COMPETIŢIA DE PROIECTE DE CERCETARE A ACADEMIEI OAMENILOR DE ŞTIINŢĂ DIN ROMÂNIA "AOŞR-TEAMS-IV" EDIŢIA 2025-2026

O nouă aplicație hardware eficientă pentru compensarea ecoului acustic în sisteme de teleconferință stereo folosind o schemă de descompunere inovatoare

ETAPA 2 – RAPORT DE CERCETARE ŞTIINŢIFICĂ

Rezumatul etapei

Obiectivul specific corespunzător celei de a doua etape intermediare a proiectului (desfăşurată în perioada august 2025 – noiembrie 2025) este:
O2. Proiectare/validare implementare HDL în virgulă fixă pentru algoritmul de tip RLS-DCD-WL și realizarea sintezei pentru dispozitivul FPGA țintă.

În cadrul acestei etape de cercetare au fost îndeplinite toate activitățile aferente lunilor de desfășurare, asociate acestui obiectiv specific, prevăzute în planul de realizare a proiectului, după cum urmează:
1. Proiectare simulator Matlab în precizie cu virgulă fixă pentru a fi folosit ca referință pentru urmărirea rezultatelor/acurateței implementării VHDL în precizie cu virgulă fixă.
2. Proiectare blocuri funcționale ale algoritmului de tip RLS-DCD-WL folosind VHDL pentru dispozitiv FPGA țintă.
3. Publicare articol într-o revistă ISI Q1/Q2.

Rezultatele din cadrul acestei etape de cercetare au fost publicate în lucrările [1, 2] din lista de referințe. Articolul [2] este acceptat în revista *IEEE Signal Processing Letters* (factor de impact 3,9), cotată ISI-WOS Q1 după scorul relativ de influență și ISI-WOS Q2 după factorul de impact.

1. Introducere

În aplicațiile folosite pentru compensarea ecoului acustic (teleconferințe, dispozitive „cu mâini libere" - *hands-free*, aparate auditive, etc.) [3]-[8], este necesară estimarea/identificarea unuia sau mai multor răspunsuri la impuls acustice. O problemă importantă care trebuie abordată atunci când avem de-a face cu astfel de scenarii sunt *cuplările* formate între difuzoare și microfoane. Prin urmare, este nevoie de îmbunătățirea calității semnalelor microfonului/microfoanelor prin compensarea ecourilor acustice nedorite. Cea mai fiabilă soluție la această problemă este utilizarea *filtrelor adaptive* [9]-[12] care generează la ieșirile corespunzătoare replici ale ecourilor, care sunt apoi scăzute din semnalele de microfon. Filtrele trebuie să modeleze sisteme necunoscute (căile de ecou acustic) și funcționează în cadrul unei configurații denumite *identificarea sistemului*. Acesta este cazul compensării ecoului acustic stereofonic (*stereophonic acoustic echo cancellation* - SAEC) [8], [13]. În sistemele de teleconferință *hands-free*, transmisia stereo asigură impresia prezenței fizice a interlocutorului datorită sistemului auditiv binaural uman. Aceste sisteme stereofonice oferă o prezență realistă pe care sistemele cu un singur canal nu o pot oferi. Configurația SAEC este prezentată în Figura 1, unde $x_{\mathrm{L_c}}(n)$ și $x_{\mathrm{R_c}}(n)$ sunt semnalele difuzorului, respectiv $y_{\mathrm{L_c}}(n)$ și $y_{\mathrm{R_c}}(n)$ sunt semnalele ecou corespunzătoare așa-numitelor canale *stânga* (*left channel*) și *dreapta* (*right channel*).

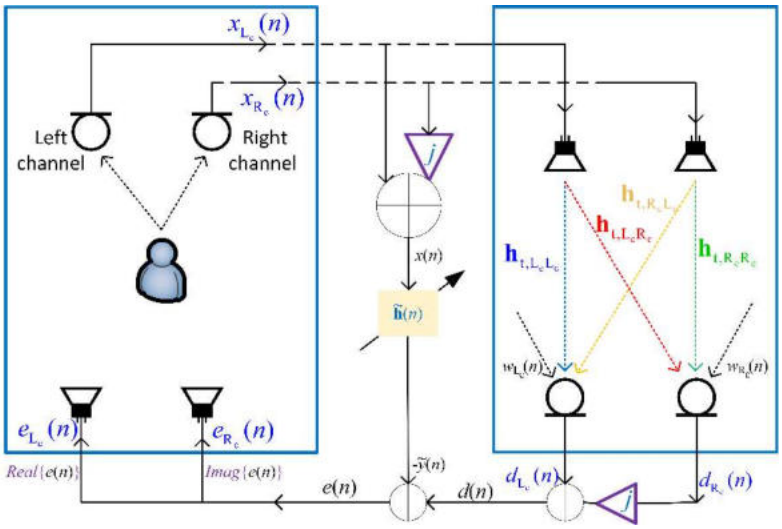

Figura 1. Compensarea ecoului acustic stereofonic cu modelul WL.

2. Soluție de implementare hardware

În continuare este descrisă posibilitatea implementării algoritmului adaptiv de tip RLS-DCD-WL din Tabelul 1, pe dispozitivele de tip FPGA (*Field Programmable Gate Array*), folosind limbajul *VHDL* (*Very High Speed Integrated Circuit Hardware Description Language*). Sunt descrise blocurile componente ale RLS-DCD-WL și modul în care a fost gândită implementarea hardware. Apoi, sunt ilustrate performanțele obținute în comparație cu soluția RLS clasică, și respectiv algoritmul RLS-DCD-WL în precizia mediului de simulare Matlab. În cele din urmă, vor fi indicate resursele ocupate pentru dispozitivul țintă, Virtex 5 XC5VFX70T-ff1136.

Tabelul 1. Algoritm de tip RLS-DCD-WL

Pasul	Ecuație
0	**Inițializare:** $\tilde{\mathbf{h}}(-1)=\mathbf{0}$, $\mathbf{r}(-1)=\mathbf{0}$, $\mathbf{R}_{\tilde{\mathbf{x}}}(0)=\delta^{-1}\mathbf{I}_{2L}$
	Pentru n=1,2,...
1	$\mathbf{R}_{\tilde{\mathbf{x}}}^{(1)}(n)=\lambda\mathbf{R}_{\tilde{\mathbf{x}}}^{(1)}(n-1)+x^*(n)\tilde{\mathbf{x}}(n)$ $\mathbf{R}_{\tilde{\mathbf{x}}}^{(2)}(n)=\lambda\mathbf{R}_{\tilde{\mathbf{x}}}^{(2)}(n-1)+x(n)\tilde{\mathbf{x}}(n)$
2	$\tilde{y}(n)=\tilde{\mathbf{h}}^H(n-1)\tilde{\mathbf{x}}(n)$
3	$e(n)=d(n)-\tilde{y}(n)$
4	$\mathbf{p}_{\tilde{\mathbf{x}}d}(n)=\lambda\mathbf{r}(n-1)-\tilde{\mathbf{x}}(n)e^*(n)$,
5	$\mathbf{R}_{\tilde{\mathbf{x}}}(n)\Delta\tilde{\mathbf{h}}(n)=\mathbf{p}_{\tilde{\mathbf{x}}d}(n)\rightarrow\Delta\tilde{\mathbf{h}}(n)$, $\mathbf{r}(n)$ **(DCD)**
6	$\tilde{\mathbf{h}}(n)=\tilde{\mathbf{h}}(n-1)+\Delta\tilde{\mathbf{h}}(n)$.

Pentru scenariul compensării ecoului acustic în sistemele stereo, cu două difuzoare și două microfoane, sunt necesare la intrarea sistemului cele două semnale audio provenite de la microfoanele de capăt îndepărtat (adică intrările celor patru căi de ecou acustic); acestea formează, conform modelului WL, semnalul complex de intrare $x(n)$. De asemenea, algoritmul adaptiv folosește semnalele dorite (captate de microfoanele de capăt apropiat) care alcătuiesc semnalul

complex $d(n)$. Pentru părțile reale și respectiv imaginare ale semnalelor complexe menționate a fost aleasă o reprezentare în complement față de 2, cu valori în intervalul [-1, 1), având un bit de semn și 15 biți pentru partea fracționară. Aceeași structură este utilizată și pentru coeficienții filtrului adaptiv ($M_b = 16$). De asemenea, pentru algoritmul DCD [14], [15] se folosesc parametrii $N_u = 8$ și $H = 1$. Frecvența de eșantionare a sistemului este 8 kHz.

Simulările efectuate au folosit pentru diverse tipuri de semnale de intrare (zgomot alb și semnal vocal) valorile cele mai mari posibile, fără a se depăși limitele intervalului [-1,1). Astfel, a fost aleasă pentru numerele ce compun matricea $\mathbf{R}_{\tilde{\mathbf{x}}}$ o reprezentare cu câte 32 de biți pentru partea reală, și respectiv cea imaginară. Considerând un bit de semn, se aleg 6 biți pentru partea întreagă, iar restul de 25 vor funcționa pentru partea fracționară. Este necesară o precizie mare a părții fracționare deoarece algoritmul RLS-DCD-WL realizează (cu valori subunitare, în mare parte din cazuri) multiplicări, comparații și deplasări de operanzi cu până la 16 poziții (multe operațiuni de multiplicare folosesc parametrul λ ales mai sus, sau pasul α al DCD-ului - multiplicări cu puteri ale lui 2, care pot fi înlocuite de deplasări).

Implementarea în precizie finită propusă pentru algoritmul de tip RLS-DCD-WL a fost realizată în limbajul *VHDL* (*VHSIC HDL – Very High Speed Integrated Circuit Hardware Description Language*). A fost țintit un ceas cu frecvența minimă de 200 MHz, care înseamnă pentru o rată de 8000 de eșantioane pe secundă un număr de 25000 de perioade de ceas disponibile pentru fiecare iterație a algoritmului adaptiv (intervalul de timp dintre două eșantioane consecutive evaluat în tacte de ceas).

Schema prezentată în Figura 2 ilustrează blocurile funcționale ale algoritmului și pe cele folosite la testarea sa. Toate entitățile componente primesc la intrare semnalul de ceas *iCLK* (folosit pentru prelucrarea sincronă a datelor) și semnalul *iRST* de inițializare a registrelor (bistabilelor) din blocurile componente. Entitatea de top *main_alg.vhd* a algoritmului are la intrare eșantioanele semnalelor complexe $x(n)$ și $d(n)$ (pe câte 32 de biți – parte reală și imaginară), plus semnalul *iDV* (pe un bit) care validează valorile menționate. Un registru pe 15 biți numără până la valoarea 24999 pentru a contoriza tactele de ceas între eșantioane consecutive, pentru a sincroniza acțiunile celorlalte blocuri funcționale și pentru a se păstra succesiunea evenimentelor. Diferitele operațiuni ale algoritmului sunt dictate de anumite valori (sau intervale de valori) ale contorului.

În *main_alg.vhd* sunt incluse trei entități. Prima dintre acestea este blocul care controlează memoria de tip RAM (*Random Access Memory* – memorie cu acces aleator), cu adresare circulară (pe 11 biți), folosită pentru stocarea valorilor complexe ale vectorului $\tilde{\mathbf{x}}(n)$. Aceasta a fost generată cu *Xilinx Core Generator* (ca și celelalte memorii utilizate) și are $2L+2=1026$ de locații, valoari complexe pe 32 de biți. Memoria menționată ocupă pe FPGA trei blocuri de BRAM (*Block RAM*) de câte 18 kb (kilobiți) de date. Două dintre locații au roluri de rezerve; sunt utilizate pentru tactele de ceas care nu presupun scrieri sau citiri din memorie. Programul de control al memoriei $\tilde{\mathbf{x}}(n)$ declanșează citirile pentru toate valorile, de la cel mai nou eșantion până la cel mai vechi (în 1024 tacte de ceas), când la intrarea blocului de control este „ridicat” un bit de comandă a citirii pentru un singur tact de ceas. La ieșire există un port folosit pentru valoarea complexă citită (pe 32 de biți) și un port pe un bit care validează eșantionul citit din memorie. Astfel, pentru citirile de valori, realizate doar pentru grupuri de 1024 numere complexe consecutive, bitul de validare a ieșirii va sta în valoarea ‘1’ pentru 1024 tacte de ceas. În schimb, scrierea în memoria $\tilde{\mathbf{x}}(n)$, cu ajutorul blocului de control, se face doar o singură dată (o singură valoare pe 32 de biți asociată cu încă un bit de comandă), pentru stocarea celui mai nou eșantion de la intrare (un singur tact de ceas este necesar).

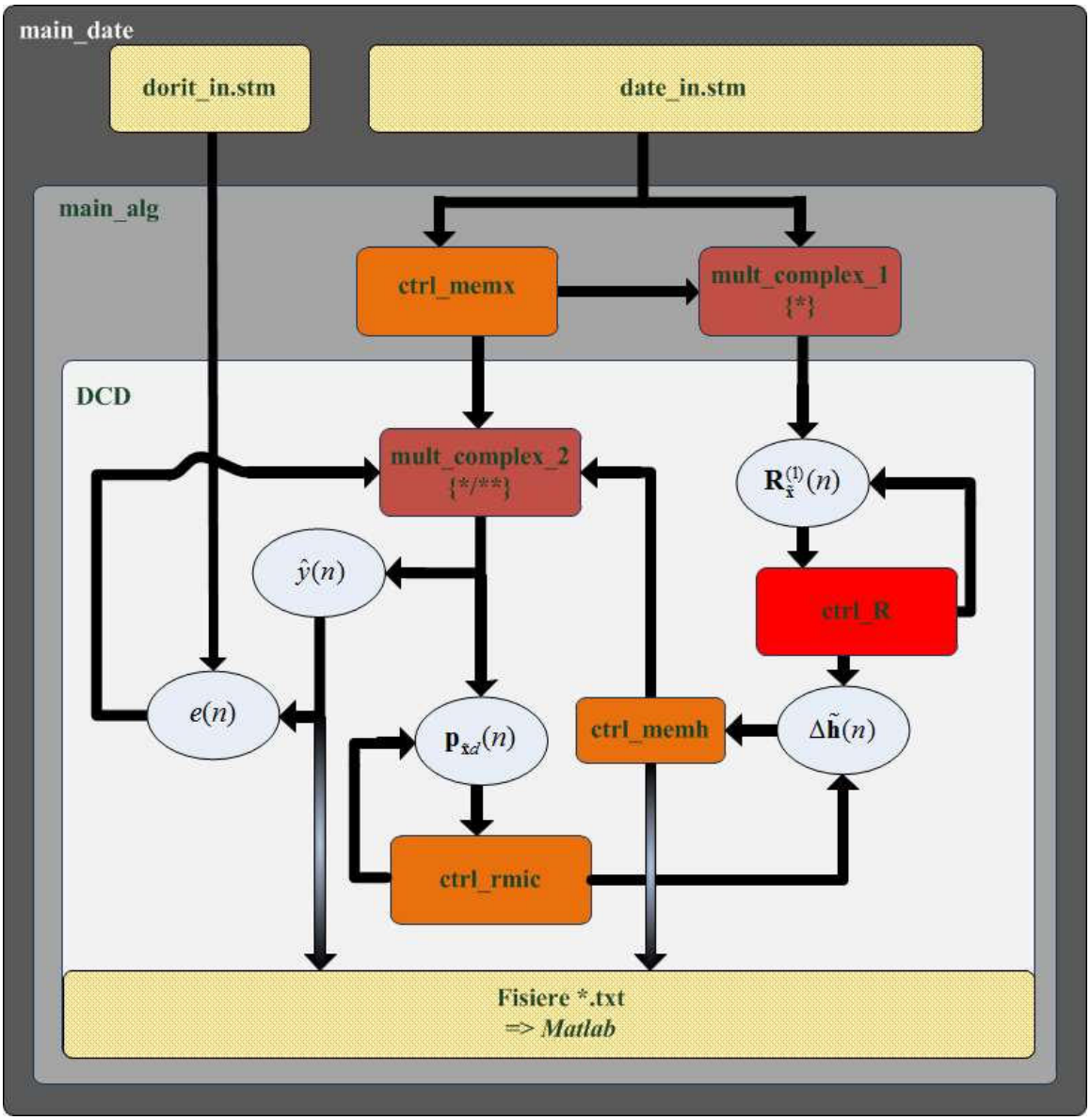

Figura 2. Schema bloc a algoritmului de tip RLS-DCD-WL în precizie finită.

A doua entitate inclusă în *main_alg.vhd* este multiplicatorul complex *MC1* folosit pentru actualizarea matricei $\mathbf{R}_{\tilde{\mathbf{x}}}$. Acesta are patru intrări pentru părțile reale și cele imaginare (pe câte 16 biți) ale celor doi operanzi complecși, plus un bit de comandă, care este propagat până la ieșire. Acesta va semnaliza când rezultatul pe 2x32 de biți (real și imaginar) este o valoare de interes pentru actualizarea $\mathbf{R}_{\tilde{\mathbf{x}}}$. Multiplicatorul lucrează într-o arhitectură de tip *pipe-line*, fiind format din 15 celule de multiplicare. Fiecare celulă deplasează doi din operanzi la stânga, iar pe ceilalți doi la dreapta. De asemenea, celulele transmit între ele bitul de validare a rezultatului și registrele corespunzătoare părții reale, și respectiv imaginare, ale rezultatului final. Operanzii deplasați la stânga sunt adunați în registrele ce compun rezultatul final, dacă biții corespunzători din valorile deplasate la dreapta (de pe poziția celui mai puțin semnificativ bit) sunt '1'. Rezultatele produselor complexe sunt disponibile la ieșirea blocului după 18 tacte de ceas. Astfel, toate produsele

complexe din pasul 1 (pentru o singură actualizare de coloană) se realizează în 1042 tacte de ceas (din cele 25000 disponibile între eșantioane consecutive). Multiplicatorul complex lucrează doar pentru actualizarea primei coloane; a doua coloană poate fi obținută din valorile deja calculate.

Al treilea bloc instanțiat în entitatea principală este cea mai amplă componentă a implementării în precizie finită (*DCD.vhd*). Aceasta are la intrare semnalele *iCLK* și *iRST*, contorul menționat mai sus, eșantioanele cele mai recente ale semnalelor $x(n)$ și $d(n)$, rezultatele produselor efectuate de multiplicatorul complex din *main.vhd* și două semnale pe 1 bit pentru validarea valorilor de la intrare. În cadrul acestei entități sunt instanțiate blocuri de control pentru 3 memorii și un al doilea multiplicator complex de tip *pipe-line*. Pentru sincronizarea operațiunilor efectuate în acest bloc, principalul element ajutător este contorul gestionat în entitatea de top. În scopul obținerii, în procesul de sinteză, a unei perioade de ceas cât mai mică pentru funcționarea sistemului, au fost folosite semnale pe un bit, care să aibă valoarea '1' atunci când trebuie realizate anumite etape al algoritmului. Altfel spus, contorul este interogat asupra unor valori planificate, în funcție de aceste valori biții de sincronizare a blocurilor pot fi ridicați, iar aceștia la rândul lor *decid* (în cadrul altor procese, în structuri de tip *if*) declanșarea funcționării unor etape ale RLS-DCD-WL (de exemplu scrieri sau citiri în memorii). Dacă procedura ar fi realizată prin interogări directe ale contorului menționat, înlănțuirea de logică între două fronturi crescătoare ale ceasului ar presupune timpi mai mari de propagare a semnalelor, ceea ce ar risca neîndeplinirea țintei de ceas de 200 MHz, adică perioada maximă de 5 nanosecunde.

Multiplicatorul complex *MC2*, instanțiat în *DCD.vhd*, funcționează cu rol dublu, folosind 15 celule de calcul, după același principiu ca și în cazul *MC1*. Rezultatele sunt apoi acumulate într-un registru, cu întreaga precizie obținută de multiplicator (2x32 biți). Valoarea finală obținută din filtrarea complexă (adică estimatul complex al ecoului) este apoi trunchiată (sunt eliminați din biții mai puțini semnificativi) pentru a fi scăzută din semnalul dorit complex (pe 2x16 biți). Valoarea semnalului eroare obținut este apoi folosită de același multiplicator complex pentru a doua etapă de funcționare, realizând înmulțirile necesare pentru actualizarea vectorului $\mathbf{p}_{\tilde{\mathbf{x}}d}$.

Prima entitate de control a memoriei inclusă în *DCD.vhd* funcționează pentru $\mathbf{R}_{\tilde{\mathbf{x}}}$. Matricea are în total $(2L)^2=1024^2$ valori complexe reprezentate pe câte 64 de biți, ceea ce înseamnă un total necesar de 67.108.864 biți de memorie. Această resursă nu este disponibilă pe dispozitivele de tip FPGA. Deși pentru studiul de față au fost aleși L=512 coeficienți folosiți pentru modelarea unei căi de ecou acustic din camera apropiată, în implementări valoarea lui L poate ajunge ușor la ordinul miilor. Memoria necesară crește cu lungimea filtrului adaptiv, astfel încât se impune utilizarea unei memorii RAM externe, care trebuie gestionată cu un *controller* (protocol specific). Comportamentul acestei memorii nu este disponibil în mediul de simulare Modelsim, iar scopul acestei lucrări nu este să trateze problema memorării eficiente a matricei $\mathbf{R}_{\tilde{\mathbf{x}}}$, ci de a demonstra faptul că sistemul de ecuații care presupune inversarea acesteia poate fi rezolvat cu o complexitate aritmetică acceptabilă pentru implementările în precizie finită. De aceea, în locul componentei de memorie externă a fost creată o entitate ce modelează prezența unei memorii RAM. Raportul asupra resurselor hardware ocupate pe FPGA va fi dat fără a include *controller*-ul de memorie menționat. Problema resurselor hardware ocupate de $\mathbf{R}_{\tilde{\mathbf{x}}}$ va fi dezvoltată în cercetări viitoare.

Blocul de control al matricei are la intrare semnalele uzuale menționate mai sus (ceasul, reset-ul, contorul). De asemenea, există câte un port intrare/ieșire pentru datele scrise/citite din memorie (pe 64 de biți) și un bit de semnalizare pentru validarea eșantioanelor citite. În plus, mai există semnale de intrare pe un bit prin care sunt semnalizate (cerute) citirea diagonalei principale, a unei coloane sau scrierea de coloană. Toate aceste operațiuni se fac pentru grupuri compacte de

$2L$=1024 valori, cu folosirea a două registre (pe 11 biți – sunt folosite și locații de rezervă) pentru a accesa liniile și respectiv coloanele matricei $\mathbf{R}_{\tilde{\mathbf{x}}}$. Pentru citirea diagonalei principale este necesară ridicarea bitului corespunzător de la intrare pentru un tact de ceas. Blocul de control al matricei numără de la 1 până la $2L$ pe linie și pe coloană pentru a accesa valorile cerute. Aceste eșantioane vor fi disponibile la ieșirea entității prin portul de ieșire menționat mai sus, însoțite de un bit de validare a ieșirii. Într-un mod similar este semnalizată și citirea unei coloane, care este indicată înainte la intrare, printr-un port pe 11 biți, pentru un singur tact de ceas. În acest caz doar indicatorul de linie numără; cel pentru coloană rămâne constant, egal cu valoarea primită la intrarea blocului. Aceeași pereche (port de ieșire plus bit de validare) este folosită pentru eșantioanele citite.

Scrierea valorilor este realizată folosind un port de intrare pentru eșantionul corespunzător și un port pe un singur bit pentru validarea acestuia la fiecare tact de ceas. Pentru 1024 tacte de ceas, bitul de validare a operațiunii de scriere în memorie rămâne în '1' binar. Adresa liniei va crește între 1 și 1024, iar adresa coloanei va indica aceeași valoare, și anume 1. La o examinare atentă a algoritmului de tip RLS-DCD-WL se poate observa că singura operațiune care necesită scriere în $\mathbf{R}_{\tilde{\mathbf{x}}}$ este cea de actualizare a sa. Se poate arăta că este de ajuns calculul primei coloane la fiecare iterație pentru actualizarea matricei, care este hermitică (conjugat-simetrică). De asemenea, se poate arăta că pentru oricare pereche de coloane vecine, numerotate $2k$+1 și $2k$+2 (unde k este număr natural), valorile de pe una dintre coloane se pot regăsi în vecina sa, în formă complex conjugată. Astfel, valorile de pe liniile 1, 2, coloana $2k$+1, se află poziționate în sens invers pe liniile 2 și 1, pe coloana $2k$+2, ș.a.m.d. Ca urmare, pasul respectiv este simplificat la calculul coloanei 1 pentru fiecare iterație, iar pentru fiecare valoare individuală sunt actualizate în matrice celelalte 3 asociate conform celor proprietăților menționate mai sus.

A doua entitate de control a memoriei (*ctrl_rmic.vhd*) gestionează memoria RAM pentru vectorul rezidual $\mathbf{r}(n)$. Fiecare valoare complexă are 64 de biți (parte reală și parte imaginară), iar întreaga memorie conține 1026 de valori, din care două locații au rol de rezerve. Resursele ocupate (raportate de *Core Generator*) sunt 3 blocuri de BRAM de 36 kb. Adresarea se face pe 11 biți, iar valorile propriu-zise ale vectorului se regăsesc între adresele 1 și respectiv 1024. Entitatea de control a $\mathbf{r}(n)$ are la intrare două semnale pe un bit folosite pentru semnalizarea scrierilor și a citirilor de date, care sunt efectuate pentru grupuri de câte 1024 de valori complexe consecutive. Pentru actualizarea $\mathbf{r}(n)$ (pasul 4 al algoritmului RLS-DCD-WL) sunt citite pe rând toate valorile sale, care apoi sunt înmulțite cu factorul de uitare (folosind doar adunări și deplasări). Valorile obținute sunt adunate cu cele de la ieșirea multiplicatorului complex (*MC2*), iar apoi retransmise către blocul de control al memoriei $\mathbf{r}(n)$ pentru a fi scrise.

Alte operațiuni care necesită citiri și scrieri în $\mathbf{r}(n)$ sunt cele implicate în procesul de rezolvare a sistemului auxiliar de ecuații. Algoritmul DCD presupune un număr limitat de actualizări ale vectorului soluție $\Delta\tilde{\mathbf{h}}(n)$ pentru fiecare iterație a algoritmului adaptiv. Pentru această implementare a fost ales N_u=8. Ca urmare, algoritmul caută de exact 8 ori maximul valorilor reale și imaginare, în sensul de valori absolute, în vectorul $\mathbf{r}(n)$. De fiecare dată sunt citite toate valorile din vectorul rezidual și este memorată valoarea asociată maximului, poziția și dacă aceasta este parte reală sau imaginară. Fiecare din cele 8 căutări este realizată cu ajutorul bitului care semnalizează citirea vectorului către blocul de control al memoriei $\mathbf{r}(n)$. După declanșarea acestor citiri, logica din *DCD.vhd* folosește alți biți de semnalizare (câte unul pentru fiecare căutare de maxim) pentru a marca („distinge") perioadele în care citirile din $\mathbf{r}(n)$ au scopul determinării maximului. Este folosit un registru m, pe 5 biți, care poate lua valori până la 16. Acesta

are rolul de a stabili valoarea pasului α (care începe la fiecare iterație a algoritmului adaptiv de la H=1) și implicit bitul asupra căruia se va efectua o eventuală modificare în vectorul soluție. Valoarea din m este inițializată la începutul fiecărei iterații a algoritmului adaptiv. Mai departe, operațiunile care implică α vor fi realizate prin deplasări ale operanzilor, în funcție de valoarea registrului m. [cod Anexa]

A treia memorie controlată din *DCD.vhd*, cu ajutorul blocului denumit *ctrl_memh.vhd*, este folosită pentru memorarea coeficienților filtrului adaptiv. Aceasta lucrează cu 1024+2 locații pe 2x16 de biți (parte reală concatenată cu partea imaginară) și ocupă 3 blocuri BRAM de cîte 18kb. Port-urile standard de intrare (ceas, reset, contor) sunt însoțite de mai multe intrări sau ieșiri pentru citirea sau scrierea din memorie. În primul rând, există un bit pentru comandarea citirii întregii memorii. Această utilizare este importantă pentru calculul estimatului ecoului complex $\hat{y}(n)$. Este de ajuns ca bitul menționat să aibă valoarea 1 un singur tact de ceas pentru a se efectua o întreagă parcurgere a $\tilde{\mathbf{h}}(n)$. La ieșire, blocul are o pereche formată dintr-un port pe 32 de biți pentru eșantioanele citite, împreună cu bitul asociat de validare.

O funcție similară este îndeplinită de un alt bit de semnalizare a citirii. Acesta stă în valoarea 1 tot un tact de ceas și determină citirea unei singure valori din memoria coeficienților filtrului adaptiv. Valoarea adresei de la care se face citirea este disponibilă la intrare într-un port pe 11 biți. De asemenea, există un al treilea bit care validează operațiunea de scriere în memorie, având la intrare în alte două port-uri asociate adresa și valoare scrisă, pe 11 și respectiv 32 biți.

Cele două funcționalități (scriere și citire) ce lucrează cu câte o adresă din memoria $\tilde{\mathbf{h}}(n)$ sunt utile pentru posibilele 8 actualizări ale coeficienților filtrului adaptiv de pe parcursul unei iterații a algoritmului de tip RLS-DCD-WL. Valoarea corespunzătoare din $\tilde{\mathbf{h}}(n)$ este citită, apoi este adunată cu o valoarea pur reală sau pur imaginară, decisă folosind valoarea registrului m într-o instrucțiune *case*. Apoi, se efectuează o scriere la aceeași adresă pentru primii sau ultimii 16 biți (real sau imaginar). Ca urmare, denumirea vectorului soluție $\Delta\tilde{\mathbf{h}}(n)$ devine una generică. În implementare modificările sunt făcute direct în coeficienții filtrului adaptiv. Crearea unui vector de lungime $2L$ ar fi ineficientă, deoarece maxim 8 valori ale sale ar fi nenule. De asemenea, timp inutil, memorie și logică suplimentară ar fi consumate pentru $2L$ citiri, adunări și scrieri.

Considerând funcționalitatea descrisă mai sus, se poate observa că numărul de operații aritmetice care trebuie implementate este redus mult față de algoritmul RLS clasic. O îmbunătățire suplimentară a fost adusă prin alegerea unui factor de uitare convenabil. Eficiența poate fi crescută prin exploatarea proprietăților $\mathbf{R}_{\tilde{\mathbf{x}}}$, adică actualizarea efectuată doar coloanei 1. Pentru matrice poate fi folosită doar un sfert din memoria necesară.

3. Testarea implementării în precizie finită

În vederea testării algoritmului RLS-DCD-WL în precizie finită, a fost folosit programul Modelsim (versiunea SE 6.5). În primele etape ale dezvoltării a fost redactat codul VHDL cu ajutorul editorului Notepad++, componentele de memorie au fost generate cu ajutorul CORE Generator 13.4, iar testarea individuală a blocurilor componente a fost făcută prin vizualizarea formelor de undă a semnalelor în Modelsim. Cu acest scop, au fost introduse seturi mici de valori la intrarea blocurilor și au fost studiate rezultatele obținute după tactele de ceas necesare prelucrării datelor. Figura 3 arată un exemplu de astfel de testare pentru blocul de calcul al produselor

complexe, care lucrează (după cum a fost mai sus menționat) în regim pipe-line. În acest caz, versiunea completă a implementării conține două instanțieri ale acestui multiplicator (*MC1*, *MC2*).

Pe măsură ce algoritmul a devenit din ce în ce mai complex, testarea prin vizualizarea valorilor binare a devenit impracticabilă, astfel încât au fost folosite librăriile *ieee.std_logic_textio.all* și *std.textio.all* pentru scrieri/citiri de fișiere în timpul rulărilor din Modelsim. Astfel, a fost creată o entitate de top, numită *main_date.vhd*, în care a fost instanțiată entitatea principală a algoritmului implementat, alături de alte două blocuri folosite pentru citirea din fișiere a semnalelor de test $x(n)$ și $d(n)$ (eșatioane complexe pe câte 32 de biți – parte reală concatenată cu parte imaginară). De asemenea, au fost instanțiate în entitatea *DCD.vhd* blocuri pentru scrierea în fișiere a valorilor rezultate după anumiți pași ai algoritmului. Toate aceste fișiere au fost citite în Matlab și comparate cu valorile calculate în acest mediu de simulare, pentru a fi depistate erorile de programare. În cele din urmă, s-a ajuns la o variantă funcțională a implementării. Pentru a se examina performanțele obținute, au fost scrise la fiecare iterație în fișier valorile și pozițiile unde sunt efectuate modificări ale coeficienților filtrului adaptiv. Datele au fost citite din nou cu Matlab, unde a fost reconstituită evoluția $\tilde{\mathbf{h}}(n)$ la fiecare iterație.

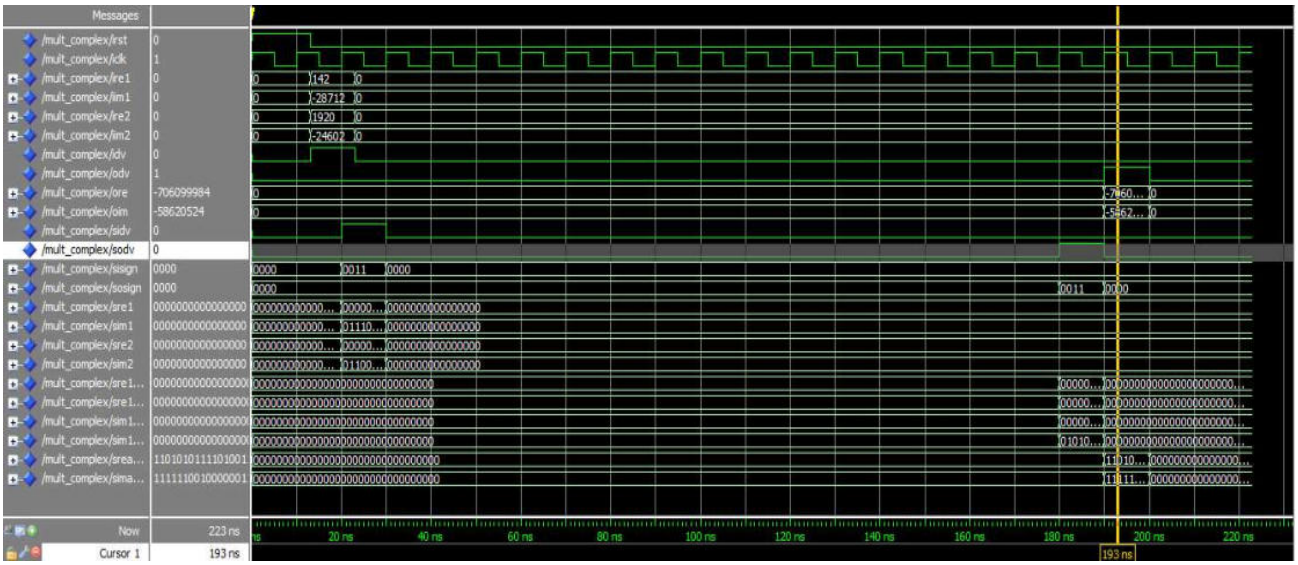

Figura 3. Forme de undă Modelsim.

Pentru algoritmul de tip RLS-DCD-WL în precizie finită, se poate alcătui un raport legat de resursele de logică ce sunt ocupate de codul VHDL descris. A fost folosit în acest scop instrumentul *Xilinx ISE 13.4*. Rezultatele sunt afișate în Tabelul 2, pentru un dispozitiv țintă Virtex 5 xc5vfx70t-ff1136 (*speed grade -1*). Se observă că sunt folosite complet mai mult de jumătate din numărul de perechi ocupate de tipul *LUT-FF* [LUT – *LookUp Table* (tabelă de căutare), FF – *Flip Flop*]. De asemenea, pentru frecvența maximă a ceasului de sistem a fost obținută o valoare de 247.097 MHz, echivalentul unei perioade minime de 4.047 nanosecunde. Frecvența dorită a fost atinsă, cu un plus de 47 de MHz. De asemenea, sunt ocupate în total 9 blocuri de memorie RAM de pe FPGA, dintre care 6 blocuri sunt de 18 kb, iar 3 sunt de 36 de kb de date. În cercetările viitoare asupra algoritmului de tip RLS-DCD-WL, la raportul de arie ocupată, din Tabelul 1, se vor adăuga resursele ocupate de controller-ul pentru memoria externă a matricei de autocorelație și protocolul de comunicație cu codec-ul audio AC97 de pe placa de test ML507.

Tabel 2. Sumar utilizare dispozitiv			
Utilizare Logică	**Folosite**	**Total**	**Utilizare [%]**
Număr de registre *slice*	6781	44800	15%
Număr de tabele de căutare (*Slice LUTs*)	12143	44800	27%
Număr de perechi *LUT-FF* complet folosite	6634	12290	53%
Număr de *buffer*-e intrări/ieșiri (*IOBs*)	82	640	12%
Număr de *buffer*-e interne (*BUFG/BUFGCTRL*)	2	32	6%

Considerând rezultatele obținute, algoritmul adaptiv de tip RLS-DCD-WL reprezintă o soluție atractivă pentru problema anulării ecoului în sistemele stereo, în contextul modelului liniar pe scară largă. Simulările au arătat că algoritmul propus poate avea performanțe bune în implementările hardware, cu o complexitate numerică de ordinul $O(2L)$, mai mică decât algoritmul RLS clasic, care necesită un volum de calcul proporțional cu $4L^2$.

Anexa

Codul de mai jos arată cum este deplasată valoarea reală de la ieșirea matricei $\mathbf{R}_{\tilde{x}}$, prin utilizarea instrucțiunii *case*.

```
shift_Rre: case m is
        when "00001" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-2 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 1);
        when "00010" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-3 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 2);
        when "00011" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-4 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 3);
        when "00100" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-5 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 4);
        when "00101" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-6 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 5);
        when "00110" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-7 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 6);
        when "00111" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-8 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 7);
        when "01000" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-9 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 8);
        when "01001" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-10 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 9);
        when "01010" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-11 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 10);
        when "01011" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-12 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 11);
        when "01100" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-13 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 12);
        when "01101" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-14 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 13);
        when "01110" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-15 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 14);
```

```
        when "01111" => R_out_shiftat_RE <= (Nb_date-1 downto Nb_date-16 =>
R_out_interm_RE(Nb_date-1)) & R_out_interm_RE(Nb_date-2 downto 15);
        when others => R_out_shiftat_RE <= (others => '0');
        end case shift_Rre;
```

Referințe

[1] R. A. Otopeleanu, C. Elisei-Iliescu, C. Paleologu, J. Benesty, C. L. Stanciu, and C. Anghel, "A robust decomposition-based RLS algorithm for echo cancellation applications," in *Proc. IEEE International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, Oct. 2025, 5 pages.

[2] C. Paleologu, J. Benesty, R. A. Otopeleanu, and S. Ciochină, "A nonparametric variable forgetting factor recursive least-squares algorithm," *IEEE Signal Processing Letters*, 5 pages, accepted for publication, Nov. 2025.

[3] S. L. Gay and J. Benesty, Eds., *Acoustic Signal Processing for Telecommunication*. Boston, USA: Kluwer Academic Publisher, 2000.

[4] J. Benesty, T. Gänsler, D. R. Morgan, M. M. Sondhi, and S. L. Gay, *Advances in Network and Acoustic Echo Cancellation*. Berlin, Germany: Springer-Verlag, 2001.

[5] J. Benesty and Y. Huang, Eds., *Adaptive Signal Processing – Applications to Real-World Problems*. Berlin, Germany: Springer-Verlag, 2003.

[6] E. Hänsler and G. Schmidt, *Acoustic Echo and Noise Control – A Practical Approach*. Hoboken, USA: Wiley, 2004.

[7] C. Paleologu, J. Benesty, and S. Ciochină, *Sparse Adaptive Filters for Echo Cancellation*. *Morgan & Claypool Publishers*, USA, 2010 / Springer Nature, Switzerland AG, 2022 (reprint).

[8] J. Benesty, C. Paleologu, T. Gänsler, and S. Ciochină, *A Perspective on Stereophonic Acoustic Echo Cancellation*. Springer-Verlag, Berlin, Germany, 2011.

[9] S. Haykin, *Adaptive Filter Theory*. 4th Edition, Upper Saddle River, USA: Prentice-Hall, 2002.

[10] A. H. Sayed, *Adaptive Filters*. New York, USA: Wiley, 2008.

[11] S. M. Kuo and D. R. Morgan, *Active Noise Control Systems – Algorithms and DSP Implementations*. New York, USA: Wiley, 1996.

[12] C. H. Hansen and S. D. Snyder, *Active Control of Noise and Vibration*. London, U.K.: E&FN Spon, 1997.

[13] C. Stanciu, J. Benesty, C. Paleologu, T. Gänsler, and S. Ciochină, "A widely linear model for stereophonic acoustic echo cancellation," *Signal Processing*, vol. 93, pp. 511-516, Feb. 2013.

[14] Y. V. Zakharov, G. P. White, and J. Liu, "Low-complexity RLS algorithms using dichotomous coordinate descent iterations," *IEEE Trans. Signal Processing*, vol. 56, no. 7, pp. 3150–3161, July 2008.

[15] Y. V. Zakharov and V. H. Nascimento, "DCD-RLS adaptive filters with penalties for sparse identification," *IEEE Trans. Signal Processing*, vol. 61, pp. 3198-3213, June 2013.

Membrii echipei proiectului,

Cristian-Lucian Stanciu

Radu-Andrei Otopeleanu

Nicolae-Cătălin Ristea