea cu placuta propriu zisa. Aici am initializat o memorie cu 16 adrese, unde fiecare e un vector de 64 de biti. Se poate scrie in memorie atunci cand sw(0) este '0', si se poate citi din memorie cand este 1. Eu am harcodat memoria cu niste valori propria, asa ca nu voi scrie in memorie. Instantiez mpg-ul pentru a avea un debouncer pentru butoane, dupa instantiez cele 3 filtre si afisorul. Primul process implementeaza un contor care se incrementeaza cu fiecare ciclu de ceas, care ma ajuta sa adresez locatiile din memorie. Iar in ultimul proces imi aleg ce filtru vreau sa afisez pe ssd-uri in functie de butoane. Daca pe switch-uri am valoarea "00" atunci afisez valoarea de pe primul filtru, daca am "01" atunci afisez valoarea de pe filtrul 2, iar daca am "10" atunci afisez valoarea de pe filtrul 3.

6. Testare

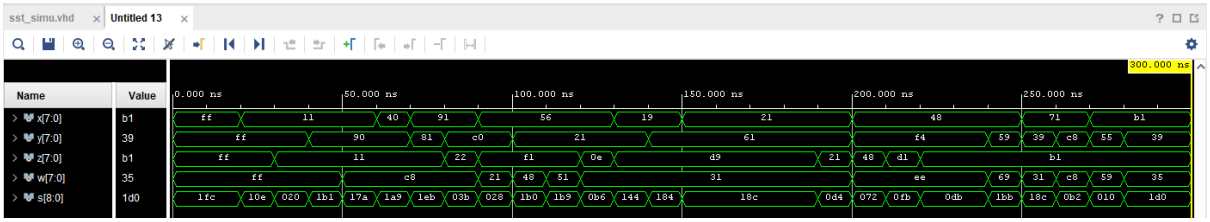
Simularea sumatorului cu propagare succesiva, unde adunam bit(x) cu bit(y) si cu transportului de intrare(tin) si afisam suma(s) si transportul(tout): (rezultate corecte)



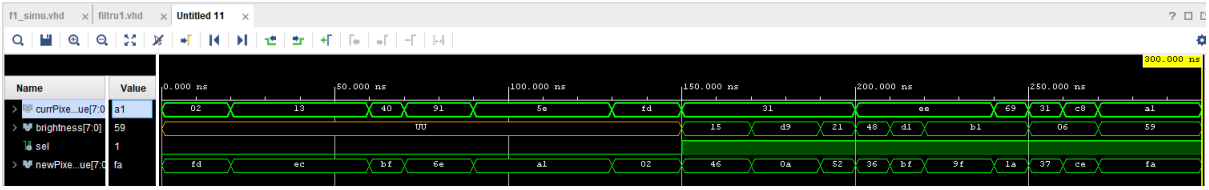
Simularea sumatorului cu transport anticipat, unde se face adunarea a doua numere de cate 8 biti fiecare: (rezultate corecte)



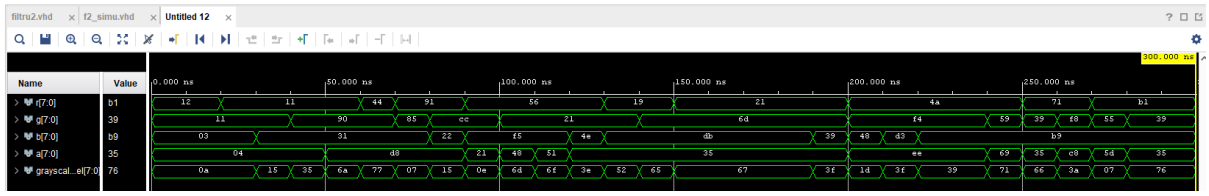
Simularea sumatorului cu salvarea transportului, unde adunam 4 numere pe 8 biti fiecare: (la adunarea FF+FF+FF+FF rezultatul correct este 3FC, iar rezultatul obtinut este 1FC, in rest rezultatele sunt correct)



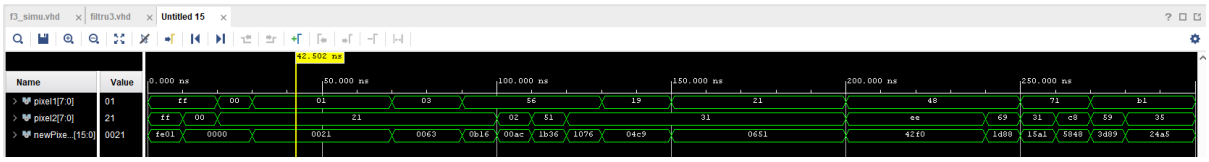
Simularea filtrului 1, unde daca avem selectia 0 facem 255- currPixel, iar daca este 1, atunci facem operatia currPixel+brightness: (rezultate corecte)



Simularea filtrului 2, unde facem media aritmetica a 4 numere: (excepantand cazul unde nu realizeaza correct adunarea FF+FF+FF+FF, restul rezultatelor sunt corecte)

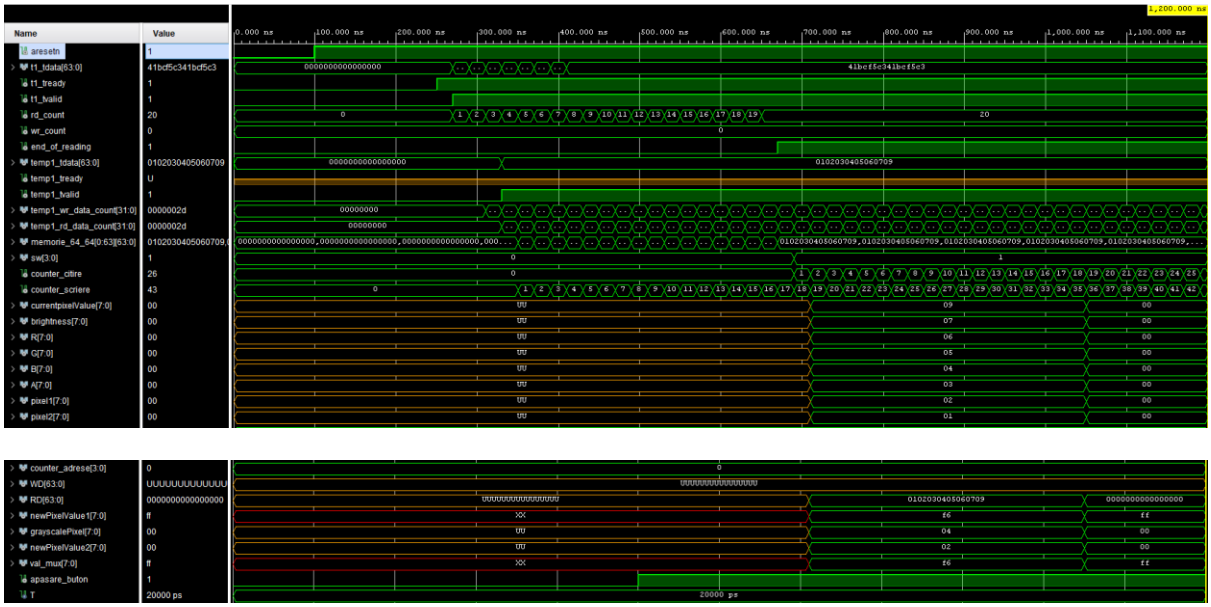


Simularea filtrului 3, unde facem inmultirea a doua numere fiecare pe 8 biti: (rezultate corecte)



Simularea finala a proiectului unde se face citirea din fisier

- Daca citirea din fisier nu e gata, iar restul este setat pe 1 si daca n am terminat de citit fisierul, atunci procesul citeste date din fisier si le furnizeaza modului fifo, atunci cand este gata
- Daca citirea a avut success, atunci valorile sunt incarcate in fifo
- In primul process, se gestioneaza scrierea din fifo in memories au citire in functie de switch-uri, unde daca datele sunt valide sis w='0' atunci datele pot fi citite din memorie
- Cand citirea din fisier se termina, sw devine 1



Pentru afisare pe placuta am observant urmatoarele cazuri pentru fiecare filtru in parte:

- La filtrul 1 datele au fost corecte

Filtru 1	Date asteptate	Date primite
	FF - 9 = F6 sau 9+6=F	F6 sau F
	FF -ED=12 sau ED+E7=1D4	12 sau d4
	FF-29=D6 sau 29+65=8E	D6 sau 8E

- La filtrul doi la valori mari greseste, dar in rest e bine

Filtrul 2	Date asteptate	Date primite
	(3+12+5+6)/4=6	6
	(E3+AC+E5+BG)/4=CA	4A
	(25+4C+16+13)/4=26	26

- La filtrul 3 datele sunt corecte, doar ca afisarea mea e doar pentru 2 ssd-uri si de aceea vad doar cifra zecimalelor si unitatilor corecte

Filtrul 3	Date asteptate	Date primite
	3*2=6	6
	FB*FA=F5IE	1E
	2B*3=81	81

7. Concluzii

În cadrul acestei documentații, am explorat implementarea procesării filtrelor în limbajul VHDL pentru a obține rezultate de prelucrare a pixelilor. Proiectul include trei filtre principale , fiecare dintre ele oferind rezultate conform așteptarilor, conform simulărilor efectuate.

Acest proiect a fost de o dificultate de mediu spre mare, deoarece cel mai greu lucru de implementat a fost citirea si scrierea din fisier. În ciuda acestor obstacole, am reușit să realizăm citirea primului rând din fișier, dar procesul de scriere nu a putut fi realizat. De asemenea, la cazuri mari sumatorul cu salvarea transportului intampina ceva erori, cee ace afecteaza si media aritmetica a celor 4 numere, deoarece calculeaza inexact adunarea, dar restul algoritmilor functioneaza bine.

Ca eventuale inbunatati, ar trebuie sa rezolv scrierea si afisarea in fisier, sa fac ca sumatorul cu salvare transportului sa suporte si numere mari, sa realizez afisarea pe maim ult de 2 ssd-uri pentru a vedea exact numerele corecte, nu doar ultimele cifre din ele si inbunatatirea formulelor.

Bibliografie

[1]. Autor: Paul Irofti

Titlu:Procesarea semnalelor-Filtre FIR

Link: https://cs.unibuc.ro/~pirofti/ps/ps-curs-6.pdf

[2] link: https://digilent.com/reference/programmable-logic/nexys-4/start

[3]. autor:Alex Serbanescu

titlu:Carte noua DSP

link: http://alexserbanescu.ro/wp-content/uploads/2013/10/Carte-noua-DSP.pdf
an:2013

[4].autor: KERTÉSZ Csaba-Zoltán si Laurențiu-Mihail IVANOVICI

Titlu: PROCESAREA DIGITALĂ A SEMNALELOR

Link: https://www.miv.ro/books/CKertesz_MIvanovici_PDS.pdf

[5]. Autor: wiki

Titlu:Communication_protocols

link: https://hmn.wiki/ro/Communication_protocols

[6]. Titlu: AMBA 4 AXI4-Stream Protocol
Link: https://
documentation-service.arm.com/static/642583d7314e245d086bc8c9?token=

an: 2010

[7].autor: Jonas Julian Jensen

Titlu: How the AXI-style ready/valid handshake works

link: How the AXI-style ready/valid handshake works - VHDLwhiz

an: 2022

[8]. autor: AXI4-Stream Communication Protocol. Introduction

titlu: L6-Design of ALU components.pdf

[9]. Autor: Wikipedia

Titlu: Pixel

Link: https://ro.wikipedia.org/wiki/Pixel

[10]. Autor: ImageJ Wiki

Titlu:Image menu

Link: https://imagej.net/ij/docs/menus/image.html

[11]. autor: Zoltan Baruch

titlu:Arith Combinationala

[13] Autor: A.Walker and E.Wolfart

Titlu: Pixel Multiplication and Scaling

Link: Image Arithmetic - Pixel Multiplication and Scaling (ed.ac.uk)

An:2003

link: https://users.utcluj.ro/~baruch/ssc/labor/Aritm-Combinationala.pdf

[12].link-ul unde am realizat diagrama: https://app.diagrams.net/